From the editors of
A.N.A.L.O.G. Computing

$14.95

THE ANALOG COMPENDIUM

The best ATARI® Home Computer Programs from the first ten issues of A.N.A.L.O.G. Computing Magazine.

THE ANALOG COMPENDIUM

The best ATARI® Home Computer Programs from the first ten issues of A.N.A.L.O.G. Computing Magazine.

From the editors of
A.N.A.L.O.G Computing

A.N.A.L.O.G Magazine Corp.
Worcester, Massachusetts

This book is dedicated to our parents.

Printed in the United States of America

ISBN 0-914177-00-1

10 9 8 7 6 5 4 3 2 1

Table of Contents

Introduction ..... Jon A. Bell 4

Checksum Programs

C:CHECK ..... Tom Hudson 7
D:CHECK2 ..... Istvan Mohos and Tom Hudson 9

Programming Utilities

Unleash The Power of Atari's CPU ..... Ed Stewart 13
Console Button Subroutine ..... Jerry White 15
Trapping Your Atari ..... Donald B. Wilcox 16
Bassnotes in BASIC ..... Jerry White 18
AUDCTL Demo ..... Jerry White 19
Variable Lister ..... Tony Messinna 20
Buncrush ..... Tony Messina 24
Sys/Stat ..... Robert Hartman 30
Faster Character Dumps ..... Joseph Trem 31
Multiprocessing ..... Mark Chasin 34

Graphics

Graphics 9 GTIA Demo 14
Circle Demo 23
Triangle Demo 29
Atari Symbol Demo ..... Craig Weiss 33
Graphics 10 GTIA Demo 36
Moving Players in Basic ..... Jerry White 39
Using DLIs ..... Joseph Trem 42
A Graphics Clipping Routine ..... Tom Hudson 44
3-D Graphs Made Fast and Easy ..... Tom Hudson 47
Sphere Demo 49
Graphic Violence ..... Tom Hudson 50
Graphics 11 GTIA Demo 57
Atari 1020 Printer Demo ..... Tom Hudson 58
Rainbow Demo 65
Swirl Demo 68
Snowflake Demo 109
Graphics 8 Color Demo 114
Moire Demo 122
Circle Radius Demo 139
Pretty Demo 157
Atari 1020 Printer Demos ..... Tom Hudson 58

Disk Utilities

Disk Files: Using NOTE and POINT ..... Jerry White 61
Disk Directory Dump ..... Tony Messina 63
Burp! ..... Charles Bachand 66
The Black Rabbit 2.0 ..... Brian Moriarty 69
Disk Tool 1, Rev. 3 ..... Tony Messina 75

Home Utilities and Education

Home Energy Consumption Analysis ..... Joseph E. Harb 101
Typing Trainer ..... Regena 110

Entertainment

Motorcycle Maze Rider ..... Charles Bachand 117
Dino Battle ..... Art V. Cestaro III 119
Triple Threat Dice ..... Michael A. Ivins 123
Bicycle ..... Dan Devos 126
Color Slot Machine ..... Micheal A. Ivins 128
Halls Of The Leprechaun King ..... Keith Evans and Ted Atkinson 135
Stuntman ..... Stephen Pogatch 140
Dungeons and Dragons Character Generator ..... Bob Curtin 143
Dungeons and Dragons Housekeeping 2 ..... Bob Curtin 149
Thunder Island ..... Craig Patchett 158
Maniac! ..... Rick Messner 161
Harvey Wallbanger ..... Charles Bachand 166
Fill 'er Up II ..... Tom Hudson 175

Introduction

Lee Pappas and Mike DesChenes of Worcester, Massachusetts bought their first Atari home computers in November of 1979. Their first year as Atari owners was spent developing their programming skills, blowing away Zylons, and tearing out their hair because of the lack of support for their new machines. Where was the information that they (and thousands of other Atari owners) so desperately needed? From the seeds of this frustration, Atari users groups began sprouting up all over the country. New Atari owners started pooling their knowledge and linking their collective consciousness via bulletin board systems. Nevertheless, there was no dedicated publication for their systems, no single source of information that could link Atari owners together. Lee and Mike decided to do something about it.

In November of 1980, they started an Atari-only publication called **A.N.A.L.O.G. 400/800 Magazine**. The first issue was only 40 pages long, and had a modest print run of 4000 copies. Gratifyingly, it sold out. Almost three years and 15 issues later, **A.N.A.L.O.G Computing** has grown to over 160 pages, with a world-wide distribution of over 80,000 copies — and no end in sight.

With the smaller print runs of the earlier issues, we had virtually no returns. Supplies of back issues sent out from our editorial offices were quickly exhausted. Reprints were done of issues 2, 3 and 4. These sold out, too. Compounding the problem was the fact that the newcomers to **A.N.A.L.O.G.** wanted any and all issues previous to the first one they purchased. The later the issue, the more back issues they needed. The solution? This book.

The A.N.A.L.O.G. Compendium is not intended as an all-encompassing primer on Atari programming. Although many of the programs included here were originally written as tutorials, it was never our intention to publish a textbook. **The A.N.A.L.O.G. Compendium** is presented solely as a collection of programs to benefit those who missed out on our first ten issues. Some of the programs here have been revised and improved since they originally appeared in the magazine. We have also included several programs never before published.

Whether you're interested in utilities, tutorials or games, we hope you enjoy our first book.

Jon A. Bell
Managing Editor
A.N.A.L.O.G. Computing

CHECKSUM PROGRAMS

Important!

All of the programs in **The A.N.A.L.O.G. Compendium** were listed from working copies of the program in order to minimize errors. However, there is a strong possibility of readers mis-typing programs, especially when entering lengthy listings. Before you type in any of these programs, it is strongly advised that you read pages 7-10. (C:CHECK and D: CHECK 2). These programs will assist you in checking for typing errors when entering in programs from **The A.N.A.L.O.G. Compendium.**

C:CHECK

16K Cassette

by Istvan Mohos and Tom Hudson

When typing programs into your computer from the **A.N.A.L.O.G. Compendium**, there is always a chance of making a mistake. C:CHECK will help you find such errors very easily. Type in the accompanying program and SAVE it. Follow the instructions below to check C:CHECK as you would any other program.

CHECKing your typing.

1. Type in the program listing from the **Compendium**. Visually check it for obvious errors (missing lines, etc).

2. LIST the program to be checked to cassette. Use the command:

LIST "C:"

3. LOAD C:CHECK and RUN it.

4. C:CHECK will ask you if you want the output to go to the screen or printer. Type S for screen or P for printer and press RETURN.

5. C:CHECK will ask for an issue number. For the **Compendium,** type 99 and press RETURN. If you read **A.N.A.L.O.G. Computing Magazine**, you can use C:CHECK to check the programs in each issue. Just type the issue number and press RETURN.

6. Position the tape to the beginning of the program to be checked and press PLAY on the program recorder. Press RETURN.

7. C:CHECK will begin reading the program from tape and generate a checksum table. This data should match the "CHECKSUM DATA" printed after the program listing you are checking. The following example shows how to check for errors.

Sample Compendium CHECKSUM DATA:

```
10 DATA 34,455,234,22,55,38,93,45,114,
285,633,442,453,23,31,2957
160 DATA 82,94,64,73,347,199,287,84,15
6,368,59,40,98,9,342,2302
310 DATA 65,356,101,25,547
```

Sample C:CHECK output:

```
10 DATA 34,455,234,22,55,38,244,45,114
,285,633,442,453,23,31,3108
160 DATA 82,94,64,73,347,199,287,84,15
6,368,59,40,98,9,342,2302
310 DATA 65,101,34,200
```

Each line of the program being checked has its own checksum value. If any characters in the line are incorrect, the checksum value will be different from the corresponding value in the **Compendium**. The checksum data is set up so that there are 15 checksum values in each line with the 16th value containing the total of the checksums.

The line number of the checksum line tells which line number is first in the checksum group. In the example above, the first line checked in the first checksum line is 10, and its checksum is 34. The first line checked in the second checksum line is 160, and its checksum is 82. The first line checked in the third checksum line is 310, and its checksum is 65.

Let's assume the CHECKSUM DATA above was listed in the **Compendium**, and you typed in the program and checked it with C:CHECK.

The first thing to do would be to look at the total of the values in the first line. This value should be 2957, as shown in the **Compendium** CHECKSUM DATA. However, in the results in the C:CHECK output, the total is 3108. This means that there is an error in the 15 checksum values in this line. Comparing the **Compendium** checksums to the C:CHECK output, we find that the seventh checksum is 244 in the C:CHECK data, and should be 93. This means that there is an error in the seventh line of the program. Note the error and continue checking. The rest of the line is correct, so we go on to the second line.

Now we check the total of the second line of checksum data. The total of 2302 in our C:CHECK

data matches the total in the **Compendium**, so we can go on to the third checksum line.

The third checksum line is different from the others in that it only checks four lines. This is because it is at the end of the program, and the program did not have an even multiple of 15 lines. The line is checked the same as the others. As you can see, the total of the line should be 547, but is only 200 in the C:CHECK data. Looking at the C:CHECK output, you will notice that there is one less checksum value (the 356 in the **Compendium** checksum data). This means that the first line in the program after line 310 is missing. The last checksum in this line is also incorrect. It is a 34 and should be 25. This means that the third line after line 310 in the program is incorrect.

To summarize, there were 3 errors in the program we checked. Two errors were caused by mistakes in the lines, and a third appeared because a whole line was missing.

Once you have noted all errors, type NEW and press RETURN. This erases the C:CHECK program. Next, bring the program being checked into memory by positioning the tape and typing:

ENTER "C:"

If the program had errors, correct the lines in error. If there were no errors, the program is correct and ready to run. □

```
100 REM CHECK DEBUGGING AID
    BY  ISTVAN MOHOS
110 REM VERSION 2 MODS AND CASSETTE
120 REM VERSION BY TOM HUDSON
130 GRAPHICS 0:? :? "This run will LIS
T data statements    to the screen or
printer."
140 ? :? "This DATA is created by eval
uating    each character of a user pro
gram,     LISTed to tape.":?
150 DIM OUT$(1),I$(128),CR$(1)
160 ? "OUTPUT TO SCREEN OR PRINTER";:I
NPUT OUT$:IF OUT$<>"S" AND OUT$<>"P" T
HEN 160
170 IF OUT$="S" THEN OPEN #2,8,0,"E:":
GOTO 200
180 CLOSE #2:? "READY PRINTER AND PRE
SS RETURN";:INPUT CR$
190 TRAP 180:OPEN #2,8,0,"P:"
200 ? :? "ENTER ISSUE NUMBER";:TRAP 20
0:INPUT ISSUE
210 ? :? "READY TAPE AND PRESS RETURN"
;:OPEN #1,4,0,"C:":? :?
220 Z=0:LINECOUNT=Z:PLIN=Z:X=2
230 TRAP 340:INPUT #1,I$:LINECOUNT=LIN
ECOUNT+1:LINUM=VAL(I$(1,5))
240 NLCK=NLCK+1:IF NLCK>1 AND NLCK<16
THEN 290
250 IF LINECOUNT=1 THEN 280
260 ? #2;TOTAL:NLCK=1
270 IF OUT$="S" THEN PLIN=PLIN+1:IF PL
IN=18 THEN ? "PRESS RETURN TO CONTINUE
";:INPUT CR$:PLIN=0
280 TOTAL=Z:? #2;LINUM;" DATA ";
290 CHKSUM=Z:IF ISSUE>9 THEN X=2
300 FOR I=1 TO LEN(I$):PRODUCT=X*ASC(I
$(I,I)):CHKSUM=CHKSUM+PRODUCT:X=X+1:IF
 X=4 THEN X=1
310 NEXT I:CHKSUM=CHKSUM+X*155:X=X+1:I
F X=4 THEN X=1
320 CHKSUM=CHKSUM-1000*INT(CHKSUM/1000
)
330 ? #2;CHKSUM;",";:TOTAL=TOTAL+CHKSU
M:GOTO 230
340 CLOSE #1:IF LINECOUNT=Z THEN 370
350 ? #2;TOTAL
360 CLOSE #2:END
370 ? "":? "Your typed-in program wa
s not properlyLISTed to tape."
380 ? :? "Please LIST your program to
tape, thenRUN ";CHR$(34);"CHECK";CHR$(
34);" again.":CLOSE #2:CLR :END
```

•

CHECKSUM DATA

(See pgs. 7-10)

```
100 DATA 198,759,11,135,191,594,198,80
6,763,467,931,100,465,572,107,6297
250 DATA 764,922,11,168,375,783,304,25
9,534,890,875,136,732,361,7114
```

•

D:CHECK 2

16K Disk

by Istvan Mohos and Tom Hudson

When typing programs into your computer from the **A.N.A.L.O.G. Compendium**, there is always a chance of making a mistake. D:CHECK2 will help you find such errors very easily. Type in the accompanying program and SAVE it. Follow the instructions below to check D:CHECK2 as you would any other program.

CHECKing your typing.

1. Type in the program listing from the **Compendium**. Visually check it for obvious errors (missing lines, etc.).
2. LIST the program to be checked to disk. Use the command:

LIST "D:progname"

3. LOAD D:CHECK2 and RUN it.
4. D:CHECK will ask for a filename. Respond:

D:progname

and press RETURN.

5. D:CHECK2 will ask for an issue number. For the **Compendium**, type 99 and press RETURN. If you read **A.N.A.L.O.G. Computing Magazine**, you can use D:CHECK2 to check the programs in each issue. Just type the issue number and press RETURN.
6. D:CHECK2 will execute. The screen will go black in order to speed up the program.
7. When D:CHECK2 finishes, it will display final instructions. At this time you should type NEW and press RETURN.
8. When D:CHECK2 executed, it created a BASIC file on disk called BUG. ENTER it into your computer with the command:

ENTER "D:BUG"

This file should match the "CHECKSUM DATA" printed after the program listing you are checking. The following example shows how to check for errors.

Sample Compendium CHECKSUM DATA:

```
10 DATA 34,455,234,22,55,38,93,45,114,
285,633,442,453,23,31,2957
160 DATA 82,94,64,73,347,199,287,84,15
6,368,59,48,98,9,342,2302
310 DATA 65,356,101,25,547
```

Sample "D:BUG" CHECKSUM DATA:

```
10 DATA 34,455,234,22,55,38,244,45,114
,285,633,442,453,23,31,3108
160 DATA 82,94,64,73,347,199,287,84,15
6,368,59,48,98,9,342,2302
310 DATA 65,101,34,200
```

Each line of the program being checked has its own checksum value. If any characters in the line are incorrect, the checksum value will be different from the corresponding value in the **Compendium.** The checksum data is set up so that there are 15 checksum values in each line with the 16th value containing the total of the checksums.

The line number of the checksum line tells which line number is first in the checksum group. In the example above, the first line checked in the first checksum line is 10, and its checksum is 34. The first line checked in the second checksum line is 160, and its checksum is 82. The first line checked in the third checksum line is 310, and its checksum is 65.

Let's assume the CHECKSUM DATA above was listed in the **Compendium**, and you typed in the program and checked it with D:CHECK2.

The first thing to do would be to look at the total of the values in the first line. This value should be 2957, as shown in the **Compendium** CHECKSUM DATA. However, in the results in the BUG file, the total is 3108. This means that there is an error in the 15 checksum values in this line. Comparing the **Compendium** checksums to the BUG checksums, we find that the seventh checksum is 244 in the BUG data, and should be 93. This means that there is an error in the seventh line of the program. Note the error and continue checking. The rest of the line is correct, so we go on to the second line.

Now we check the total of the second line of checksum data. The total of 2302 in our BUG file matches the total in the **Compendium**, so we can go on to the third checksum line.

The third checksum line is different from the others in that it only checks four lines. This is because it is at the end of the program, and the program did not have an even multiple of 15 lines. The line is checked the same as the others. As you can see,

the total of the line should be 547, but is only 200 in the BUG file. Looking at the BUG file, you will notice that there is one less checksum value (the 356 in the **Compendium** checksum data). This means that the first line in the program after line 310 is missing. The last checksum in this line is also incorrect. It is a 34 and should be 25. This means that the third line after line 310 in the program is incorrect.

To summarize, there were 3 errors in the program we checked. Two errors were caused by mistakes in the lines, and a third appeared because a whole line was missing.

Once you have noted all errors, type NEW and press RETURN. This erases the D:CHECK2 program. Next, bring the program being checked into memory by typing:

ENTER "D:progname"

If the program had errors, correct the lines in error. If there were no errors, the program is correct and ready to run. □

```
10 REM CHECK DEBUGGING AID
   BY  ISTVAN MOHOS
20 REM VERSION 2 MODS BY TOM HUDSON
30 GRAPHICS 0:? :? "This run will LIST
 data statements    with the name: BUG
, to the disk."
40 ? :? "The BUG DATA is created by ev
aluating each character of a user prog
ram,     LISTed to disk.":?
50 DIM FI$(15)
60 CLOSE #1:? "ENTER FILENAME";:INPUT
FI$
70 PIK=PEEK(559):Z=0:REM constants
80 ? :? "ENTER ISSUE NUMBER";:TRAP 80:
INPUT ISSUE
90 TRAP 60:OPEN #1,4,0,FI$
100 ON X GOTO 180,280
110 ? "K":? "DISABLING SCREEN...STAND
BY...":FOR I=1 TO 800:NEXT I:POKE 559,
Z:REM debug before poking
120 LINECOUNT=Z:DIM I$(126)
130 TRAP 150:INPUT #1;I$:LINECOUNT=LIN
ECOUNT+1
140 GOTO 130
150 CLOSE #1:Q=INT(LINECOUNT/15):DIM C
(LINECOUNT),R(Q),S$(5):IF (LINECOUNT=Z
 OR I$="") THEN 530
160 IF ASC(I$(1,1))<48 OR ASC(I$(1,1))
>57 THEN 530
170 X=1:GOTO 90
180 RANGE=Z:LINE=Z:FOR I=1 TO 5:S$(I,I
)=" ":NEXT I
190 COUNT=Z
200 INPUT #1;I$:T=1:COUNT=COUNT+1
210 IF I$(T,T)<>" " THEN S$(T,T)=I$(T,
T):T=T+1:GOTO 210
220 LINE=VAL(S$)
230 R(RANGE)=LINE:RANGE=RANGE+1
240 TRAP 270:INPUT #1;I$
250 COUNT=COUNT+1:IF COUNT=15 THEN 190
260 GOTO 240
270 CLOSE #1:X=2:GOTO 90
280 FOR I=1 TO LINECOUNT:CHECKSUM=Z
290 GET #1,NUMBER:PRODUCT=X*NUMBER:CHE
CKSUM=CHECKSUM+PRODUCT:X=X+1:IF X=4 TH
EN X=1
300 IF NUMBER=155 THEN 320
310 GOTO 290
320 CHECKSUM=CHECKSUM-1000*INT(CHECKSU
M/1000):C(I)=CHECKSUM:IF ISSUE>9 THEN
X=2
330 NEXT I
340 CLOSE #1:OPEN #1,8,0,"D:BUG":LINE=
R(Z):ITEM=Z
350 COUNT=15:TOTAL=Z:IF LINECOUNT<15 T
HEN COUNT=LINECOUNT
360 PRINT #1;LINE;" DATA ";
370 FOR I=1 TO COUNT:DATUM=C(15*ITEM+I
):PRINT #1;DATUM;",";:TOTAL=TOTAL+DATU
M:NEXT I
380 PRINT #1;TOTAL
390 ITEM=ITEM+1:LINECOUNT=LINECOUNT-15
:IF LINECOUNT<1 THEN 420
400 LINE=R(ITEM)
410 GOTO 350
420 CLOSE #1:POKE 559,PIK
430 ? "KTo check BUG data against pri
nted     data statements, type NEW. Th
en type:"
440 ? "ENTER ";CHR$(34);"D:BUGRETURN .
                    Type LIST after the
READY prompt."
450 ? :? "The line number of each data
 statementcoincides with the first lin
e of the"
460 ? "user program which the data sta
tement evaluates."
470 ? "Numbers within each data statem
ent     represent consecutive lines of
the     user program."
480 ? "The last number is the total."
490 ? :? "Check the last number of eac
h state-  ment against the printed ver
sion;"
500 ? "only in case of a discrepancy c
heck    each number in the data stateme
nt."
510 ? "Make note of the lines containi
ng the bugs. Then ENTER ";CHR$(34);"D:
yourprogRETURN"
520 ? "to make the corrections.":END
530 POKE 559,PIK:? "K":? "Your typed-
in program was not properlyLISTed to d
isk."
540 ? :? "Please LIST your program to
disk, thenRUN ";CHR$(34);"D:CHECK";CHR
$(34);" again.":CLR :END
```

•

CHECKSUM DATA

(See pgs. 7-10)

```
10 DATA 44,815,767,524,686,389,806,850
,86,721,921,593,591,704,974,9471
160 DATA 482,125,389,696,567,797,442,5
61,230,89,717,216,943,541,299,7094
310 DATA 719,711,741,427,244,435,288,5
84,553,441,711,499,803,322,515,7993
460 DATA 246,684,406,232,123,700,480,7
74,500,4145
```

•

PROGRAMMING UTILITIES

UNLEASH THE POWER OF ATARI's CPU

by Ed Stewart

Would you like to get as much as a 30% increase in speed from your ATARI 6502 CPU? Would you also like to get this benefit without any additional capital expense? If your answer is no, you probably don't like apple pie, either...but if your answer is yes, read on, and I will tell you how to accomplish such a feat with one simple BASIC POKE in the right place.

First, a little background information about one of the many things going on inside your ATARI computer. The particular thing that I want you to know about is how display information reaches your TV screen. There is a hardware chip called ANTIC that has most of the responsibility for seeing that the display gets to your TV screen. ANTIC does this by operating independently from the main 6502 CPU in your computer. ANTIC is, in fact, a primitive CPU in its own right. It executes a program which is located in RAM, just as the 6502 executes a program in RAM or ROM. We can therefore call the ATARI a multiprocessing computer, since more than one CPU may be active at any time.

A peculiar and somewhat unfortunate thing happens when a multiprocessing system such as the ATARI is actively executing instructions — both CPUs desire access to memory simultaneously. The two CPUs cannot both access memory at the same time, so one must wait until the other completes its access request. This memory access conflict is common to all computers containing more than one CPU — from micros to macros — and is generally not something to be concerned about.

The ANTIC chip fetches its data from memory using a technique called "Direct Memory Access" or DMA. Whenever this memory fetch is occurring, the 6502 is temporarily halted. DMA is said to be "stealing" a portion of the computer's available time, called a cycle. There are 1,789,790 cycles of computer time available per second. If DMA had not "stolen" that cycle of computer time, the 6502 would not have been halted and, therefore, would have finished its program instructions sooner. It is only logical to conclude that the more this DMA activity occurs on behalf of the ANTIC chip, the more our 6502 will be slowed down.

The ANTIC chip re-displays the entire TV display 60 times each second. During this period, many computer cycles are stolen from the 6502. During each of these 60 times, the ANTIC chip also "interrupts" the 6502 and causes it to perform such tasks as updating various software timers and reading game controllers (joysticks and paddles). When the 6502 finishes what it must do in response to the ANTIC "interrupt," it may continue with what it was doing previous to being sidetracked by ANTIC. You should be getting the picture by now that, although ANTIC is indispensable, it causes a slowdown in the 6502 CPU. But how much?

I wrote a simple BASIC program for my ATARI 800 in an attempt to answer this question. A FOR/NEXT loop was executed 100,000 times with no intervening statements as follows:

```
20 FOR I=1 TO 100000:NEXT I
```

The first thing to measure was how long this loop executes with no ANTIC DMA active. A POKE 559,0 turned DMA off, and the TV screen went black. A POKE 559,34 turned DMA back on, and the original display was restored. The FOR/NEXT loop was executed in graphics modes 0-8 *with* DMA active, and the executive times were observed as shown in **Table 1**. The execution times *with* DMA increased from as little as 10% for graphics 3 to as much as 47% for graphics 8.

It is easy to see that — if you do a lot of number crunching and you don't need the TV screen, software timers or game controllers — then turn off the ANTIC DMA for a while, and you'll get your answer back sooner. It is also apparent from the chart below that your programs will execute faster if you are using graphics modes 3, 4, or 5.

I hope you have learned a little bit more about the ATARI computer and how the ANTIC DMA interferes with the 6502 CPU. You may someday be able to leash that latent power within during a computer chess tournament, and — when someone asks how in the world you did it — you can smile and say, "me and my DMA." □

GRAPHICS MODE	EXECUTION SECONDS	% INCREASE (over no-DMA)
NO DMA	148	
GRAPHICS 0	216	46
GRAPHICS 1	188	27
GRAPHICS 2	186	26
GRAPHICS 3	163	10
GRAPHICS 4	164	11
GRAPHICS 5	167	13
GRAPHICS 6	173	17
GRAPHICS 7	185	25
GRAPHICS 8	218	47

Graphics 9 GTIA Demo

```
10 REM GRAPHICS 9 GTIA DEMO (OVAL)
20 REM
30 GRAPHICS 9
40 C=0:SETCOLOR 4,C,0
50 FOR X=0 TO 39
60 FOR Y=0 TO 95
70 XW=39-X:YW=95-Y:COLOR INT(SQR(XW*XW
+YW*YW)/6.5)
80 PLOT X,Y
90 PLOT 79-X,Y
100 PLOT X,191-Y
110 PLOT 79-X,191-Y
120 NEXT Y
130 NEXT X
140 C=C+1:IF C>15 THEN C=0
150 SETCOLOR 4,C,0
160 FOR TIME=1 TO 500:NEXT TIME
170 GOTO 140
```

CHECKSUM DATA
(See pgs. 7-10)

```
10 DATA 682,253,174,886,290,293,938,61
7,923,418,747,766,767,154,494,8402
160 DATA 433,716,1149
```

CONSOLE BUTTON SUBROUTINE

16K Cassette or Disk

by Jerry White

The ATARI BASIC Reference Manual describes decimal location 53279 as "Console switches" (bit 2= Option; bit 1= Select; bit 0= Start. POKE 53279,0 before reading. 0= switch pressed).

The would-be BASIC programmer has got to be a bit confused after reading the above. In BASIC, you normally don't think about bit settings, and the beginner has a long way to go before he or she will have to worry about such things.

The point is that a BASIC PEEK (53279) will tell you which console buttons, if any, are pressed. You can see how pressing one or more buttons changes the value of that location with a one-line program. Enter line 10 below, then type RUN and RETURN. Watch the screen as you press the various console buttons, then press BREAK to abort.

```
10 PRINT PEEK(53279):GOTO 10
```

Now for a somewhat more useful demonstration, enter the CONSOLE BUTTON SUBROUTINE. Note that although it is a subroutine, it has been set up so that it will run without any additional code. Of course, you could access it from your own program with a GOTO 30000.

This routine provides the user with three options. It will allow you to RERUN THIS PROGRAM (the program currently in RAM), RETURN TO BASIC (which is a fancy way to say END), or RUN A MENU PROGRAM from diskette. Naturally, you could change these options to whatever your own program requires. The START button is used to execute the option that is currently displayed, using inverse video. Pressing the OPTION or SELECT buttons will change the previously highlighted option back to normal video and highlight the next option. When the desired option is highlighted, the START button is used to say "DO IT."

Since this is a routine you will modify and include in many of your own programs, it should be LISTed onto cassette (LIST "C:") or disk (LIST "D: BUTTON. LST," 30000, 30170). When you want to include it as part of your own program currently in RAM, ENTER "C": from cassette or ENTER "D: BUTTON. LST" from disk. □

```
0 REM CONSOLE BUTTON SUBROUTINE
1 REM BY JERRY WHITE 6/5/82
30000 GRAPHICS 0:POKE 752,1:POKE 710,4
8:POKE 82,2:POKE 201,9
30010 ? "{↓↓↓↓ Use the OPTION or SELECT
 button to":? :? " highlight your choi
ce below, then"
30020 ? :? "{ press the start button."
:FOR ME=0 TO 8:POKE 53279,ME:NEXT ME:G
OSUB 30100:SEL=11
30030 POSITION SEL,SEL:? "RERUN THIS P
ROGRAM"
30040 BUTTON=PEEK(53279):IF BUTTON=7 T
HEN 30040
30050 GOSUB 30140:IF CHOICE=6 THEN 301
10
30060 SEL=SEL+2:IF SEL>15 THEN SEL=11:
GOSUB 30100:GOTO 30030
30070 IF SEL=13 THEN GOSUB 30100:POSIT
ION 11,SEL:? "RETURN TO BASIC":GOTO 30
040
30080 IF SEL=15 THEN GOSUB 30100:POSIT
ION 11,SEL:? "RUN MENU PROGRAM":GOTO 3
0040
30090 GOTO 30040
30100 POSITION 11,11:? "RERUN THIS PRO
GRAM":? :? ,"RETURN TO BASIC":? :? ,"R
UN MENU PROGRAM":RETURN
30110 TRAP 30000:POKE 201,10:IF SEL=15
 THEN ? "{":? :? ,"LOADING MENU":RUN "
D:MENU":TRAP 40000
30120 IF SEL=13 THEN GRAPHICS 0:? :? "
BASIC":? "IS";:POKE 752,0:TRAP 40000:E
ND
30130 TRAP 40000:RUN
30140 GOSUB 30170
30150 CHOICE=BUTTON:BUTTON=PEEK(53279)
:IF BUTTON<>7 THEN 30150
30160 GOSUB 30170:RETURN
30170 FOR ME=0 TO 8:POKE 53279,ME:NEXT
 ME:RETURN
```

•

CHECKSUM DATA

(See pgs. 7-10)

```
0 DATA 874,802,472,699,670,842,127,197
,560,185,623,205,215,413,935,7819
30130 DATA 746,141,237,526,717,2367
```

•

TRAPPING YOUR ATARI

by Donald B. Wilcox

It is often frustrating to be forced to restart a program because an inadvertent error caused the program to crash. ATARI BASIC provides a special word — TRAP — that often can be used to prevent a program from ending before intended. Many errors are subject to automatic correction or compensation through a little extra effort on the part of the programmer.

If you are not yet familiar with the TRAP statement, the following examples show how to use it to detect INPUT errors. These occur when the user of a program types invalid values into a numeric variable.

```
10 INPUT X
20 PRINT X
30 GOTO 10
```

In the above listing, typing a non-numeric response to the INPUT statement in line 10 (such as accidentally pressing return with no number entered) will result in an "ERROR-8 AT LINE 10" message. By adding a TRAP statement, this problem can be avoided completely.

```
10 TRAP 10:INPUT X
20 PRINT X
30 GOTO 10
```

In the slightly modified example above, if an input error occurs, the TRAP statement will catch the error and go back to line 10 to try the INPUT again.

After perusal of these five examples, you should be able to understand how to make your programs less vulnerable to errors that prematurely end your program.

Listing 1 — If you mistakenly create a new file using a file name that already exists, you will destroy the already existing file. No error message will warn you of the impending disaster. **Listing 1** will prevent this.

Listing 2 — If you try to OPEN a non-existent file, you will get an error message 170 and your program will crash. This can be prevented by using **Listing 2**.

Listing 3 — If you try to input data from a disk file beyond the end-of-file, you will get an error message 136, and your program will terminate. You may not always know beforehand where the file data ends, so an automatic end-of-file trap can be programmed easily to prevent the error. **Listing 3** solves this problem.

Listing 4 — You forgot to turn on your printer or interface unit and get error message #138. If you attempt to use the Continue command after you turn on the correct unit, your program will continue beginning at the line number that follows the line that caused the error. Often this can create erroneous results (not always detected), because the instructions on the line that caused the error may not have been executed correctly before the error.

Listing 5 — You are reading in data with a READ statement and you do not want to use an end-of-data dummy value as a flag, nor do you want to count the entries to determine when all the data has been read. **Listing 5** demonstrates a simple method to prevent error #6 (Out Of Data) from prematurely terminating your program.

Finally, for those of you who are relatively new to ATARI BASIC, there are several locations (addresses) that you may PEEK to find out which error occurred and which line caused the error. Location 195 contains the error number. Locations 186 and 187 contain the line number where the error occurred, low byte, high byte, respectively. To display this information on your screen, use the following statements:

```
10 REM DISPLAY ERROR NUMBER
20 REM AND LINE NUMBER OF ERROR
30 PRINT PEEK(195);" AT LINE ";PEEK(18
6)+PEEK(187)*256
```

Listing 1.

```
100 ? "ß":CLR :REM CLEAR SCREEN AND VA
RIABLES
110 REM PREVENT ERASURE OF PROGRAM ALR
EADY STORED ON DISK
120 DIM ATRAP$(6),A$(124),NAME$(8),FIL
E$(10)
130 REM SET UP DISK SUFFIX 'D:' FOR FI
LE NAME.  IOCB IS FILE(DEVICE) NUMBER
140 FILE$="D:":IOCB=2:IN=4:GNU=8
150 REM GNU=8 IS THE OUTPUT MODE
160 SET=160:CLOSE #IOCB:IF ATRAP$="SPR
UNG" THEN PRINT " FILE NAME DID NOT PR
EVIOUSLY EXIST":GOTO 200
170 TRAP SET:PRINT "ENTER FILE NAME"
180 INPUT NAME$:FILE$(3)=NAME$:ATRAP$=
"SPRUNG":OPEN #IOCB,IN,0,FILE$
190 PRINT FILE$;" ALREADY EXISTS":? "U
SE A DIFFERENT NAME":CLOSE #IOCB:GOTO
170
200 OPEN #IOCB,GNU,0,FILE$
210 PRINT FILE$;" OPENED SUCCESSFULLY"
220 CLOSE #IOCB
```

Listing 2.

```
100 PRINT "ß":CLR :REM CLEAR SCREEN AN
D VARIABLES
110 DIM ATRAP$(6),NAME$(5),FILE$(8)
120 REM SET UP DISK SUFFIX FOR FILE NA
ME.  IOCB IF THE FILE(DEVICE) NUMBER.
 IN=4 IS THE INPUT MODE
130 FILE$="D:":IOCB=2:IN=4
140 REM WRITE ERROR IF TRAP IS SPRUNG.
  IT IS GOOD PRACTICE TO CLOSE FILES T
O PREVENT ERROR #129 IF YOU LOOP BACK
150 REM TO A PREVIOUS PART OF YOUR PRO
GRAM THAT OPENS A FILE.
160 SET=160:CLOSE #IOCB
170 IF ATRAP$="SPRUNG" THEN ? "ERROR 1
70, FILE ";FILE$;" NON-EXISTANT":FOR D
=1 TO 1000:NEXT D:GOTO 100
180 REM KEEPS MESSAGE ON SCREEN TEMPOR
ARILY BEFORE RETURNING TO BEGINNING OF
 PROGRAM
190 TRAP SET:PRINT "TYPE IN FILE NAME"
:PRINT "DO NOT INCLUDE 'D:' PREFIX":IN
PUT NAME$
200 FILE$(3)=NAME$:REM CONCATENATES FI
LE NAME ONTO DEVICE PREFIX 'D:'
210 ATRAP$="SPRUNG"
220 REM IF THE 'OPEN' STATEMENT WORKS,
 WE HAVE A VALID FILE NAME ALREADY STO
RED ON DISK READY FOR INPUT
230 OPEN #IOCB,IN,0,FILE$
240 PRINT "FILE ";FILE$;" OPENED SUCCE
SSFULLY"
250 CLOSE #IOCB
```

Listing 3.

```
100 PRINT "ß":CLR :REM CLEAR SCREEN AN
D VARIABLES
110 REM CATCH END-OF-FILE ERROR
120 DIM ATRAP$(6),A$(124),NAME$(8),FIL
E$(10)
130 FILE$="D:":IOCB=2:IN=4:GNU=8
140 REM 'D:' IS FILE NAME PREFIX.  IN=
4 IS INPUT MODE.  GNU=8 IS OUTPUT MODE
.  IOCB IS DEVICE(FILE) NUMBER
150 REM FIRST WE MUST CREATE A FILE AN
D PUT SOME DATA IN IT BEFORE TRYING TO
 READ THE DATA.
160 PRINT "ENTER A FILE NAME":PRINT "D
O NOT INCLUDE THE 'D:' PREFIX"
170 INPUT NAME$:FILE$(3)=NAME$:REM CON
CATENATES PREFIX AND FILE NAME
180 OPEN #IOCB,GNU,0,FILE$
190 REM WRITE DATA ONTO FILE.
200 PRINT #IOCB;"FIRST"
210 PRINT #IOCB;"SECOND"
220 PRINT #IOCB;"LAST"
230 CLOSE #IOCB:REM IT IS GOOD PRACTIC
E TO KEEP A FILE CLOSED WHEN NOT USED
240 REM FAILURE TO PROPERLY CLOSE A FI
LE CAN CAUSE IT TO BE LOST
250 REM
260 REM READY TO READ THE FILE
270 OPEN #IOCB,IN,0,FILE$
280 SET=310:TRAP SET
290 REM READ DATA FROM FILE AND PRINT
EACH VALUE AS IT IS READ
300 INPUT #IOCB,A$:PRINT A$:GOTO 290
310 PRINT "FINISHED READING FILE SUCCE
SSFULLY":CLOSE #IOCB
320 REM DELETE LINE 280 AND YOU WILL G
ET AN ERROR MESSAGE 136 (END OF FILE)
```

Listing 4.

```
100 PRINT "ß":CLR :REM CLEAR SCREEN AN
D VARIABLES
110 REM CATCH DEVICE TIMEOUT ERROR # 1
38
120 REM YOU FORGOT TO TURN ON AN INPUT
 OR OUTPUT DEVICE
130 DIM ATRAP$(6)
140 SET=140:IF ATRAP$="CAUGHT" THEN PR
INT "TURN ON I/O DEVICE"
150 TRAP SET:ATRAP$="CAUGHT"
160 LPRINT "PROGRAM RAN SUCCESSFULLY"
170 REM RUN THIS PROGRAM WITH PRINTER
TURNED ON AND OFF
180 REM CHANGE LINE 160 TO USE DISK, I
NTERFACE, OR SOME OTHET I/O DEVICE
```

Listing 5.

```
100 PRINT "ß":CLR :REM CLEAR SCREEN AN
D VARIABLES
110 REM READ DATA AND TRAP OUT-OF-DATA
 ERROR #6
120 SET=140:TRAP SET:REM DELETE THIS L
INE AND ERROR #6 WILL OCCUR
130 READ N:PRINT N:GOTO 130
140 PRINT "FINISHED READING DATA"
150 DATA 20,4,156,83,12
```

BASSNOTES IN BASIC

16K Cassette or Disk

by Jerry White

Those of you who have written music using ATARI BASIC may have noticed that even the lowest note available in distortion level 10 is not really a low bass note.

The secret to getting a deep, rich bass note is to use distortion level 12. The BASIC program called **Bassnote** will display the notes and pitch numbers for two octaves of low bass notes.

It will also play the deep bass introduction to the theme from *Barney Miller*. While doing this, the sound commands used will be displayed on your screen. □

```
10 REM BASSNOTE TUTORIAL BY JERRY WHIT
E
20 ?
30 GOSUB 290:GOSUB 190:GOTO 100
40 SOUND 0,0,0,0:READ PITCH:D=12:V=14:
SETCOLOR 2,PITCH,0:SOUND 0,PITCH,D,V
50 POSITION 10,20:? "  SOUND 0,";PITCH
;",";D;",";V;"    ":RETURN
60 FOR HOLD=1 TO 200:NEXT HOLD:SOUND 0
,0,0,0:PITCH=0:D=0:V=0:GOSUB 50:RETURN
70 FOR HOLD=1 TO 50:NEXT HOLD:RETURN
80 FOR HOLD=1 TO 25:NEXT HOLD:SOUND 0,
0,0,0:RETURN
90 DATA 102,90,85,82,75,72,67,67,60,57
,60,67,75,67,51,60,75,90
100 FOR TIME=1 TO 2:GOSUB 40:GOSUB 60:
GOSUB 60
110 GOSUB 40:GOSUB 70:GOSUB 40:GOSUB 7
0
120 GOSUB 40:GOSUB 60:GOSUB 60
130 GOSUB 40:GOSUB 70:GOSUB 40:GOSUB 7
0
140 GOSUB 40:GOSUB 60:GOSUB 60
150 FOR QUARTERNOTE=1 TO 8:GOSUB 40:GO
SUB 70:NEXT QUARTERNOTE
160 GOSUB 40:GOSUB 80:GOSUB 40:GOSUB 8
0
170 GOSUB 40:GOSUB 80:RESTORE :NEXT TI
ME
180 RESTORE :GOSUB 40:GOSUB 60:POKE 75
2,0:END
190 ? :? ," PITCH = NOTE":GOSUB 310
200 ? :? "25=E","27=D#","28=D ","30=C#
"
210 ? "31=C ","33=B ","36=A#","37=A "
220 ? "40=G#","42=G ","45=F#","48=F "
230 ? "51=E ","55=D#","57=D ","60=C#"
240 ? "63=C ","67=B ","72=A#","75=A "
250 ? "82=G#","85=G ","90=F#","97=F "
260 ? "102=E":GOSUB 310
270 ? :? "  THE ATARI BASIC SOUND COMM
AND:"
280 ? :? "SOUND VOICE,PITCH,DISTORTION
,VOLUME":GOSUB 310:RETURN
290 GRAPHICS 0:POKE 752,1:GOSUB 310:?
"      THE THEME FROM BARNEY MILLER"
300 ? :? "BASSNOTES USING SOUND DISTOR
TION 12":GOSUB 310:RETURN
310 FOR CTRLR=2 TO 36:? "-";:NEXT CTRL
R:RETURN
320 REM ******************************
330 REM * D=DISTORTION V=VOLUME     *
340 REM * GOSUB 50 FOR WHOLE NOTE   *
350 REM * GOSUB 70 FOR QUARTER NOTE *
360 REM * GOSUB 80 FOR EIGHTH NOTE  *
370 REM * GOSUB 700 TO DRAW A LINE  *
380 REM ******************************
```

•

CHECKSUM DATA
(See pgs. 7-10)

```
10 DATA 705,653,272,597,653,822,191,72
1,617,79,221,55,227,61,557,6431
160 DATA 242,874,327,262,121,988,40,41
,32,89,691,764,69,86,692,5318
310 DATA 45,788,982,780,31,927,705,806
,5064
```

•

AUDCTL DEMO

16K Cassette or Disk

by Jerry White

AUDCTL is an abbreviation for AUDIO CONTROL, and a label given to decimal location 53768. For those interested in reading up on the functions of the various sound registers, I strongly recommend that you read the SOUND chapter in **De Re ATARI** about three times, and that you try the little demonstration routines supplied.

For those who don't really care to know why things happen, but like to take advantage of the amazing range of sound effects that are available from BASIC, I submit the following little demo program. In a nutshell, POKE commands into decimal locations 53760 through 53767 are used to create a full C major chord. To further enhance the effect of this program, we slide up to the higher C note in line 180.

At the prompt, you may enter a value which will be POKEd into decimal location 53768. Start by entering zero, so you can hear the effect with no distortion before we begin experimenting. By entering other values from 1 to 255, you will notice some strange sounds coming from your TV speaker.

There is probably no better way to learn how to create sound effects than by trial and error. Hopefully, this little demonstration will provide some food for thought. □

```
10 GOSUB 150:REM AUDCTL DEMO BY JERRY
WHITE 6/2/82
20 FOR OFF=0 TO 3:SOUND OFF,0,0,0:NEXT
 OFF:REM TURN OFF ALL SOUNDS
30 ? :? "ENTER A NUMBER BETWEEN 0 AND"
:? "255 THEN PRESS RETURN";
40 POKE 764,255:TRAP 30:INPUT NUMBER
50 NUMBER=INT(NUMBER):IF NUMBER<0 OR N
UMBER>255 THEN 30
60 POKE 53760,243:POKE 53762,81:POKE 5
3764,96:POKE 53766,121:REM C MAJOR
70 FOR X=53761 TO 53767 STEP 2:POKE X,
162:NEXT X
80 REM DISTORTION=10 VOLUME=2 (10*16+2
=162)
90 POKE 53768,NUMBER:REM AUDCTL
100 FOR X=243 TO 60 STEP -1:POKE 53760
,X:NEXT X:REM SLIDE SOUND
110 ? :? "PRESS ESC TO END":? "PRESS A
NOTHER KEY TO CONTINUE":POKE 764,255
120 KEY=PEEK(764):IF KEY=28 THEN POKE
82,2:? :? "BASIC":? "IS";:END
130 IF KEY<>255 OR PEEK(53279)<>7 THEN
 20
140 GOTO 120:REM WHATCHA WANT? PRESSA
KEY!
150 GRAPHICS 0:SETCOLOR 2,9,0:POKE 82,
5:? :REM CLEAR SCREEN/LEFT MARGIN=5
160 ? :? "This program was designed"
170 ? :? "to demonstrate the effects"
180 ? :? "made possible by altering"
190 ? :? "the Audio Control Register"
200 ? :? "at decimal location 53768"
210 ? :? "($d208).":RETURN
```

•

CHECKSUM DATA
(See pgs. 7-10)

```
10 DATA 414,918,846,950,670,674,194,76
3,431,47,269,213,84,763,717,7953
160 DATA 360,641,576,412,839,359,3187
```

•

VARIABLE LISTER

16K Cassette or Disk

by Tony Messina

Have you ever written a program and then tried to go back and document all of the variables that were used? If you're one of the elite 10% who are organized, you probably wrote down all of your variables and their meanings as you wrote the program. If you're like the other 90% of us, who write a program and then spend several agonizing hours documenting it, then help has arrived.

The following utility was written to help me keep track of my variables. It doesn't tell me what I used them for, but it does tell me what I used. This utility is just the start of another utility I'm working on (a cross reference program). You can run out and spend anywhere from $9.00 to $45.00 for any of a multitude of utilities, but I don't have much money — and writing the things myself has taught me more about the inner workings of the ATARI than any listing could. Let me explain how your ATARI stores variable names. It will help you to understand how and why the program works.

Behind the scenes.

Within the heart of your ATARI lurks the **Variable Name Table**. This table contains all of the variables used (and, sometimes, not used) by a program. How do they get there? Good question. When you type in <A=10> for example, the ATARI BASIC cartridge takes the "A" and puts it in the first available slot of the **Variable Name Table**. It also stores the value of our "A" into the **Variable Value Table**. Sounds simple, so far . . . now enters *the curve*. IN ATARI BASIC, variable names can be up to 128 characters long. How does the interpreter know where one variable name ends and the next one begins? What about string variables and dimensioned variables?

Here's the scoop. The very last character of each variable name is stored in the table as an inverse character. Our "A" character would actually be stored in the name table as an inverse A, since the beginning and ending character for the variable is A. If the variable name was "TEST," then "TES" would be stored as normal characters and the last "T" would be stored as an inverse "T." TEST$, a string variable, would be stored as "TEST" (normal) and "$" (inverse). If a variable has dimensions [e.g., DIM A (26)], then the variable is stored as "A" (normal) and "(" (inverse). Knowing where the **Variable Name Table** starts, we should be able to go in and pick out all the variables in any given program.

How do we know where to stop? The end of the table is denoted by a blank byte following the last character of the last variable name. For the purpose of our utility, however, we want to stop picking off variables when we encounter the first variable of the utility program. Armed with this information, let's try our first experiment.

Listing 1.

```
5 REM TYPE A CONTROL COMMA BETWEEN THE
 QUOTES IN LINE 70 TO PRODUCE A HEART
10 ? CHR$(125):REM *CLEAR SCREEN*
20 ? "ASCII","CHAR","ADDRESS":REM * HE
ADINGS *
30 A=10:TEST1=10:DIM B$(1),YES(5,5):RE
M * SAMPLE VARIABLES *
40 START=PEEK(130)+PEEK(131)*256:REM *
GET DECIMAL START ADDRESS OF VAR NAME
TABLE *
50 ? " ";PEEK(START)," ";CHR$(PEEK(STA
RT))," ";START:REM * PRINT ASCII, LETT
ER AND ADDRESS *
60 START=START+1:REM * GET NEXT ONE *
70 IF PEEK(START)=ASC("♥") THEN END :R
EM * IF BLANK THEN END *
80 GOTO 50:REM * GO PRINT NEXT CHARACT
ER *
```

●

CHECKSUM DATA
(See pgs. 7-10)

```
5 DATA 331,198,962,568,625,190,352,707
,530,4463
```

As you can see, the variables for the program itself were printed to your screen. This was just a sample for the non-believers out there. The variables presented are representative of all types used by the ATARI: regular, string and dimensioned. Another thing you will notice is that the variables follow the order in which they were typed. Line 30 is the first place variables were typed in. If we look at the output of our program, we see that the variables follow the same order as Line 30: A,TEST1,B$,YES and START. The address of each letter is also printed in the last column. This will be helpful when we conduct our other experiments, so type in this program.

I hope this little demo illustrated the points I made previously. Here is an explanation of how the utility operates.

The program.

Listing 2 is the utility program. Program flow is as follows:

32500 clears the deck and initializes the utility variables. 32502 clears the screen and outputs a message to the printer. 32504 takes the contents of the current address and stores it in TEMP. A check is then made to see if TEMP is an inverse character (i.e., >=128), or if it is a blank. If one of the conditions is true, the program goes to the subroutine at Line 32514 to find out what the character is. If neither condition is true, we drop through and store the value from TEMP, and store it into the appropriate location in VAR$. We are building our variable name in VAR$ for output to the printer. A check is made of the error flag ERRER. If set, an asterisk is appended to our variable name in VAR$. If clear, then SKIP is checked. If it is set ("set" meaning it is equal to 1), then it's time to print our variable name. If clear ("clear" meaning it's equal to zero), we increment the current address CURADD, the character count CHARCNT and then go back for the next line.

32514-32522 are the subroutine lines used to determine the type of variable. We get here if the value in TEMP was an inverse character or a blank. If the content of TEMP is an ASCII blank, then the program goes to Line 32512, prints out some information and stops. If TEMP contains an inverse "$," then we change it to a normal "$" (TEMP-128) and GOTO 32522. If TEMP contains an inverse "(," then it is changed to a normal "(," and we GOTO 32522. If all of the above fail, then we assume an ASCII number or letter. It is changed to a normal character, and a check is made to see if the new number falls between 48 and 90. If you look in *Appendix C* of the ATARI manual, you see that ASCII 48-90 contains the numbers, some other characters and then the letters A-Z. If the value in TEMP does not fall between any of these values, we have an error, and the error flag is set. If everything is okay, line 32522 increments the number of variables VACNT, sets the skip flag SKIP to 1 and returns.

32524 appends an asterisk to our variable if an error occurred, sets ERRER back to 0 and returns.

32526-32528 check what is in the string VAR$. If the actual name VAR$ is there, then the program ends. If not, then the variable name and its address in RAM is printed. The character count CHARCNT is cleared (set to zero), SKIP is cleared, VAR$ is cleared, and we return to build the next variable name.

32512 prints the start and end address of the name table. It also prints out the number of variables in the target program.

How to use it.

Type the program in exactly as shown in the listing. When you've finished, check everything and then save it using the LIST"D:VARLST" for disk or LIST"C:" for cassette commands. The reason we use LIST rather than CSAVE or SAVE"D:filename" is so that we can merge the utility with your target programs without disturbing anything. Once the program is saved, you can load in any BASIC target program. By target program, I don't mean a program that has target in it; I mean any program you want to obtain a variable listing from, utilizing the utility. Once the target program is loaded, use the following commands to merge the utility. If you have a cassette, cue up the utility and type ENTER"C:" and hit RETURN. After the beep, press the play button, hit RETURN again and the program will load. For disk users, type in ENTER "D:VARLST". The program will then load from disk. Once the utility is loaded, type in (using direct mode) GOTO 32500 and the utility will do its thing.

This utility is set up for output to a printer. If you don't have one, simply replace all LPRINTs with PRINTs. Be prepared to hit CONTROL 1 to stop the screen listing, so you can copy the variable names. Hit CONTROL 1 again to resume output.

You are probably wondering why I have the address printed out. If you don't want it printed, REPLACE Line 32528 with the following:

```
32528 LPRINT VAR$:CHARCNT=0:SKIP=0:VAR
$="":RETURN
```

This will prevent the address from being printed and leave you with a clean piece of paper to document your program. There is a method to my madness in printing the address.

The method.

Consider this. . . if we know the address locations of our variable names, it would follow that, if we POKE different characters into the table, we could change our variable names. This is not only true, but offers other potential benefits and (if the reader is

not careful) problems. *Beware*!! The following experiments should be tried after reading the following paragraph.

The interpreter does not care about variable names, other than when they are initially defined. After that, it doesn't care. *Why*? Well, once you define a variable, it is assigned a number from 128-255. The first variable is assigned 128, the second variable is assigned 129, etc....up to 255. In the tokenized version of your program, these variable *number* assignments become important, not the names. When you list your program, the interpreter scans the tokenized form of your program in memory, and matches all the numbers with KEYWORDS, such as GOTO, REM, COLOR, etc. When he hits a variable number — 128, for example — he says, "Oh...This is a variable; its number is 128, but, to me, that's variable number 1. Let me go into the Variable Name Table and get the name. Since it's number 1, it is the first name in the table." Once the name is retrieved, it is put up on the display. All of this happens in mere microseconds, but that's what your interpreter does. If we happen to change the names in the table, the interpreter will blindly go in and grab whatever is or isn't there. He grabs the variable name based on the *number*, not the *name*. Remember the inverse character at the end of each variable name? Joe Interpreter uses this as a signal to tell him when he has gotten the whole thing. Enough theory, next experiment.

Experiment #2.

Let's try changing some names. If you haven't done so, type in the short example program at the beginning of this article. If you did type it, then load it. RUN the program and follow along with me. On the screen you should see the variable "A" in inverse. Let's change it to "Z". In direct mode, type the following:

```
POKE ADDRESS,ASC("Z")
```

Make sure the "Z" is an inverse "Z." The address will vary with the amount RAM you have and the configuration, so use the address that is on the screen (e.g., the address given for variable "A"). Hit RETURN and, when READY appears, LIST the program. The former statement "A=10" will magically be replaced by "Z=10"!

Let's try once more. Let's change "TEST1" to "BLAH1". First re-RUN the program, then, in direct mode, type the following:

```
POKE ADDRESS,ASC("B"):POKE ADDRESS+1,A
SC("L"):POKE ADDRESS+2,ASC("A"):POKE A
DDRESS+3,ASC("H")
```

("A"):POKE ADDRESS+3,ASC("H")

Again, we have the starting address of "TEST1". Since each letter occupies one byte, then "T" begins at the address listed in your output; "E" is located at the address+1, etc. Since the 1 is already there and in inverse, we don't have to use inverse letters in our POKE statements above. Use the regular old everyday non-inverse letters between the quotes. Hit RETURN and LIST the program. If you did everything right, "TEST1" will be replaced by "BLAH1". Of course, we only replaced variable names with those that had the same length. For experimenting, use the same length name because you can really make a mess out of things. If you are adventurous, try anything!! Just remember that the variable names must end in an inverse character.

Experiment #3.

RUN the program again, then, in direct mode, type the following:

```
FOR Z=FIRSTADDR TO LASTADDR:POKE Z,155
:NEXT Z
```

Substitute the appropriate addresses on the screen for FIRSTADDR and LASTADDR. When READY appears, LIST the program. *Surprise!* All you see now is KEYWORDS. Not a variable in sight! Run the program — yes, just type RUN. *Surprise II*. It works just like normal. Except where the variables once were is now filled with empty space. *Whahappened*??

The 155 POKEd into the name table is a non-printing character. The interpreter picked up the name and even printed it on the screen...we just couldn't see it. You could do this to that *secret* program of yours and let your friend borrow it. When he LISTs it to learn all of your secrets...*boy*, will he get a surprise. *Try it*! I have, and what a ruckus it caused. Be sure to save a copy of the original for yourself, or *you* may be the one who is surprised!

The last experiment.

For our last trick, try this. First load the program, or, if you didn't save it, type it in again (SAVE it this time). Now RUN it. In direct mode, type the following:

```
FOR A=FIRSTADDR TO LASTADDR:POKE A,ASC
("♥"):NEXT A
```

Again use the addresses that are on the screen. When READY appears, LIST the program. Check out all of the *garbage*!! I'll let you figure it out for yourself. (*Hint* — The interpreter searches for inverse characters.)

Final notes.

VARLST will interfere if your target program has the same line numbers as the utility. I started at 32500 as all of my program line numbers fall way below that figure. If necessary, change the line numbers higher or lower, but remember to change all of the GOSUBs and GOTOs. Also, if your target has more than 119 variables in it, VARLST will not load. I've never seen a program with that many variables, but it is possible. If you have any variable names longer than 30 characters, VARLST will not work.* Have fun experimenting! □

*Dimension VAR$ larger in this case.

Listing 2.

```
32500 CLR :DIM VAR$(30):TABLESTART=PEE
K(130)+PEEK(131)*256:CURADD=TABLESTART
:CHARCNT=1:VACNT=0:ERRER=0:INV=128
32502 SKIP=0:? "ß":LPRINT "THE FOLLOWI
NG VARIABLES ARE IN THIS PROGRAM":FOR
X=1 TO 50:NEXT X
32504 TEMP=PEEK(CURADD):IF TEMP>=INV O
R TEMP=ASC("♥") THEN GOSUB 32514
32506 VAR$(CHARCNT,CHARCNT)=CHR$(TEMP)
:IF ERRER THEN GOSUB 32524
32508 IF SKIP THEN GOSUB 32526
32510 CURADD=CURADD+1:CHARCNT=CHARCNT+
1:GOTO 32504
32512 LPRINT :LPRINT "TABLESTART= ";TA
BLESTART:LPRINT "TABLE END = ";CURADD-
4:LPRINT "# OF VARIABLES= ";VACNT-1
32513 END
32514 IF TEMP=ASC("♥") THEN POP :GOTO
32512
32516 IF TEMP=ASC("▒") THEN TEMP=TEMP-
128:GOTO 32522
32518 IF TEMP=ASC("▒") THEN TEMP=TEMP-
128:GOTO 32522
32520 TEMP=TEMP-128:IF TEMP<48 OR TEMP
>90 THEN ERRER=1
32522 VACNT=VACNT+1:SKIP=1:RETURN
32524 VAR$(CHARCNT+1,CHARCNT+1)="▒":ER
RER=0:RETURN
32526 IF VAR$="VAR$" THEN POP :GOTO 32
512
32528 LPRINT VAR$,,,"  ADDRESS= ";CURA
DD-CHARCNT+1:CHARCNT=0:SKIP=0:VAR$="":
RETURN
```

•

CHECKSUM DATA
(See pgs. 7-10)

```
32500 DATA 256,390,205,624,952,663,823
,557,377,141,159,129,984,932,148,7340
32528 DATA 715,715
```

•

Circle Demo

```
10 XC=160:YC=80
20 RD=60:INC=10:YS=0.75
30 GRAPHICS 8:COLOR 1
40 GOSUB 1000:END
1000 REM ---------------------
1010 REM CIRCLE DRAWER ROUTINE
1020 REM ---------------------
1030 REM
1040 REM XC:  x-coordinate of center
1050 REM YC:  y-coordinate of center
1060 REM RD:  circle radius
1070 REM INC: drawing increment 1-360
1080 REM YS:  y-scaling factor
1090 REM
1100 DEG :PLOT XC,YC+RD*YS
1110 FOR CIRCLE=0 TO 360 STEP INC
1120 XCOORD=XC+SIN(CIRCLE)*RD
1130 YCOORD=YC+COS(CIRCLE)*RD*YS
1140 DRAWTO XCOORD,YCOORD
1150 NEXT CIRCLE:RETURN
```

CHECKSUM DATA
(See pgs. 7-10)

```
10 DATA 118,981,32,473,165,240,167,278
,180,184,463,8,40,284,645,4258
1110 DATA 469,958,422,868,442,3159
```

BUNCRUSH

16K Cassette or Disk

by Tony Messina

In our last episode, we left our hero (Bruno Bitmangler) tearing out his hair, looking for his lost energy variable E amidst all the garbage on the TV screen. Meanwhile, Bruno Jr. screams, "I wanna play Missile Command!" and Mrs. Bitmangler shouts, "Both of you get in here... DINNER is getting COLD!" If only our hero had BUNCRUSH, his problem would be solved. What's a BUNCRUSH? It's the BASIC Unembellished No-Cost Cross Reference Utility and Software Helper. If you want to get it up and running, type in **Listing 2** and skip to the "How to use BUNCRUSH" section. Those of you who want to learn a little more about the ATARI BASIC token structure and how BUNCRUSH was developed should read on.

Design considerations.

Several major considerations were involved in designing BUNCRUSH. The list I used was as follows.

1.) Build upon the concepts presented in **Utility #1 — Variable Lister** (see page 20)

2.) Allow use with both Cassette and Disk systems.

3.) Allow screen or printer output.

4.) Output should include the variable name, its associated line reference numbers and be neat in appearance.

5.) Make the output fast and simple.

6.) Provide flexibility for user modifications.

With these considerations in mind, I sat down and wrote BUNCRUSH. It's been rewritten three or four times. Each time it was improved and streamlined. **Listing 2** is the final version.

With all the above ground rules set, I'll dive into the background material, namely ATARI token structure.

BASIC's background.

As was explained in the last utility article, variables are assigned numbers in our token program. Names do not matter, unless we want to print out a program listing. It follows that, if we could locate the start of our token program, scan each line for a variable # (128-255), save the line numbers that contain the variables we are looking for and print out this information, we would be all set. Of course, we would have to do this for every variable number, and it could take some time. We'll worry about the time later. The first question is: where does the tokenized version of our BASIC program begin? Glad you asked! The start location can be found at address 135,137 (Decimal) or $88,89 (Hex). This is not where the program begins, but rather the pointer to where it begins. To obtain the decimal location number, we would execute the following BASIC statement.

```
TOKEN=PEEK(136)+PEEK(137)*256
```

The variable token would be set equal to the start address of our token program. Now what? Well, it's time to scan the program from start to finish for our first variable. Before we do this, I'll digress into my "Here's how a tokenized BASIC line is set up" tap dance routine.

I saw a hand in the back of the room... "What's this 'tokenized program' you keep referring to?" I'm sorry...let me explain. When you type in a program line in BASIC and hit RETURN, several things happen. First, the BASIC cartridge takes each item you typed in and converts it into tokens for its own use. Each command (GOTO, TRAP, etc.), operator (+, -, =, etc.) and function (STR$, SIN, COS, etc.) has a special token associated with it. The interpreter scans, tokenizes, places the token in the program area and continues till it hits your carriage return. If everything is correct with respect to syntax, the cursor appears on the left side of the screen, and you can continue on with the next line. If you make a mistake, the interpreter stops scanning and prints the line out with an error message and an inverse cursor to show you where it stopped.

After you correct your mistake, the interpreter goes through the line again. This process continues until you have entered your entire program.

The tokenizing process is used to save space by converting the ASCII input to tokens. For example, the Restore command would normally take 7 bytes (one per letter). Through tokenization, it only takes 1 byte containing the number 35 Decimal. Tokens serve another important purpose. At Run-Time, the BASIC interpreter fetches a token. This token is actually an index for a jump table. This jump table

points to the various routines within the system. When a token has been executed, BASIC returns, fetches the next token and continues the process of execution.

With that simple explanation out of the way, let's look at the structure of a tokenized line of BASIC. Each line varies depending on its length and the number of multiple statements in it. Some items don't get tokenized. ASCII strings are an example. In a statement such as PRINT "This is a test," the PRINT statement will get tokenized. When the interpreter encounters the quotes, it replaces them with a 15-token (string follows token), saves one space, then puts each letter of the string in one byte until it hits the last quote. The byte after 15 then gets updated to the number of ASCII characters in the string. Similarly, numbers are put in BCD representation. BCD numbers take up 6 bytes for the number itself. For example, with PEEK 130, the PEEK would get a token of 70, and the "(" a token of 58. Then a 14 would be placed next. Fourteen is the "BCD number follows token." After the 14 would be the 6 byte BCD representation of 130 (65 1 48 0 0 0). Don't worry, no need to memorize BCD numbers. Just remember how they appear. Anyway, our example of a simple tokenized BASIC line follows.

BASIC line:
20 PRINT PEEK(Z)

Tokenized form (in decimal):

Bytes	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
	20	0	10	10	32	70	58	128	44	22

Bytes 1 and 2 — Line number LSB MSB FORMAT

Byte 3 — Numerical offset to the next line number in bytes

Byte 4 — Numerical offset to next statement number of bytes. This is used to keep track of where the interpreter is when a line has multiple statements; i.e., 10 GOTO 20:GOSUB 200: PRINT X:GOTO 5 — The remainder of the bytes consists of the tokenized form of our BASIC line.

Byte 5 is the token of PRINT.

Byte 5 is the PEEK token.

Byte 7 is the left parenthesis token ("(").

Byte 8 is the variable number assigned to Z.

Byte 9 is the right parenthesis token (")").

Byte 10 is the end of line token.

To help you get a feel for these concepts, I've included the ATARI BASIC TOKEN TABLE 1. I've also included a short program that prints out the tokenized version of line numbers within a program. This is **Listing 1**. I call it TOKLOOK. Type it in and save it using the LIST command. Now load in one of your BASIC programs. When "ready" appears, load the TOKLOOK program, using the ENTER command. When it's in, type GOTO 32500. Answer the prompt with a line number. The tokenized version of the line will appear, as well as the BASIC form. Use Table 1 and compare the token version with the table. This little utility helped me a great deal in understanding how things get tokenized.

Back to BUNCRUSH.

Well, with that digression out of the way, let's look at **Listing 2**, the actual BUNCRUSH utility. You may notice some similarity to the Variable Lister program. I built BUNCRUSH around it. Variable names were shortened and some unnecessary items removed. There are 2 parts to BUNCRUSH. I used BASIC to handle the string manipulation tasks of finding the variable names and formatting the names/line numbers for output. The ML routine works hand in hand with BASIC. All the ML routing does is search the token program for our variable number (we start at number 128). When it finds it, it returns the line number to BASIC. BASIC then takes the number and puts it in the string VAR$. If VAR$ exceeds the print length of 80, the program prints out that line. BASIC then jumps back into the ML routine, and the search goes on until all variables and line references are output.

Program flow.

Line 32500 — Clears all variables and sets up the program parameters.

Line 32502 — Outputs heading credit. (Go ahead — put your own name in there if you want.)

Line 32503 — Skips some lines, prints out column headings and reads in the ML routine data.

Line 32504 — Gets our variable name, one character at a time. Remember from Variable Lister, an inverse character marks the end of a variable name. If TP>=128 then we subtract 128 and set a flag at 1690 for use later on. I call it the Variable Name Complete Flag. If TP is not >=128 we move on.

Line 32506 — Puts the variable name in VAR$. CC is the Character Count.

Line 32508 — Checks our Variable Name Complete Flag. If it set (=1) we GOSUB 32526. If not, we fall through.

Line 32510 — Updates the current address (CA), the character count (CC) and goes back to 32504 to get the next character of the variable name.

Line 32512 — Skips a few lines and prints out the variable count at the end of the program.

Line 32513 — Ends the program.

Line 32526 — Is a subroutine; we jump here from Line 32508. First we check if our variable name is VAR$. If yes, pop the stack and end the program. If not, we drop through.

Line 32527 — Pads VAR$ with blanks. Variable names can be up to 15 characters long. If you have variable names longer than 15, just

change the 15 to whatever you want. I haven't had any problems yet. Fifteen is a safe number.

Line 32530 — Jumps to our ML routine. The source listing is included as **Listing 3**. The ML routine searches every line of the token program, looking for our variable number. It returns to BASIC under two conditions.

Condition 1: It finds our variable number in a line.

Condition 2: It encounters Line 32500, which is the start of the utility.

Some simplifications were necessary in writing the search program.

1.) If you find our variable, stop searching that line and return to BASIC with the line number. There is no need to search any further, even if the variable appears 10 times in the line. All we care about is the line number, not how many times the variable appears therein.

2.) If we encounter a DATA or REM statement, skip it. There are no variables in DATA or REM statements.

3.) If we pick up a "BCD Number Follows" token (14), skip past it. Searching it is not healthy — we'll get an erroneous cross-reference in some instances.

4.) If we encounter a "String Follows" token (15), skip past the string, as any inverse characters will trigger the "I found our variable" signal. Remember, we look for variable numbers from 128-255.

5.) If we hit a "Statement End" token (15), skip past the next byte. It contains an offset number which can cause errors.

I won't go into too much detail on the ML routine. It's not even very elegant, as a matter of fact. Things can be done to speed it up, but — as you'll see — it's plenty fast enough!!! Anyway, we return to BASIC.

Line 32532 — Checks the con location at 1680 decimal. If set, we are continuing — GO process the line number. If not, we are done with this variable, so drop through.

Line 32534 — Erases the comma at the end of the last line number. If X <=16 then no line numbers were generated for this variable and therefore there are no references for it.

Line 32535 — Prints out VAR$, zeros out the character count, clears out VAR$ and NUM$ and returns to 32510 to get the next variable flag.

Line 32536 — Gets the current line number (CL) from locations 1683 and 1684 — that's where the ML routine put them.

Line 32538 — Converts the line number to a string. It checks to see if the length of this line number, when added to the current length of VAR$, will be greater than 80. If it would, VAR$ gets printed first, then is padded with 15 blanks.

Line 32540 — The line number get added to VAR$ and a (comma space) is appended. Here, X is updated to reflect the length of VAR$. We then jump back to the ML routine so we can continue on.

How to use BUNCRUSH.

Type in the program from **Listing 2**. Double-check everything, especially the ML DATA, to ensure a good program. Save the program to disk using the LIST "D:BUNCRUSH" command or to cassette using the LIST "C:" command. To use BUNCRUSH:

1.) Load in the program you want to cross reference.

2.) Load in BUNCRUSH using the ENTER "D:BUNCRUSH" command for disk or the ENTER "C:" command for cassette.

3.) When READY appears, be sure your printer and interface are turned on.

4.) Type in immediate mode GOTO 32500.

5.) BUNCRUSH should now print out the title and the column header VAR LINE NUMBERS to the printer.

6.) The CRT display should say READING ML PROGRAM. After 3-5 seconds GOOOO!! should appear, and the printer should be busy dumping out the Variable Cross Reference.

Modifications.

The program in **Listing 2** is set up for an ATARI 825 printer with a line output of 80 columns. Modifications for other printers follow:

1.) PRINTER — If you have an ATARI 40-column printer, change the >80 in Line 32538 to >40.

2.) NO PRINTER — If you don't have a printer, change all LPRINT statements to PRINT in Lines 32502, 32503, 32512, 32535 and 32538. In addition, change the 80 in Line 32538 to 39. Everything will now be dumped out to the screen. Use the CNTRL 1 key to STOP/START the listing.

3.) LINE NUMBERS — If you want to change the line numbers for BUNCRUSH in order to move it up or down, you must *beware* or certain items. All GOTO and GOSUB references must be changed to reflect the new line numbers. The most *important* change of all is in the ML routine itself. The ML routine checks to see if the current line number is 32500. If you change the starting line number of BUNCRUSH, you must change the check in the ML routine.

DATA Line 32548, item 14 is a 126 which is the MSB of the line number 32500; DATA Line 32500, item 5 is a 244 which is the LSB of 32500. Anyway, whatever your new line number, break it down into LSB/MSB format

and substitute the appropriate numbers in the above mentioned locations.

4.) OTHER CHANGES — Other things which you may want to add to BUNCRUSH are ERROR CHECKING and an INPUT line which will let you title the listing in expanded print so you know what program is being Cross Referenced. Another change which would require some work is to output an alphabetical Cross Reference. The possibilities for additions are limited only by your imagination.

Drawbacks and limitations.

BUNCRUSH has some limitations which I thought should be mentioned prior to receiving a bunch of nasty phone calls and letters. Limitations on BUNCRUSH are identical to those of the **Variable Lister Utility** on page 20. BUNCRUSH will not work correctly if:

1.) The target program uses more than 120 variables. BUNCRUSH will abort the load procedure with an ERROR 4 (Too Many Variables).

2.) Line numbers are the same as BUNCRUSH. In this case, BUNCRUSH will merge just fine with the target program but may cause problems if the target program has line numbers not contained in BUNCRUSH.

3.) The target program is so large that BUNCRUSH will not load due to an ERROR 2 (Insufficient Memory).

I've never had problems with item 2 or 3. I have a 48K system, however, and this may be the reason. I have encountered item 1 only once, and it was with a canned program. There is a way around all of these problems — a method by which BUNCRUSH will work on ANY BASIC program. If BUNCRUSH were written entirely in machine language, without BASIC overhead, everything would work fine. I'll leave that as an exercise for the reader. □

Listing 1.

```
32500 CLR :DIM VAR$(1):ST=PEEK(136)+PE
EK(137)*256:NT=ST
32502 ? CHR$(125):? "INPUT LINE # TO E
XAMINE":INPUT A
32504 TL=PEEK(NT)+PEEK(NT+1)*256:BC=PE
EK(NT+2):IF TL=32500 OR TL>A THEN ? "L
INE NOT FOUND!!":GOTO 32512
32506 IF TL<>A THEN NT=NT+BC:GOTO 3250
4
32507 ? " LINE#","NXT LINE","NXT STMNT
"
32508 ? "LSB/MSB"," OFFSET"," OFFSET"
32509 ? " ";PEEK(NT);"/";PEEK(NT+1),"
  ";PEEK(NT+2),"   ";PEEK(NT+3)
32510 ? "TOKENIZED STATEMENT":FOR X=NT
+4 TO NT+BC-1:? PEEK(X);" ";:NEXT X:?
:? "BASIC STATEMENT":LIST A
32512 ? :? "ANOTHER LINE? Y/N ":INPUT
VAR$
32514 IF VAR$(1,1)="Y" THEN NT=ST:GOTO
 32502
32516 END
```

CHECKSUM DATA
(See pgs. 7-10)

```
32500 DATA 945,677,832,112,92,760,733,
158,821,323,566,6019
```

Listing 2.

```
32400 REM ***************************
32410 REM * THIS REM IS TO LET YOU *
32415 REM * KNOW THAT THIS VERSION *
32420 REM * OF BUNCRUSH HAS BEEN   *
32430 REM * IMPROVED TO HANDLE ALL *
32440 REM * CASES OF IF/THEN. DON'T*
32450 REM * TYPE IN THIS REM, JUST *
32460 REM * READ FOR INFORMATION.. *
32470 REM ***************************
32480 REM *
32500 CLR :DIM VAR$(80),NUM$(5):CA=PEE
K(130)+PEEK(131)*256:CC=1:POKE 1699,0
32502 ? "K":LPRINT "CROSS REFERENCE UT
ILITY VER. 2.6 BY TONY MESSINA NEWPORT
, RI"
32503 LPRINT :LPRINT :LPRINT "VAR
      LINE NUMBERS":LPRINT :GOSUB 325
42
32504 TP=PEEK(CA):IF TP>=128 THEN TP=T
P-128:POKE 1699,1
32506 VAR$(CC,CC)=CHR$(TP)
32508 IF PEEK(1699) THEN GOSUB 32526
32510 CA=CA+1:CC=CC+1:GOTO 32504
32512 LPRINT :LPRINT "# OF VARIABLES=
";PEEK(1695)-128
32513 END
32526 IF VAR$="VAR$" THEN POP :GOTO 32
512
32527 FOR X=CC+1 TO 15:VAR$(X,X)=" ":N
EXT X:GOTO 32530
32530 A=USR(1536)
32532 IF PEEK(1694) THEN GOTO 32536
32534 VAR$(X-1,X-1)=" ":IF X<=16 THEN
VAR$(LEN(VAR$)+1)="NO REFERENCES"
32535 LPRINT VAR$:LPRINT :CC=0:POKE 16
99,0:VAR$="":NUM$="":RETURN
32536 CL=PEEK(1697)+PEEK(1698)*256
32538 NUM$=STR$(CL):IF LEN(VAR$)+LEN(N
UM$)+2>80 THEN LPRINT VAR$:VAR$="
      "
32540 VAR$(LEN(VAR$)+1)=NUM$:VAR$(LEN(
VAR$)+1)=", ":X=LEN(VAR$):GOTO 32530
32542 RESTORE 32546:? "READING ML PROG
RAM":FOR X=1536 TO 1699:READ TP:POKE X
,TP:NEXT X
32544 ? CHR$(125):? "GOOOOOL":RETURN
32546 DATA 169,0,205,158,6,208,8,165,1
36,133
32548 DATA 205,165,137,133,206,160,0,1
77,205,141
32550 DATA 161,6,200,177,205,141,162,6
,201,126
32552 DATA 208,7,173,161,6,201,244,240
,96,200
32554 DATA 177,205,141,157,6,160,4,177
,205,201
32556 DATA 20,208,9,192,4,240,1,200,20
0,76
32558 DATA 115,6,205,159,6,240,59,201,
0,240
32560 DATA 49,201,1,240,45,201,14,208,
8,152
32562 DATA 24,105,7,168,76,115,6,200,2
01,15
32564 DATA 208,23,136,136,177,205,200,
200,201,27
32566 DATA 240,13,177,205,140,160,6,23
8,160,6
32568 DATA 24,109,160,6,168,204,157,6,
144,183
```

```
32570 DATA 32,144,6,76,15,6,141,158,6,
32
32572 DATA 144,6,76,142,6,136,238,159,
6,140
32574 DATA 158,6,104,96,165,205,24,109
,157,6
32576 DATA 133,205,144,2,230,206,96,0,
0,128
32578 DATA 0,0,0,0,0,0
```

●

CHECKSUM DATA
(See pgs. 7-10)

```
32400 DATA 582,785,847,663,792,800,659
,796,596,385,897,819,873,152,100,9746
32500 DATA 501,933,946,557,148,806,127
,271,666,612,288,126,988,675,560,8204
32546 DATA 698,145,91,876,121,536,822,
763,817,137,860,924,484,815,716,8805
32576 DATA 795,693,1488
```

●

Assembly language listing.

```
0005 ;*********************************
0010 ;* ML SEARCH AIDE FOR ATARI 400*
0015 ;* /800 BY TONY MESSINA.       *
0020 ;* 48 DUDLEY AVE NEWPORT, RI   *
0025 ;* 02840 VERSION 2.6 10 JUL 83 *
0030 ;*********************************
0035 ;*********************************
0040 ;* EQUATES FOR PROGRAM FOLLOW   *
0045 ;*********************************
0050 ;
0055 DATA       .DI 1               ; DATA TOKEN
0060 REMARK     .DI 0               ; REM TOKEN
0065 BCD        .DI 14              ; BCD # TOKEN
0070 STRING     .DI 15              ; STRING TOKEN
0075 STMT       .DI 20              ; STATEMENT END
0080 THEN       .DI 27              ; THEN TOKEN
0085 TOKPTR     .DI $0088           ; POINTER TO BAS
0090 PG0        .DI $00CD           ; LOC ON PAGE 0
0095 ;
0100 ;*********************************
0105 ;* THIS PROGRAM DOES A SEARCH   *
0110 ;* TO AIDE BUNCRUSH. BASIC WAS *
0115 ;* TOO SLOW, SO THIS ML ROUTINE*
0120 ;* WAS WRITTEN TO SPEED THINGS *
0125 ;* UP A BIT..                  *
0130 ;*********************************
0135 ;
0140            .OS                 ; STORE OBJECT IN MEM
0145            .BA $0600           ; ORIGIN PG6
0150 BEGIN      LDA #0              ; LOAD A WITH 0
0155            CMP CON             ; CK WITH CON FLAG
0160            BNE CONTIN          ; SKIP INIT IF NOT 0
0165 INIT       LDA *TOKPTR         ; GET LSB OF POINTER
0170            STA *PG0            ; STORE IT
0175            LDA *TOKPTR+1       ; GET MSB OF POINTER
0180            STA *PG0+1          ; STORE IT ALSO
0185 CONTIN     LDY #0              ; START Y AT ZERO
0190            LDA (PG0),Y         ; GET LSB OF LINE NUMBER
0195            STA LINNUM          ; SAVE IT FOR BASIC
0200            INY                 ; INCREMENT OFFSET BY 1
0205            LDA (PG0),Y         ; GET MSB OF LINE NUMBER
0210            STA LINNUM+1        ; SAVE IT FOR BASIC
0215 ;
0220 ;**** CHECK THIS LINNUM FOR 32500 ****
0225 ;
0230            CMP #$7E            ; IS IT = TO MSB
0235            BNE NOEQ            ; IF NO THEN START
0240            LDA LINNUM          ; YES SO CK LSB
0245            CMP #$F4            ; DO IT
0250            BEQ DONE            ; IF EQ. DONE THIS VAR
0255 NOEQ       INY                 ; INC PTR TO NEXT LOCATION
0260            LDA (PG0),Y         ; GET BASIC LINE BYTE CNT
0265            STA COUNT           ; SAVE IT FOR FUTURE CKS
0270            LDY #4              ; GET NEW OFFSET
0275 START      LDA (PG0),Y         ; GET A BYTE INDIRECTLY
0280            CMP #STMT           ; CK FOR A STMNT/DIM TOKEN
0285            BNE TARGCK          ; IF NO, CK FOR TGT TOKEN
0290            CPY #4              ; WAS IT 1ST BYTE?
0295            BEQ WASDIM          ; YES..IT WAS A DIM!
0300            INY                 ; INC 2 IF STMNT
0305 WASDIM     INY                 ; INC 1 FOR DIM
0310            JMP CKCNT           ; SEE IF WE ARE DONE
0315 TARGCK     CMP TARGET          ; IS IT OUR TARGET
0320            BEQ PROCIT          ; IF= GO PROCESS THIS LINE
0325            CMP #REMARK         ; NO CK REM
0330            BEQ SKIPIT          ; IF REM SKIP THIS LINE
0335            CMP #DATA           ; NOT REM. CK DATA
0340            BEQ SKIPIT          ; IF DATA SKIP IT ALSO
0345            CMP #BCD            ; NOT DATA CK BCD NUMBER
0350            BNE STRCK           ; IF NOT BCD CK FOR STRING
0355            TYA                 ; ITS BCD PUT OFFSET IN A
0360            CLC                 ; CLEAR CARRY FOR ADD
0365            ADC #7              ; ADD 7 TO SKIP THE BCD #
0370            TAY                 ; PUT NEW OFFSET BACK IN Y
0375            JMP CKCNT           ; AND GO CK COUNT
0380 STRCK      INY                 ; INC PTR BY ONE
0385            CMP #STRING         ; CK IF STRING TOKEN
0390            BNE CKCNT           ; IF NO, GO CK THE COUNT
0395            DEY                 ; ELSE DEC FOR
0400            DEY                 ; THEN CHECK
0405            LDA (PG0),Y         ; GET PREVIOUS TOKEN
0410            INY                 ; THEN RESTORE
0415            INY                 ; ORIGINAL POINTER
0420            CMP #THEN           ; IS IT THEN?
0425            BEQ CKCNT           ; YES. IF/THEN NOT STRING!
0430            LDA (PG0),Y         ; NO..GET STRING CNT
0435            STY YSAVE           ; SAVE Y
0440            INC YSAVE           ; INC PAST THE LAST STRING
0445            CLC                 ; CLEAR CARRY FOR ADD
0450            ADC YSAVE           ; ADD STRING COUNT TO OLD
0455            TAY                 ; PUT CNT BACK IN Y REG
0460 CKCNT      CPY COUNT           ; ARE WE IN NXT BASIC LINE
0465            BCC START           ; IF NO GET THIS BYTE
0470 SKIPIT     JSR TOKUP           ; IF YES UPDATE TOKEN PTR
0475            JMP CONTIN          ; CONTINUE TO LOOK
0480 PROCIT     STA CON             ; MAKE CON NON-ZERO
0485            JSR TOKUP           ; UPDATE PAGE 0 POINTER
0490            JMP BASIC           ; EXIT TO BASIC
0495 DONE       DEY                 ; DEC Y TO ZERO
0500            INC TARGET          ; UPDATE TARGET NUMBER
0505            STY CON             ; ZERO OUT CON FOR BASIC
0510 BASIC      PLA                 ; PULL NASTYNESS OFF
0515            RTS                 ; RETURN TO BASIC
0520 ;
0525 ;*******************************
0530 ;*    * SUBROUTINE TOKUP *     *
0535 ;*    ********************     *
0540 ;* THIS SUBROUTINE UPDATES THE *
0545 ;* PG0 PTR OF THE TOKEN PROGRAM*
0550 ;* THE OLD PTR IS LOADED AND   *
0555 ;* THEN THE BYTE CNT IS ADDED. *
0560 ;* IF CARRY IS SET PG0+1 IS    *
0565 ;* ALSO UPDATED                *
0570 ;*******************************
0575 ;
0580 TOKUP      LDA *PG0            ; GET LSB OF POINTER
0585            CLC                 ; CLEAR CARRY FOR ADD
0590            ADC COUNT           ; ADD CNT TO NXT LINE #
0595            STA *PG0            ; PUT IT BACK
0600            BCC OUT             ; IF CARRY CLEAR GET OUT
0605            INC *PG0+1          ; OOPS, CARRY SET. INC MSB
0610 OUT        RTS                 ; SCRAM SAM!!
0615 ;
0620 ;******** LOCAL VARIABLES FOLLOW ********
0625 ;
0630 COUNT      .BY 0               ; BYTE CNT THIS BASIC LINE
0635 CON        .BY 0               ; FLAG FOR BASIC & ML ROUT
0640 TARGET     .BY 128             ; VARIABLE TOKEN # START A
0645 YSAVE      .BY 0               ; Y REGISTER SAVE AREA
0650 LINNUM     .BY 0               ; BASIC LINE NUMBER LSB
0655            .BY 0               ; MSB OF LINE #
0660 INVFLG     .BY 0               ; INVERSE FLAG AREA
0665            .EN
```

ATARI BASIC TOKEN TABLE

COMMANDS		OPERATORS		FUNCTIONS	
HEX	DEC	HEX	DEC	HEX	DEC
00	0 REM	0E	14 [NUM CONST]	3D	61 STR$
01	1 DATA	0F	15 [STR CONST]	3E	62 CHR$
02	2 INPUT	10	16 "	3F	63 USR
03	3 COLOR	11	17 [NOT USED]	40	64 ASC
04	4 LIST	12	18 ,	41	65 VAL
05	5 ENTER	13	19 $	42	66 LEN
06	6 LET	14	20 : [STMT END]	43	67 ADR
07	7 IF	15	21 ;	44	68 ATN
08	8 FOR	16	22 [LINE END]	45	69 COS
09	9 NEXT	17	23 GOTO	46	70 PEEK
0A	10 GOTO	18	24 GOSUB	47	71 SIN
0B	11 GO TO	19	25 TO	48	72 RND
0C	12 GOSUB	1A	26 STEP	49	73 FRE
0D	13 TRAP	1B	27 THEN	4A	74 EXP
0E	14 BYE	1C	28 #	4B	75 LOG
0F	15 CONT	1D	29 <= [NUMERICS]	4C	76 CLOG
10	16 COM	1E	30 <>	4D	77 SQR
11	17 CLOSE	1F	31 >=	4E	78 SGN
12	18 CLR	20	32 <	4F	79 ABS
13	19 DEG	21	33 >	50	80 INT
14	20 DIM	22	34 =	51	81 PADDLE
15	21 END	23	35 ↑	52	82 STICK
16	22 NEW	24	36 *	53	83 PTRIG
17	23 OPEN	25	37 +	54	84 STRIG
18	24 LOAD	26	38 -		
19	25 SAVE	27	39 /		
1A	26 STATUS	28	40 NOT		
1B	27 NOTE	29	41 OR		
1C	28 POINT	2A	42 AND		
1D	29 XIO	2B	43 (		
1E	30 ON	2C	44)		
1F	31 POKE	2D	45 = [ARITHM ASSIGN]		
20	32 PRINT	2E	46 = [STRING ASSIGN]		
21	33 RAD	2F	47 <= [STRINGS]		
22	34 READ	30	48 <>		
23	35 RESTORE	31	49 >=		

24	36	RETURN	32	50	<
25	37	RUN	33	51	>
26	38	STOP	34	52	=
27	39	POP	35	53	+ [UNARY]
28	40	?	36	54	-
29	41	GET	37	55	([STRING LEFT PAREN]
2A	42	PUT	38	56	([ARRAY LEFT PAREN]
2B	43	GRAPHICS	39	57	([DIM ARRAY LEFT PAREN]
2C	44	PLOT	3A	58	([FUN LEFT PAREN]
2D	45	POSITION	3B	59	([DIM STR LEFT PAREN]
2E	46	DOS	3C	60	, [ARRAY COMMA]
2F	47	DRAWTO			
30	48	SETCOLOR			
31	49	LOCATE			
32	50	SOUND			
33	51	LPRINT			
34	52	CSAVE			
35	53	CLOAD			
36	54	[IMPLIED LET]			
37	55	ERROR- [SYNTAX]			

Triangle Demo

```
5 C=1
10 GRAPHICS 23
15 E=INT(300*RND(1))
20 D=INT(300*RND(1))
25 C=1
30 COLOR C
35 B=39
40 A=79
45 FOR S=1 TO D STEP E
50 FOR X=A TO B STEP -2
55 PLOT 80,A-X
60 DRAWTO 80+X,INT(A/5)
65 DRAWTO 80,X
70 DRAWTO 80-X,INT(A/5)
75 DRAWTO 80,A-X
80 IF PEEK(764)<>255 THEN END
85 COLOR C
90 NEXT X
95 C=C+1
100 NEXT S
105 SETCOLOR 0,T,2
110 T=T+1
115 GOTO 5
```

CHECKSUM DATA

(See pgs. 7-10)

```
5 DATA 693,999,498,483,967,760,238,227
,138,962,886,140,38,146,502,7677
80 DATA 871,785,403,77,748,508,326,662
,4380
```

SYS/STAT

16K Cassette or Disk

by Robert Hartman

System Status is a BASIC program that allows the user to look at a formatted listing of all the devices accessible to him/her. It also has the capability to display 64 files on drives one through four. Its main purpose, however, is not to be a menu, but to supply the user with information regarding the accessibility of the four RS-232 ports.

NOTE: If a drive is started up after the program has been run, it is necessary to re-run the program in order to get a menu on that particular drive. □

```
10 REM Analog System Status
20 REM Version 1.1
30 REM Copyright (C) April, 1981
40 REM by Robert W. Hartman
50 DIM A$(20),B$(6),F$(5),A(5):GRAPHIC
S 0:POKE 752,1:POKE 559,0:POKE 82,1:PO
KE 83,39:FR=FRE(0):LSCH=764:CON=53279
60 POKE 65,0:REM Noisy I/O off
70 REM SET UP SCREEN
80 FOR I=19 TO 22:POSITION 18,I:? "|":
NEXT I
90 POSITION 12,1:? "analog  systat":FO
R I=12 TO 25:POSITION I,0:? "-":POSITI
ON I,2:? "-":POKE CON,0:NEXT I
100 FOR I=0 TO 38:POSITION I,3:? "-":P
OSITION I,19:? "-":NEXT I
110 REM CHEAT (just a little)
120 POSITION 12,5:? "Devices Present":
FOR I=12 TO 26:POSITION I,6:? "-":NEXT
 I:POSITION 1,7:? "K -Keyboard"
130 POSITION 1,9:? "S -Screen":POSITIO
N 1,11:? "E -Editor":POSITION 1,13:? "
C -Cassette":C=7:R=26
140 REM SYSTAT
150 TRAP 190:OPEN #1,6,0,"D1:*.*":POSI
TION R,C:? "D1 -Drive #1":GOSUB 260:D1
=1
160 OPEN #2,6,0,"D2:*.*":POSITION R,C:
? "D2 -Drive #2":GOSUB 260:D2=1
170 OPEN #3,6,0,"D3:*.*":POSITION R,C:
? "D3 -Drive #3":GOSUB 260:D3=1
180 OPEN #4,6,0,"D4:*.*":POSITION R,C:
? "D4 -Drive #4":GOSUB 260:D4=1
190 TRAP 200:OPEN #5,8,0,"R:":GOSUB 27
0
200 CLOSE #5:TRAP 210:OPEN #5,8,0,"P:"
:POSITION 14,14:? "P -Printer"
210 REM MEMORY
220 POSITION 1,19:? "Amount-of-Memory"
:POSITION 1,20:? FR:FOR I=1 TO 5:POSIT
ION I,20:GET #6,A:A(I)=A:NEXT I
230 FOR I=1 TO 5:F$(I,I)=CHR$(A(I)+128
):NEXT I:POSITION 1,20:? ,:POSITION 2,
21:? F$;" bytes"
240 GOTO 280
250 FOR I=1 TO 7:CLOSE #I:NEXT I:RETUR
N
260 C=C+2:RETURN
270 POSITION 12,16:FOR I=1 TO 4:? "R";
CHR$(I+48+128);", ";:NEXT I:? "4":POS
ITION 12,17:? "RS232-C ports":RETURN
280 TRAP 32767:POSITION 23,19:? "Comma
nds":POSITION 24,20:? "K-Menu(s)"
290 POSITION 24,21:? "R-Run again    "
:POSITION 24,22:? "E-EXIT":POKE 559,34
:SETCOLOR 2,4,4:POKE LSCH,255
300 CLOSE #5:OPEN #5,4,0,"K:":GET #5,A
:IF A<>69 AND A<>77 AND A<>82 THEN 300
310 IF A=69 THEN GRAPHICS 0:POKE 65,1:
GOSUB 250:NEW
320 IF A=82 THEN RUN
330 REM MENU(S)
340 POSITION 23,19:? "----------":POKE 2
01,14:FOR I=20 TO 22:POSITION 24,I:? ,
:NEXT I:POSITION 24,21:? "Enter Drive"
350 TRAP 280:POSITION 37,21:INPUT DR
360 IF DR<1 OR DR>4 THEN 280
370 IF DR=1 AND D1=1 THEN DRV=1:GOTO 4
20
380 IF DR=2 AND D2=1 THEN DRV=2:GOTO 4
20
390 IF DR=3 AND D3=1 THEN DRV=3:GOTO 4
20
400 IF DR=4 AND D4=1 THEN DRV=4:GOTO 4
20
410 GOTO 280
420 ? "K":POSITION 2,1:? "Menu for Dri
ve #";DRV:? :? :GOSUB 250:B$="D :*.*":
B$(2,2)=STR$(DRV)
430 OPEN #1,6,0,B$:OPEN #2,4,0,"K:"
440 TRAP 480:INPUT #1,A$:N=N+1
450 ? A$(2,LEN(A$)):IF PEEK(90)=21 THE
N POKE 82,PEEK(82)+20:POSITION PEEK(82
),4
460 IF N=35 THEN GOTO 520
470 GOTO 440
480 ? CHR$(28);"                    ":? :I
F LEN(A$)>15 THEN IF A$(10,11)="SE" TH
EN GOTO 500
490 A$(LEN(A$)+1)=" FREE SECTORS"
500 FOR I=1 TO LEN(A$):A$(I,I)=CHR$(AS
C(A$(I,I))+128):NEXT I:? A$
510 POKE LSCH,255:GET #2,A:CLOSE #2:RU
N
520 REM Get rest of Menu After Char
530 POKE LSCH,255:GET #2,A:POKE 82,2:?
 "K":POSITION 2,3
540 TRAP 570:INPUT #1,A$:? A$(2,LEN(A$
))
550 IF PEEK(90)=22 THEN POKE 82,PEEK(8
2)+20:POSITION PEEK(82),4
560 GOTO 540
570 TRAP 32767:GOTO 480
```

•

CHECKSUM DATA

(See pgs. 7-10)

```
10 DATA 988,4,923,497,183,455,780,739,
522,805,473,0,266,62,125,6822
160 DATA 971,986,1,844,985,35,329,933,
723,526,292,597,140,128,961,8451
310 DATA 771,800,146,204,921,738,288,2
97,306,287,718,918,984,932,640,8950
460 DATA 418,728,64,134,615,333,767,23
7,39,830,729,937,5831
```

FASTER CHARACTER DUMPS

16K Cassette or Disk

by Joseph T. Trem

If you are an avid ATARI enthusiast as I am, then you probably have gone through quite a number of different programs. Many of the better programs use character redefinition. Unfortunately, to define a new character set one must move ATARI's character set into a new defined memory location. In BASIC, this takes time. For 1024 bytes (128 characters or 4 pages) it takes approximately eleven seconds. It's downright boring!

This article demonstrates a machine language routine in string form which transfers ATARI's character set into a user-chosen RAM area. It runs in a split second. Before going any further, I must state that this article is not a tutorial on character redefinition or animation, although they are both used in the demo.

Here is a brief summary of the four sample programs included. **Program 1** demonstrates character redefinition and the time involved in transferring 1024 bytes (line 40). This takes approximately 11 seconds. **Program 2** incorporates the machine language routine and takes less than a second (line 30). **Program 3** demonstrates a sample program with sound and animation. There are five redefined characters. After the program executes once, it recycles and re-executes all over again. Notice the time it takes to rerun...remember: every time this program runs, it is dumping 1024 bytes in under a second! **Program 4** is the source code for the machine language routine.

The technique used in this program is called a block move. We simply look at what is in ATARI's character base address and move that data to our new character address, one byte at a time. This technique is also good for player/missile graphics. You can zero out all player/missile data in a split second. Just think, no more time delay to clear P/M memory. Sound great? Then read on...here is the documentation for the first three programs.

Program 1.

Line 20 — Sets up character variables
Line 40 — Transfers characters (slow)
Lines 50-70 — Reads in new character
Line 80 — Points to new character base

Program 2.

Line 20 — Sets up character variables
Line 30 — Transfers characters (fast)
Lines 40-60 — Reads in new character
Line 70 — Points to new character base

Program 3.

Line 10 — Sets up variables
Line 100 — Transfers 128 characters
Lines 1000-2170 — Alters character set
Line 3000 — Points to a new character base
Lines 3500-7000 — Main loop for animation
Line 10000 — Sound routine

Here's some information on our USR call:

```
A=USR(ADR(E$),ADDR,PAGE)
```

ADDR=address where new character set is to reside

PAGE=the number of 256 blocks you wish to move.

In closing, I hope that everyone will enjoy the substantial increase in speed this subroutine can provide. Just think, no more "Please wait..." prompts. □

Program 1

```
10 REM ***DUMPS 1024 BYTES TO NEW CHBA
S USING ONLY BASIC (APROX. 11 SECONDS)
***
20 DIM E$(50):RAMTOP=PEEK(106)-8:POKE
106,RAMTOP:CHBAS=RAMTOP:ADDR=CHBAS*256
30 GRAPHICS 17:POSITION 8,9:? #6;"MOVI
NG CHARACTER SET"
40 FOR X=0 TO 1023:POKE ADDR+X,PEEK(57
344+X):NEXT X
50 CHAR=59:POS=ADDR+(CHAR*8)
60 DATA 0,24,36,66,153,66,60,0
70 FOR X=0 TO 7:READ A:POKE (POS+X),A:
NEXT X
80 GRAPHICS 18:POKE 752,1:POKE 756,CHB
AS
90 POSITION 10,5:? #6;"["
100 GOTO 100
```

CHECKSUM DATA
(See pgs. 7-10)

```
10 DATA 377,95,598,506,976,318,760,421
,361,683,5095
```

•

Program 2

```
10 REM ***DUMP5 1024 BYTES TO NEW CHBA
5 IN MACHINE LANGUAGE (NO DELAY)***
20 DIM E$(50):RAMTOP=PEEK(106)-8:POKE
106,RAMTOP:CHBA5=RAMTOP:ADDR=CHBA5*256
:PAGE=4
30 FOR X=1 TO 40:READ N:E$(X)=CHR$(N):
NEXT X:A=USR(ADR(E$),ADDR,PAGE):REM *D
UMP ROUTINE*
40 DATA 104,104,133,207,104,133,206,10
4,104,133,212,169,0,133,204,169,224,13
3,205,162
50 DATA 1,160,0,177,204,145,206,200,20
8,249,230,205,230,207,232,228,212,208,
240,96
60 CHAR=59:POS=ADDR+(CHAR*8)
70 DATA 0,24,36,66,153,66,60,0
80 FOR X=0 TO 7:READ A:POKE (POS+X),A:
NEXT X
90 GRAPHICS 18:POKE 752,1:POKE 756,CHB
A5
100 POSITION 9,5:? #6;"["
110 GOTO 110
```

•

CHECKSUM DATA
(See pgs. 7-10)

```
10 DATA 659,729,470,848,552,978,320,76
2,423,292,689,6722
```

•

Program 3

```
10 CLR :DIM E$(50):RAMTOP=PEEK(106):CH
BA5=RAMTOP-8:ADDR=CHBA5*256:PAGE=4:SND
=10000
100 FOR X=1 TO 40:READ N:E$(X)=CHR$(N)
:NEXT X:A=USR(ADR(E$),ADDR,PAGE):REM *
DUMP ROUTINE*
110 DATA 104,104,133,207,104,133,206,1
04,104,133,212,169,0,133,204,169,224,1
33,205,162
120 DATA 1,160,0,177,204,145,206,200,2
08,249,230,205,230,207,232,228,212,208
,240,96
1000 CHAR=59:POS=ADDR+(CHAR*8)
1010 DATA 0,0,144,96,144,0,0,0
1020 FOR X=0 TO 7:READ A:POKE (POS+X),
A:NEXT X
2000 CHAR=60:POS=ADDR+(CHAR*8)
2010 DATA 0,6,6,15,6,6,0,0
2020 FOR X=0 TO 7:READ A:POKE (POS+X),
A:NEXT X
2050 CHAR=61:POS=ADDR+(CHAR*8)
2060 DATA 0,0,0,20,8,20,0,0
2070 FOR X=0 TO 7:READ A:POKE (POS+X),
A:NEXT X
2100 CHAR=62:POS=ADDR+(CHAR*8)
2110 DATA 0,0,20,10,60,20,10,0
2120 FOR X=0 TO 7:READ A:POKE (POS+X),
A:NEXT X
2150 CHAR=63:POS=ADDR+(CHAR*8)
2160 DATA 0,148,104,57,86,72,2,0
2170 FOR X=0 TO 7:READ A:POKE (POS+X),
A:NEXT X
3000 GRAPHICS 17:POKE 752,1:POKE 756,C
HBA5
3010 POSITION 1,20:? #6;"ASSY CHARACTE
R DUMP"
3500 FOR X=0 TO 4:POKE 708,14:POSITION
 X,5:? #6;" [ ":GOSUB SND:POKE 708,8:P
OSITION X,5:? #6;" \ ":GOSUB SND
3510 NEXT X
3520 FOR I=1 TO 20:POKE 708,14:GOSUB S
ND:POKE 708,8:GOSUB SND
3530 NEXT I
4000 FOR X=5 TO 10:POKE 708,14:POSITIO
N X,5:? #6;" [ ":GOSUB SND:POKE 708,8:
POSITION X,5:? #6;" \ ":GOSUB SND
4010 NEXT X
4520 FOR I=1 TO 20:POKE 708,14:GOSUB S
ND:POKE 708,8:GOSUB SND
4530 NEXT I
5000 FOR X=11 TO 14:POKE 708,14:POSITI
ON X,5:? #6;" [ ":GOSUB SND:POKE 708,8
:POSITION X,5:? #6;" \ ":GOSUB SND
5010 NEXT X
6000 POSITION 15,5:? #6;"]":FOR D=14 T
O 10 STEP -1:SOUND 0,30,8,D:NEXT D
6010 POSITION 15,5:? #6;"^":FOR D=10 T
O 5 STEP -1:SOUND 0,100,8,D:NEXT D
6020 POSITION 15,5:? #6;"_":FOR D=5 TO
 0 STEP -1:SOUND 0,30,8,D:NEXT D
6030 POSITION 15,5:? #6;"_":FOR D=5 TO
 0 STEP -1:SOUND 0,30,8,D:NEXT D
6040 POSITION 15,5:? #6;"^":FOR D=10 T
O 5 STEP -1:SOUND 0,100,8,D:NEXT D
6050 POSITION 15,5:? #6;"]":FOR D=14 T
O 10 STEP -1:SOUND 0,30,8,D:NEXT D
6070 POKE 708,0:SOUND 0,0,0,0:SOUND 1,
0,0,0:SOUND 2,0,0,0
7000 GOTO 10
10000 SOUND 0,200,12,8:SOUND 2,203,12,
8:SOUND 1,RND(0)*10,10,8:RETURN
```

•

CHECKSUM DATA
(See pgs. 7-10)

```
10 DATA 772,278,748,506,876,319,217,87
2,803,219,878,993,224,877,233,8815
2120 DATA 222,883,396,227,344,928,573,
549,57,506,809,536,59,508,398,6995
5010 DATA 538,815,747,838,839,750,820,
756,623,551,7277
```

•

Assembly listing.

```
0100 ;CHARACTER DUMP BY JOE TREM
0110 OLD=$CC ;TEMP. LOCATION OF ATARI'S CHARACTER SET
0120 NEW=$CE ;TEMP. LOCATION OF NEW CHARACTER SET
0130 PAGE=$D4 ;NUMBER OF 256 BYTE BLOCKS
0140  *=$600
0150  PLA
0160  PLA ;PULL HIGH BYTE OF ADDR
0170  STA NEW+1
0180  PLA ;PULL LOW BYTE OF ADDR
0190  STA NEW
0200  PLA ;PULL HIGH BYTE-DON'T NEED
0210  PLA ;PULL NUMBER OF BLOCKS TO MOVE
0220  STA PAGE
0230  LDA #00 ;LOADS IN ATARI CHR.SET
0240  STA OLD
0250  LDA #$E0 ;ATARI CHR. SET IS AT $E000 OR 57344 IN BAS
0260  STA OLD+1
0270  LDX #1
0280  LDY #0
0290 LOOP LDA (OLD),Y
0300  STA (NEW),Y ;MOVES TO NEW AREA
0310  INY
0320  BNE LOOP
0330  INC OLD+1
0340  INC NEW+1
0350  INX
```

```
0360  CPX PAGE
0370  BNE LOOP
0380  RTS ;IF ALL BLOCKS ARE LOADED RETURN TO BASIC
0390   .END
```

Atari Symbol Demo

```
0 REM **********************
1 REM *                    *
2 REM *   ATARI SYMBOL     *
3 REM *  BY CRAIG WEISS    *
4 REM *                    *
5 REM **********************
10 GRAPHICS 24:COLOR 1:POKE 559,0
20 R=0
24 REM
25 REM *** PLOT STRAIGHT LINES ***
26 REM
30 READ W,X,Y,Z:PLOT W,X:DRAWTO Y,Z:R=
R+1
100 DATA 144,13,144,76,144,13,156,15,1
44,13,128,28,156,15,156,88,160,16,156,
20,160,16,160,176
110 DATA 160,16,180,20,180,20,180,176,
184,21,180,24,184,21,194,24,194,24,194
,84,240,154,240,172
120 DATA 240,172,220,172,180,176,160,1
76,160,176,144,176,144,176,144,144,88,
180,68,180,88,180,88,160
130 DATA 68,180,68,160,68,160,88,160,1
28,28,128,76,184,21,184,84
134 REM
135 REM *** PLOT FALSE CURVES ***
136 REM
140 DATA 128,77,126.5,94,126.5,94,124,
108,124,108,120,122,120,122,112,137,11
2,137,104,145
150 DATA 104,145,96,150,96,150,88,155,
88,155,80,158,80,158,72,160
160 DATA 144,76,142.5,94,142.5,94,140,
108,140,108,135,122,135,122,126,137,12
6,137,120,145
170 DATA 120,145,114,151,114,151,108,1
55.5,108,155.5,100,158.5,100,158.5,88,
160
180 DATA 156,88,153.5,112,153.5,112,15
0,128,150,128,144,144,143,144,136,156,
136,156,124,168
190 DATA 124,168,112,176,112,176,102,1
79,102,179,96,180,96,180,88,180
200 DATA 194,84,194,92,194,92,198,112,
198,112,208,130.5,208,130.5,216,141,21
6,141,224,148
210 DATA 224,148,232,152,232,152,240,1
54
220 DATA 184,84,186,104,186,104,189.5,
120,189.5,120,196,136,196,136,204,148
230 DATA 204,148,216,160,216,160,228,1
68,228,168,240,172
240 DATA 182,122,184,132,184,132,188,1
40,188,140,196,152,196,152,208,164,208
,164,220,172
250 IF R<68 THEN 30
260 IF R=68 THEN 500
310 REM
320 REM FILL
330 REM
400 Q=0
500 READ A,B,C,D:PLOT A,B:POSITION C,D
:Q=Q+1
900 POKE 765,1
910 XIO 18,#6,0,0,"S:"
1000 DATA 144,13,144,76,144,76,142.5,9
4,142.5,94,140,108,140,108,135,122,135
,122,126,137,126,137,120,145
1010 DATA 120,145,114,151,114,151,108,
155.5,108,155.5,100,158.5,100,158.5,88
,160,88,160,88,180
1020 DATA 160,16,160,176,184,21,184,84
1030 DATA 184,84,186,104,186,104,189.5
,120,189.5,120,196,136,196,136,204,148
1040 DATA 204,148,216,160,216,160,228,
168,228,168,239,171
2000 IF Q<20 THEN 500
2010 IF Q=20 THEN 2800
2500 REM
2510 REM *** MACHINE LANGUAGE ***
2520 REM
2800 POKE 559,34:FOR X=1 TO 1000:NEXT
X
3000 FOR I=1664 TO 1673:READ A:POKE I,
A:NEXT I
3010 DATA 232,142,10,212,142,24,208,76
,128,6
3020 ? USR(1664):RETURN
3030 RETURN
```

CHECKSUM DATA

(See pgs. 7-10)

```
0 DATA 552,194,141,406,200,562,472,981
,265,567,271,550,605,686,546,6998
130 DATA 16,87,243,89,84,805,118,371,6
01,330,184,540,938,889,345,5640
250 DATA 655,495,81,759,87,225,685,793
,764,530,838,34,78,73,592,6689
2010 DATA 864,292,243,294,870,63,617,1
61,786,4190
```

MULTIPROCESSING

16K Cassette or Disk

by Mark Chasin

No, this article will not enable you to set up a time-sharing service on your ATARI home computer, but it will demonstrate how to implement a form of multiprocessing which has been used in a number of recently released programs for the ATARI. To understand the principles of this program, you will need some background on how the video display operates.

The beam of electrons generated in the cathode ray tube of your TV set is focused and directed at the phosphors on the screen. The beam begins scanning the screen at the upper left corner, and proceeds across the screen from left to right. At the right edge, it returns to the left side and drops down one scan line, and proceeds to the right again. This process is repeated 262 times until the whole screen has been scanned, and then the beam is turned off and returned to the upper left corner to repeat the process again, sixty times a second.

This seems like a great deal to handle in one-sixtieth of a second, but your ATARI has a machine cycle time of 560 nanoseconds, so in that time interval, the ATARI can execute approximately 30,000 cycles. The result of this is that when the beam returns to the upper left corner of the screen, there is a good deal of time to waste before it must start scanning again. At this point, the ATARI goes off on its own, performing a number of housekeeping functions, updating timers and the like. Ultimately, it returns to the business of drawing on the screen.

The folks who built your ATARI designed the system so that it could be modified easily by anyone wanting to do so, and the remainder of this article will discuss such a modification. The computer "knows" where to go during the wait described above because two memory locations contain a hexadecimal address telling it where to go, and every time it gets to the upper left corner of the display, it looks in these memory locations and goes to the indicated address, where the housekeeping routines are stored. This process is called vectoring. There are actually two independent routines performed during each interval, and separate vectors exist for each. The immediate vertical blank vector is found at hexadecimal address $0222 and $0223, and the second vector, called the deferred vertical blank vector, is found at $0224 and $0225. What we are about to do is change the address located at $0224, $0225 to point to our own routine, and then we'll jump back into the routine that the ATARI was originally pointing to. When this is accomplished, our routine will execute 60 times per second, and will continue to execute until we either turn off the computer or hit SYSTEM RESET. This will be totally independent of anything else we may be doing at the time, such as programming, editing, or playing a game!

The BASIC program shown in **Figure 1** is simply an implementation in BASIC of the Assembly language program shown in **Figure 2**, so I will describe the operation of the Assembly language program in detail. First, I will list the locations and their uses within the program.

COUNT 1 — used to determine how many times we have gone through the routine, to calculate when to start and stop the notes to be played.

COUNT 2 — used to remember which note the routine is playing.

VVBLKD — the location of the deferred vertical blank vector.

SETVBV — an ATARI routine, described in more detail below.

MUSIC — the location where the list of notes to be played is stored.

RETURN — this is where we need to jump to return to the ATARI housekeeping routines.

SND — the frequency register for SOUND 0.

VOL — the distortion and volume register for SOUND 0.

Lines 130 and 170-190 are housekeeping functions of this routine. Line 130 provides the PLA instruction necessary for accessing the routine from BASIC, and lines 170-190 set both COUNTs to 0. Lines 230-270 repoint the delayed vector to our routine, as follows. Since the 6502A inside your ATARI is an 8 bit processor, we can only handle one

byte at a time. It should be obvious that if the computer tries to access this vector after we have changed one byte of the address, but before we have changed the second, the computer will go on a wild goose chase looking for where it should be. To prevent this, those clever folks who wrote the operating system for your computer built in a routine, called SETVBV, which will change these vectors without the chance of fouling things up. To use it, we load the Y register with the new vector low byte, and the X register with the new vector high byte, $20 and $06 respectively in this case, since our routine is loaded at $0620. We then load the accumulator with a 7 if we are setting the deferred vector, or a 6 if the immediate vector, and then we JSR to the subroutine SETVBV. Presto! Our vector is changed, and the routine starts operating.

This routine will play a little familiar background music while you slave away over a hot computer. Later on, I'll describe how to change the tune to your own selection. The routine starts on line 320. This is the first time through, so we increase COUNT 1 to one. If COUNT 1=12, we'll turn the note off, and when COUNT 1=15, we'll play the next note, and reset COUNT 1 to 0. Lines 360-370 shut off the note, lines 410-420 reset the count to 0, and lines 430-470 play the next note. The tune consists of eight notes repeated over and over, and COUNT 2 keeps track of which note is being played. When it gets up to 8, it's reset to 0 (lines 480-530), so the first note is played right after the eighth. If COUNT 1 is not equal to either 12 or 15, the routine ends and returns to the normal housekeeping functions performed by the ATARI during the vertical blank period (line 400). Also, after a new note is started, the same thing happens (line 540). The table of notes played in the tune is located in line 590.

The BASIC program in **Figure 2** simply converts the instructions described above into decimal form, and POKEs the routines into the correct place in memory. The routine is then set in motion with the USR call in line 27000, and from that point on, can be ignored. It will continue by itself!

Changing the tune being played is very simple. Choose a song in which all the notes are the same length, e.g., quarter notes. In line 24000, change the 1639 to (1632+the number of notes in your tune-1), replace the data in lines 25000 and 26000 with the notes for your tune, and change the 8 in line 22000 to the number of notes in your tune. Remember, the tune will play over and over, so pick something which sounds good on repetition.

The routine presented here can be ended by a power-off, power-on sequence, or by a SYSTEM RESET. A third method, probably more useful for use in a program, is this:

```
POKE 1562,104:POKE 1544,98:POKE 1546,2
28:X=USR(1542):SOUND 0,0,0,0
```

This is a simple demonstration of the use of vertical blank interrupt routines. There are many other potential uses for this approach, such as background music for another program, checking for keyboard or joystick input during a program, or implementation of multitasking. It is perfectly feasible to have two completely separate programs running "simultaneously," but the programming for this gets fairly complicated. One program would run in real time, and the other during the vertical blank interrupt routines. Play around with the ideas presented here, and learn all about simultaneous processing. □

Figure 1.

```
8000 REM ********************  FIRST,
 WE'LL POKE IN THE LINES FROM 130-270
OF THE ASSEMBLY LISTING ***********
9000 RESTORE 10000:FOR I=1536 TO 1552:
READ A:POKE I,A:NEXT I
10000 DATA 104,169,0,133,192,133
11000 DATA 194,160,32,162,6,169
12000 DATA 7,32,92,228,96
13000 REM ******************************
 THEN WE'LL POKE IN THE MAIN ROUTINE
**********************************
14000 RESTORE 15000:FOR I=1568 TO 1619
:READ A:POKE I,A:NEXT I
15000 DATA 230,192,166,192
16000 DATA 224,12,144,5,169,0
17000 DATA 141,1,210,224,15,176
18000 DATA 3,76,98,228,169,0
19000 DATA 133,192,166,194,189,96
20000 DATA 6,141,0,210,169,166
21000 DATA 141,1,210,230,194,166
22000 DATA 194,224,8,144,4,169
23000 DATA 0,133,194,76,98,228
23500 REM ******************************
FINALLY, WE'LL POKE IN THE TABLE OF NO
TES ********************************
24000 RESTORE 25000:FOR I=1632 TO 1639
:READ A:POKE I,A:NEXT I
25000 DATA 243,243,217,243,204,243
26000 DATA 217,243
26500 REM ******************************
NOW WE'LL RUN THE ROUTINE!!**********
27000 X=USR(1536)
```

•

CHECKSUM DATA
(See pgs. 7-10)

```
8000 DATA 133,466,426,611,26,547,951,8
74,164,587,365,735,335,421,397,7038
23000 DATA 403,616,945,911,197,274,175
,3521
```

•

Figure 2.

```
10  *= $0600
20 COUNT1 = $00C0
30 VVBLKD = $0224
40 COUNT2 = $00C2
50 SETVBV = $E45C
60 MUSIC = $0660
70 RETURN = $E462
80 SND = $D200
90 VOL = $D201
0100 ;
```

```
0110 ;   PLA FOR BASIC ACCESS
0120 ;
0130  PLA
0140 ;
0150 ;   INITIALIZE COUNTERS TO ZERO
0160 ;
0170  LDA #0
0180  STA COUNT1
0190  STA COUNT2
0200 ;
0210 ;   NOW RESET DEFERRED VECTOR
0220 ;
0230  LDY #$20
0240  LDX #$06
0250  LDA #07
0260  JSR SETVBV
0270  RTS
0280 ;
0290 ;   MAIN INTERRUPT ROUTINE
0300 ;
0310  *= $0620
0320  INC COUNT1
0330  LDX COUNT1
0340  CPX #12 ;TIME TO STOP NOTE?
0350  BCC K1 ;NO
0360  LDA #0 ;YES, SO STOP IT
0370  STA VOL
0380 K1 CPX #15  ;15/60 SECONDS GONE?
0390  BCS PLAY  ;YES, PLAY NEXT NOTE
0400  JMP RETURN  ;NO, END INTERRUPT
0410 PLAY LDA #0
0420  STA COUNT1 ;RESET COUNT1 TO ZERO
0430  LDX COUNT2 ;GET NOTE TO PLAY
0440  LDA MUSIC,X ;LOOK IT UP
0450  STA SND ;SET IT'S FREQUENCY
0460  LDA #$A6
0470  STA VOL ;SET PURE NOTE,VOLUME=6
0480  INC COUNT2 ;SET UP NEXT NOTE
0490  LDX COUNT2
0500  CPX #8  ;ALL NOTES USED UP?
0510  BCC DONE ;NO
0520  LDA #0 ;YES, START OVER AGAIN
0530  STA COUNT2
0540 DONE JMP RETURN  ;ALL DONE
0550 ;
0560 ;   TABLE OF MUSICAL NOTES
0570 ;
0580  *= $0660
0590  .BYTE 243,243,217,243,204,243,217,243
```

●

Graphics 10 GTIA Demo

```
10 REM GRAPHICS 10 GTIA DEMO
20 REM
30 GRAPHICS 10
40 REM CHANGE DATA TO CHANGE COLORS
50 FOR CN=0 TO 7:READ CV:POKE 705+CN,C
V:NEXT CN:DATA 6,12,23,42,53,62,73,84
60 C=0:SETCOLOR 4,C,0
70 FOR X=0 TO 39
80 FOR Y=0 TO 95
90 XW=39-X:YW=95-Y:DIST=INT(SQR(XW*XW+
YW*YW))
100 COLOR 1+8*(DIST/8-INT(DIST/8))
110 PLOT X,Y
120 PLOT 79-X,Y
130 PLOT X,191-Y
140 PLOT 79-X,191-Y
150 NEXT Y
160 NEXT X
170 REM ROTATE COLOR REGISTERS
180 CHOLD=PEEK(705)
190 X=705
200 POKE X,PEEK(X+1)
210 X=X+1:IF X<712 THEN 200
220 POKE 712,CHOLD
230 GOTO 180
```

CHECKSUM DATA

(See pgs. 7-10)

```
10 DATA 989,253,992,750,196,890,294,29
7,224,512,822,151,427,756,775,8328
160 DATA 776,485,793,323,635,532,502,7
18,4764
```

GRAPHICS

MOVING PLAYERS IN BASIC

16K Cassette or Disk

by Tom Hudson

Player-missile graphics are one of the most powerful graphic features of ATARI personal computer systems. Unlike traditional graphics, players and missiles can be moved around on the screen without disturbing the existing display.

In order to use players and missiles, one must first reserve a portion of memory. Once this is done, the user can begin designing and displaying the players and missiles.

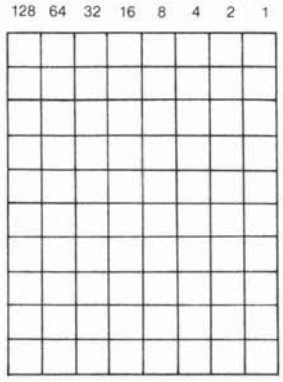
The problems begin when the user wants to move a player or missile around on the screen. Horizontal movement is done easily. A POKE to the appropriate horizontal position memory location will move the desired player to any horizontal location on the screen. If the user wants to move a player or missile vertically, he or she must copy the P/M bit image to another location in memory. BASIC is too slow to do this smoothly, but it can call a machine-language subroutine to do the "dirty work."

Designing Players

Before we start using the player movement subroutine, we must have some sort of graphic image to place in the player.

Players are eight pixels (picture elements) wide, so the first step in designing the player image is to draw a matrix eight cells across and as tall as the desired image. In the "two line" resolution player mode (each pixel in the player is two television scan lines tall), the player can be up to 128 pixels high. The computer can display players with pixels one scan line tall, but the one-line resolution requires twice the memory of the two-line mode. This demonstration uses the 2 line resolution in order to save memory. For our purposes, we will set up an 8 x 10 matrix to design the player image (**Figure 1**).

Figure 1.



If you look at **Figure 1,** you will notice numbers over each column in the matrix. These numbers range from 1 on the right to 128 on the left. These numbers will be used to create a DATA statement that will represent a player image.

Figure 2.

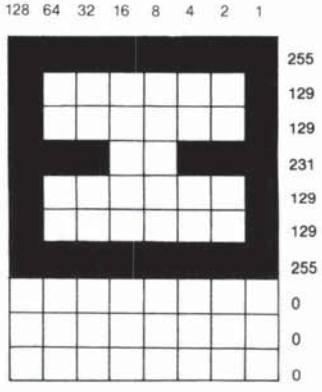


Figure 2 shows the simple player image used in the demonstration program following this article. The number to the right of each row is the total of the column numbers in which a pixel is "on." If all pixels in a row are on, the number is 255 (128+64+32+16+8+4+2+1). If no pixels are on, the total would be zero. You will note that the player image in figure 2 is seven pixels tall, meaning that in order to display this player image we will have to move seven bytes to player memory. Try designing your own player images using this method. Remember that players using the two-line resolution mode can be up to 128 pixels tall.

The program.

Once you have designed your player images, you are ready to display them with the computer. The BASIC program in **Listing 1** will move all four players around on the screen. It calls the P/M movement assembly language routine, shown in **Listing 2.**

As listed, the program will move the shape designed in **Figure 2** around on the screen at random. The shape of the player is stored as a series of bytes in the string PO$. By placing your player image data in line 420, you can change the shape that appears on the screen. There are currently seven bytes in line 420, but if your player image has a different number, place the appropriate value in lines 130 and 290.

Lines 110-180 — Set up the subroutine and turn on the P/M graphics.

Lines 220-230 — are for demonstration purposes only. You can put your program code in this section.

Line 110 — Loads the string PMMOV$ with the P/M movement subroutine.

Line 130 — Places the data that defines the graphics image into the string P0$. If your player image is more or less than seven pixels tall, place the appropriate value in this line.

Line 140 — This line tells the system where the P/M memory is located.

Line 150 — This line saves the address of the string that holds the player image data.

Line 170 — Turns on P/M direct memory access so that the image will appear on the screeen.

Line 180 — Sets the color of player 0 to blue. The value 136 is derived by multiplying the color number (8) by 16 and adding the luminance value (8). The result is (8*16)+8 or 136.

Line 220 — Initializes the X and Y coordinates of the player. The coordinates refer to the upper left corner of the player. The X coordinate may range from 0-255, and the Y coordinate from 0-127.

Lines 230-280 — This section simply changes the player's coordinates randomly.

Line 290 — This USR call moves the player to the desired X and Y location. This statement has 7 parameters inside the USR parentheses:

```
A=USR(MOVE,0,PMB,PMD,X,Y,7)
```

"MOVE" is set up in line 110. It is the address of the P/M mover subroutine. Do not change this value.

"0" means that we want to move player zero. This value can range from 0-3, moving any one of the four players.

"PMB" is the P/M base address set up in line 150. Do not change this value.

"PMD" is the address of the string that holds the player image data. This should be set to the address of the string you are using to hold your player shape data. If your player shape data is in a string called "PL$," you could replace PMD with ADR(PL$).

The X and Y variables are the horizontal and vertical coordinates of the player.

The last parameter, "7," indicates that the player we are displaying is 7 pixels tall (see lines 130 and 420). If the player you design is 10 bytes long, place a 10 here.

Line 300 — This line determines when to randomly change the player's movement direction. If a random number is chosen that is greater than .95, a new direction is tried.

Line 310 — This line loops back to line 240 if no new direction is needed.

Lines 350-380 — These lines contain the assembly-language code for the player movement subroutine. Do not change these lines, or the subroutine will probably not work.

Line 420 — This line contains the values which represent the player image's shape. Place your image values here.

Summary.

The ATARI computer systems' player-missile graphics capabilities are actually very easy to use, given the proper tools. The subroutine presented here will help even the beginning ATARI programmer experience the wonders of player-missile graphics. □

Listing 1.

```
10 REM ********************************
20 REM * P/M MOVER SUBROUTINE DEMO *
30 REM *                           *
40 REM *        BY TOM HUDSON      *
50 REM *                           *
60 REM *  A.N.A.L.O.G. COMPUTING   *
70 REM ********************************
80 REM
90 REM ********** SETUP **********
100 REM
110 DIM PMMOV$(100),P0$(30):MOVE=ADR(P
MMOV$):FOR X=1 TO 100:READ N:PMMOV$(X)
=CHR$(N):NEXT X:REM *READ ML DATA*
120 REM *** NOW READ SHAPE DATA ***
```

```
130 FOR X=1 TO 7:READ N:P0$(X)=CHR$(N)
:NEXT X
140 PMBASE=INT((PEEK(145)+3)/4)*4:POKE
 54279,PMBASE:REM *** SET UP P/M AREA
***
150 PMB=PMBASE*256
160 PMD=ADR(P0$):REM *** P/M DATA ADDR
ESS ***
170 POKE 559,46:POKE 53277,3:REM *** P
/M DMA ***
180 POKE 704,136:REM *** PLAYER 0 COLO
R ***
190 REM
200 REM **** YOUR PROGRAM HERE! ****
210 REM
220 X=128:Y=64
230 XI=1-INT(RND(0)*3):YI=1-INT(RND(0)
*3)
240 X=X+XI:Y=Y+YI
250 IF X<50 THEN X=50:GOTO 270
260 IF X>190 THEN X=190
270 IF Y<20 THEN Y=20:GOTO 290
280 IF Y>110 THEN Y=110
290 A=USR(MOVE,0,PMB,PMD,X,Y,7)
300 IF RND(0)>0.95 THEN 230
310 GOTO 240
320 REM
330 REM *** PM MOVER DATA ***
340 REM
350 DATA 216,104,104,104,133,213,104,2
4,105,2,133,206,104,133,205,104,133,20
4,104,133,203,104,104,133,208
360 DATA 104,104,133,209,104,104,24,10
1,209,133,207,166,213,240,16,165,205,2
4,105,128,133,205,165,206,105
370 DATA 0,133,206,202,208,240,160,0,1
62,0,196,209,144,19,196,207,176,15,132
,212,138,168,177,203,164
380 DATA 212,145,205,232,169,0,240,4,1
69,0,145,205,200,192,128,208,224,166,2
13,165,208,157,0,208,96
390 REM
400 REM *** PLAYER IMAGE DATA ***
410 REM
420 DATA 255,129,129,231,129,129,255
```

●

CHECKSUM DATA

(See pgs. 7-10)

```
10 DATA 532,930,996,64,0,483,544,265,9
09,74,765,328,743,901,536,8070
160 DATA 729,778,445,101,552,79,854,96
8,479,920,983,921,954,424,374,9561
310 DATA 704,84,580,90,639,732,435,188
,105,191,83,56,3887
```

●

Listing 2.

```
01
0100 ;
0110 ;PLAYER-MISSILE MOVER SUBROUTINE
0120 ;
0130 ;BY TOM HUDSON
0140 ;A.N.A.L.O.G. COMPUTING
0150 ;
0160 ;
0170 ;PAGE ZERO USAGE
0180 ;
0190 PMSTR = $CB          ;P/M BASIC STRING
0200 PLADR = $CD          ;PLAYER ADDRESS
0210 PMEND = $CF          ;PLAYER IMAGE END
0220 XPOS = $D0           ;X POSITION
0230 YPOS = $D1           ;Y POSITION
0240 HOLD = $D4           ;HOLD AREA
0250 PLNUM = $D5          ;PLAYER # TO MOVE
0260 ;
0270 ;OPERATING SYSTEM EQUATES
0280 ;
0290 HPOSP0 = $D000
0300 ;
0310 ;PROGRAM STARTS HERE!
0320 ;
0330  *= $6000            ;ANY ADDRESS
0340 START CLD            ;CLEAR DECIMAL MODE
0350  PLA                 ;DISCARD
0360  PLA                 ;DISCARD # HI
0370  PLA                 ;PULL PLAYER # LO
0380  STA PLNUM           ;AND SAVE IT!
0390  PLA                 ;PULL P/M BASE HI
0400  CLC                 ;ADD OFFSET TO GET
0410  ADC #2              ;PLAYER MEMORY ADDR
0420  STA PLADR+1         ;AND SAVE!
0430  PLA                 ;PULL P/M BASE LO
0440  STA PLADR           ;AND SAVE!
0450  PLA                 ;PULL STRING HI
0460  STA PMSTR+1         ;AND SAVE!
0470  PLA                 ;PULL STRING LO
0480  STA PMSTR           ;AND SAVE!
0490  PLA                 ;DISCARD X HI
0500  PLA                 ;PULL X LO
0510  STA XPOS            ;AND SAVE IT!
0520  PLA                 ;DISCARD Y HI
0530  PLA                 ;PULL Y LO
0540  STA YPOS            ;AND SAVE IT!
0550  PLA                 ;DISCARD LENGTH HI
0560  PLA                 ;PULL LENGTH LO
0570  CLC                 ;ADD Y POSITION
0580  ADC YPOS            ;TO GET END
0590  STA PMEND           ;AND SAVE IT!
0600  LDX PLNUM           ;GET PLAYER#
0610  BEQ ENDCAL          ;NO INDEX NEEDED!
0620 PLCALC LDA PLADR     ;ADD 128 TO
0630  CLC                 ;PLAYER
0640  ADC #128            ;ADDRESS
0650  STA PLADR           ;TO
0660  LDA PLADR+1         ;POINT TO
0670  ADC #0              ;NEXT
0680  STA PLADR+1         ;PLAYER.
0690  DEX                 ;ANOTHER ADJUSTMENT?
0700  BNE PLCALC          ;YES!
0710 ENDCAL LDY #0        ;ZERO P/M COUNT
0720  LDX #0              ;ZERO STRING COUNT
0730 COPYLP CPY YPOS      ;COPYING DATA YET?
0740  BCC ZERO            ;NO!
0750  CPY PMEND           ;FINISHED COPYING?
0760  BCS ZERO            ;YES!
0770  STY HOLD            ;SAVE Y REG
0780  TXA                 ;MOVE X REG...
0790  TAY                 ;TO Y REGISTER
0800  LDA (PMSTR),Y       ;GET P/M BYTE
0810  LDY HOLD            ;GET P/M OFFSET
0820  STA (PLADR),Y       ;CHANGE PLAYER!
0830  INX                 ;NEXT STRING BYTE.
0840  LDA #0              ;FORCE BRANCH
0850  BEQ NEXT            ;TO NEXT BYTE!
0860 ZERO LDA #0          ;ZERO OUT...
0870  STA (PLADR),Y       ;PLAYER BYTE!
0880 NEXT INY             ;NEXT P/M BYTE
0890  CPY #128            ;DONE W/COPY?
0900  BNE COPYLP          ;NOT DONE YET!
0910  LDX PLNUM           ;GET PLAYER #
0920  LDA XPOS            ;NOW JUST SET
0930  STA HPOSP0,X        ;X LOCATION!
0940  RTS                 ;FINIS!
0950  .END
```

●

USING DLIs

16K Cassette or Disk

by Joseph T. Trem

For many years there have been powerful computers on the market which performed multi-tasking functions. Not until a few years ago did the home computer acquire this capability. At last! ATARI!

Having a 6502 microprocessor for its brain, your ATARI computer has the capability of using interrupts. An interrupt is a tricky way of freezing the state of the microprocessor while performing some other function, then moving on when completed.

Here is an example. On a raster scan TV, the picture you see is drawn sixty times a second. The beam starts in the upper left-hand corner and eventually ends up in the lower right-hand corner. This is done sixty times a second. The time taken for the beam to travel from the bottom of the screen back to the top is called vertical blank. During vertical blank, there is plenty of time for other processing. Using an interrupt, one could check for vertical blank. When vertical blank occurs, it is possible to perform some other function, then continue on. Some of the more common functions would be moving player/missiles, updating score counters, changing colors. . . all between Vblank, as it is more commonly called. If these functions are performed during Vblank, there is no unsightly flicker on the screen. Besides, Vblank is processing time to kill, right?

ATARI goes a step further by implementing a display list interrupt or DLI. On a raster scan TV, the beam sweeps across the screen, from left to right, moves down one line, then does it again. One sweep of the beam is one scan line. It takes 262 sweeps of that beam to create a single frame on your TV, all done sixty times a second. In other words, there are 262 scan lines available on your TV.

ATARI designed their computer to evolve around the architecture of your TV set. Even better, the display list uses all combinations of the scan line from graphics 0 to graphics 8 and allows you to set up a DLI on any line. For example, one could draw a scan line, change the background color, and so on. The final picture will appear to have a different color on each line.

To set up a DLI, there are a few steps which have to be taken. First, we have to create a DLI routine in machine language that will do what we want. This is called a service routine. Then we must let the microprocessor know where to find that routine by vectoring through $200 (low byte) and $201 (high byte). That's 512 and 513 decimal. Next, we set the display list lines that we want the service routine to occur after with a DLI instruction. Finally, we must enable the DLI.

Because the concept of the DLI is a hard one to follow and needs some understanding of Assembly language, I have presented an example. . . a picture is worth a thousand words! This program, written in BASIC, twinkles a starfield while running player/missiles. . . both appearing independent of one another.

The program is well documented. In the example, the service routine is located at $600. Every display list line has been set in the graphics 7 mode with a DLI instruction (**Figure 1**). This was determined by using the chart in **Figure 2**. The DLI instruction for graphics 7 is 141 decimal. Included with the BASIC program is the assembled listing of the service routine which simply stuffs colors in the color register.

Hopefully, this program will help you gain a better understanding of the DLI. It is among the most powerful programming tools you can use. Take some time to understand the concept, and you will greatly increase your programming expertise. □

Standard Graphics 7 Display List		New Graphics 7 Display List with DLI Set	
70	8 Blank lines	70	8 Blank lines
70	8 Blank lines	70	8 Blank lines
70	8 Blank lines	70	8 Blank lines
4D	Antic Mode 13 (Basic mode 7)	4D	Antic Mode 13 (Basic mode 7)
60		60	
70		70	
0D		8D	Antic Mode 13 with DLI set
0D		8D	
0D		8D	
0D		8D	
0D		8D	

Figure 1.

Display List Interrupt Instruction Chart

Graphics Mode		DLI Instruction	
Basic	Antic	Hex	Decimal
0	$02	$82	130
None	$03	$83	131
None	$04	$84	132
None	$05	$85	133
1	$06	$86	134
2	$07	$87	135
3	$08	$88	136
4	$09	$89	137
5	$0A	$8A	138
6	$0B	$8B	139
None	$0C	$8C	140
7	$0D	$8D	141
None	$0E	$8E	—
8	$0F	$8F	143

Figure 2.

```
100 REM FLICKERING STARFIELD
110 REM BY JOE TREM (c) 1982
120 REM
130 REM SETS GRAPHICS 7 FULL SCREEN, D
RAWS SURFACE WITH SOUND
140 GRAPHICS 23:POKE 708,136:COLOR 1:F
OR X=0 TO 159:SOUND 0,10,X,4:PLOT X,95
:DRAWTO X,80+RND(0)*5:NEXT X
150 REM CALCULATES DISPLAY LIST, SETS
SPEED OF PLAYER TO 0
160 SP=0:DLST=PEEK(560)+PEEK(561)*256
170 REM SETS UP DLI FOR EACH GRAPHICS
7 SCAN LINE
180 FOR L=6 TO 84:POKE DLST+L,141:NEXT
 L
190 REM READS MACHINE LANGUAGE ROUTINE
 INTO PAGE 6
200 FOR J=0 TO 3:READ A:POKE 1536+J,A:
NEXT J
210 COLOR 3:REM SETS COLOR TO FLICKER
220 REM PLOTS STARS WITH SOUND
230 FOR X=1 TO 50:SOUND 0,X,X,4:PLOT R
ND(0)*159,RND(0)*75:NEXT X
240 REM SETS STARTING ADDRESS FOR DLI
(PAGE 6) AND ENABLES DLI
250 POKE 512,0:POKE 513,6:POKE 54286,1
92
260 REM SETS UP PLAYER/MISSILE 0
270 YP=0:POKE 559,62:PMBAS=PEEK(106)-3
2:POKE 54279,PMBAS:POKE 53277,3:PM0=PM
BAS*256+1024
280 GOSUB 350
290 REM PLAYER/MISSILE COLOR, MOVE RIG
HT
300 POKE 704,INT(RND(0)*15)*16+8:FOR X
=30 TO 230 STEP SP:POKE 53248,X:SOUND
0,X,8,4:NEXT X
310 GOSUB 350
320 REM PLAYER/MISSILE COLOR, MOVE LEF
T
330 POKE 704,INT(RND(0)*15)*16+8:FOR X
=230 TO 30 STEP -SP:POKE 53248,X:SOUND
 0,X,8,8:NEXT X:GOTO 280
340 REM ROUTINE ERASES OLD PLAYER, DET
ERMINES SPEED, AND VERTICAL LOCATION O
F PLAYER 0
350 SP=SP+1:FOR X=YP TO YP+4:POKE PM0+
X,0:NEXT X:IF SP>15 THEN SP=1
360 YP=30+RND(0)*150:POKE PM0+YP,24:PO
KE PM0+YP+1,255:POKE PM0+YP+2,255:POKE
 PM0+YP+3,24:RETURN
370 REM MACHINE LANGUAGE DATA
380 DATA 142,24,208,64
390 REM NOTE TO ASSEMBLY PROGRAMMERS..
WSYNC WAS NOT USED FOR MORE ERRATIC FL
ICKERING
```

•

CHECKSUM DATA
(See pgs. 7-10)

```
100 DATA 190,749,80,569,86,100,793,712
,498,713,519,482,263,294,178,6226
250 DATA 994,406,125,990,763,160,971,4
16,696,808,181,59,947,793,791,9100
```

•

Assembly listing.

```
0100 ; FLICKERING STARFIELD
0110 ; DLI SERVICE ROUTINE
0120 ;
0130 ; ADDRESS $D018 IS THE
0140 ; COLOR/LUMINANCE REGISTER
0150 ; OF PLAYFIELD 2
0160 ;
0170 COLPF2 = $D018
0180 ;
0190  *=$600
0200 ;
0210 ; SAVE WHATEVER IS IN THE
0220 ; X-REGISTER INTO PLAYFIELD
0230 ; COLOR 2 HARDWARE REGISTER
0240 ;
0250  STX COLPF2 ; STORE COLOR
0260  RTI ; RETURN FROM INTERRUPT
0270 ;
0280  .END
```

•

A GRAPHICS CLIPPING ROUTINE

16K Cassette or Disk

by Tom Hudson

Probably every ATARI user who has ever dabbled in the graphics area has encountered the infamous "ERROR 141 — CURSOR OUT OF RANGE." This error message occurs when you try to PLOT or DRAWTO a point which is off the screen. The program listings presented in this article will demonstrate a BASIC subroutine which eliminates this problem, while drawing the portion of the line which is on the screen.

Listing 1 is the clipping routine. Type in this subroutine and check it for typing errors. **List** this onto tape (**LIST "C:"**) or disk (**LIST "D:filenam**), so that it can be easily merged with other programs.

Listing 2 is a demonstration of the clipping routine's capabilities. This program is a general-purpose shape rotation routine and will be explained in detail later. Type NEW and enter this listing into your computer, then check it for typing errors.

When you are sure **Listing 2** has been entered correctly, ENTER the clipping routine from tape (ENTER "C:") or disk (ENTER "D:filename"). The two listings will merge, forming one program. RUN the program. You will see a square appear. It will begin rotating and increase in size until its corners run off the screen completely, and it disappears altogether. Press BREAK to stop the program.

How it works.

Line 150 — This line sets the BASIC DEGREE flag. This tells the computer that all angles will be expressed in degrees.

Line 160 — This line sets up a full-screen GRAPHICS 6 screen.

Line 170 — This line tells the computer to use color 1 when drawing.

Line 180 — This line sets the shape size increment (SI) to 1.1. This means that each time the shape is drawn, it will be 1.1 times as large as the previous plot. If SI is set to 1, the shape will stay the same size. If SI is set to 0.5, the shape will shrink to half its size each time it is drawn.

Line 190 — This line establishes the initial size of the shape. Since SF is set at 0.5, the object will start out half as big as defined.

Line 200 — This line sets RF, the rotation factor, to 10. With this value, the shape will rotate 10 degrees counter-clockwise each time it is drawn. A negative value will rotate it clockwise, and a value of zero will result in a non-rotating shape.

Line 210 — This line defines CX and CY, the center coordinates of the object. The present values will place the object at the center of the screen. Try other values here and observe the results.

Line 220 — This line is essential to the operation of the clipping routine. It defines the limits of the screen area you wish to use. These values are currently set to the normal GRAPHICS 6 screen limits (X RIGHT=159, X LEFT=0, Y BOTTOM=95, Y TOP=0). By changing these values, a smaller "window" may be created. For example, make the following changes to line 220:

```
220 XR=80:XL=40:YB=50:YT=30
```

RUN the program and observe the result. The shape will be clipped to the new window limits. By using this technique, very interesting displays can be created with independent clipping windows!

Line 230 — This line sets the DATA pointer to line 360. This line contains the data which defines the shape of the object.

Line 240 — This line reads the number of points in the shape and dimensions X and Y coordinate work arrays accordingly.

Line 250 — This line reads the X and Y coordinates of each point in the shape and scales them as requested in line 190.

Lines 260-270 — These lines increment the rotational position of the object. Rotation values greater than 360 degrees are adjusted

properly.

Line 280 — This line adjusts the size of the shape as requested in line 180.

Lines 290-300 — These lines rotate each point in the shape using the BASIC functions SIN and COS (sine and cosine). The adjusted points are stored in the X2 and Y2 arrays.

Line 310 — This line clears the screen for the next plot. If this line is removed, the images of the rotating square will build up into an interesting display.

Line 320 — This line adjusts each point in the shape to its proper screen position by adding the centerpoint coordinates (defined in line 210).

Lines 330-340 — These lines are very important, as they send the PLOT AND DRAWTO coordinates to the clipping routine. The clipping routine requires four variables: X1, Y1, X2 and Y2. The routine analyzes the coordinates and simulates the function:

```
PLOT X1,Y1:DRAWTO X2,Y2
```

To see what happens when the clipping routine is not used, replace the GOSUB 1000 statements in lines 330 and 340 with PLOT X1, Y1: DRAWTO X2, Y2 and RUN the program. The program will operate correctly until the square runs off the screen. When this happens, the program will end with an error condition.

Line 350 — This line simply loops back to line 260, where the drawing process starts again.

Line 360 — This DATA statement contains information about the shape we want to draw. The first number is the number of points in the object. Since this is a square we are using, there are 4 points. The rest of the data values are the X and Y coordinate pairs for each point. To make a hexagon, for example, try this data statement:

DATA 6,11,0,6,-10,-6,-10,-11,0,-6,10,6,10

Figure 1 is an X-Y coordinate grid which is helpful in defining a shape. The shape rotates around the intersection of the X and Y axes (0,0) which in this case is the center of the square. You can set up any shape you like merely by changing this DATA line.

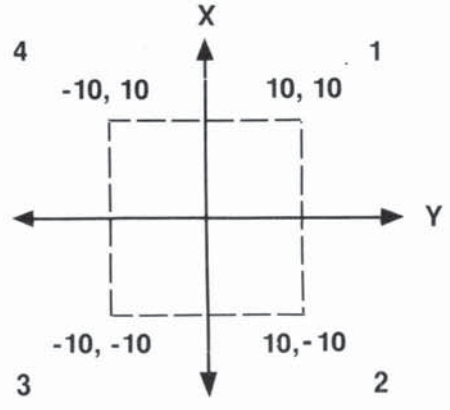


Figure 1.

The clipping routine.

Line 1050 — This line clears all the flags which determine when clipping is necessary.

Lines 1060-1130 — These lines check the X and Y coordinates to see if they have exceeded the screen limits defined in line 220 of the shape rotation demonstration. If the coordinates exceed the limits, flag variables are set to indicate this.

Line 1140 — If both X coordinates are to the left or right of the screen, or both Y coordinates are to the top or bottom of the screen, the line will not show up on the screen at all, and the plot is abandoned.

Line 1150 — This line sets up work variables to clip one end of the line, if necessary, and GOSUBs to line 1210 to perform this function.

Line 1160 — In order to clip the other end of the line, this line copies the second set of coordinate flags to the first.

Line 1170 — This line saves the XW and YW values, which are the last set of clipped endpoints. It then sends the X and Y endpoints to the clipping calculator and clips the other end of the line.

Line 1180 — If any of the clipped points could not be placed within the clipping area, the plot is abandoned.

Line 1190 — This line PLOTs and DRAWs the clipped line.

Line 1200 — This line exits the clipping routine after the clipped line is drawn.

Line 1210 — This line is the start of the clipping calculator, the heart of the clipping routine. If the total of the clipping off-screen flags is zero, no clipping is required. The XW and YW values are set up, and the clipping routine is exited.

Line 1220 — If the line goes past the left side of the screen, this line calculates the point at which the line crosses the left limit, and saves the X and Y coordinates of that point. If this point is on the screen then the calculation is complete and the subroutine is exited.

Line 1230 — If the line goes past the right side of the screen, this line calculates the point at which the line crosses the right limit, and saves the X and Y coordinates of that point. If this point is on the screen then the calculation is complete and the subroutine is exited.

Line 1240 — If the line goes past the bottom of the screen, this line calculates the point at which the line crosses the bottom limit, and saves the X and Y coordinates of that point. If this point is on the screen then the calculation is complete and the subroutine is exited.

Line 1250 — If the line goes past the top of

the screen, this line calculates the point at which the line crosses the top limit, and saves the X and Y coordinates of that point. If this point is on the screen then the calculation is complete and the subroutine is exited.

Line 1260 — This line forces a return from the subroutine after all calculations are complete.

Final comments.

The graphics clipping routine can be used in many graphics applications where it is possible to exceed screen limits. This routine can be used with any graphics mode, and can allow the use of graphics "windows" anywhere on the screen.

To use the clipping routine in your own programs, simply use lines 1000-1260 and set up the desired screen limits to the XL, XR, YT, and YB variables. When you want to draw a line, instead of the command:

```
PLOT X1,Y1:DRAWTO X2,Y2
```

use the following:

(Set up X1, Y1, X2, and Y2)
GOSUB 1000

This will work for any line, even those that are completely off the screen. □

Listing 1.

```
100 REM ************************
110 REM * SHAPE ROTATION DEMO *
120 REM *                    *
130 REM *    BY TOM HUDSON    *
140 REM ************************
150 DEG
160 GRAPHICS 6+16
170 COLOR 1
180 SI=1.1
190 SF=0.5
200 RF=10
210 CX=80:CY=48
220 XR=159:XL=0:YB=95:YT=0
230 RESTORE 360
240 READ N:DIM X(N),Y(N),X2(N),Y2(N)
250 FOR X=1 TO N:READ W1,W2:X(X)=W1*SF
:Y(X)=W2*SF:NEXT X
260 RW=RW+RF:IF RW>360 THEN RW=RW-360:
GOTO 32767
270 IF RW<0 THEN RW=RW+360
280 FOR X=1 TO N:X(X)=X(X)*SI:Y(X)=Y(X
)*SI:NEXT X
290 FOR X=1 TO N:X2(X)=X(X)*COS(RW)+Y(
X)*SIN(RW)
300 Y2(X)=-X(X)*SIN(RW)+Y(X)*COS(RW):N
EXT X
310 GRAPHICS 6+16
320 FOR X=1 TO N:X2(X)=X2(X)+CX:Y2(X)=
Y2(X)+CY:NEXT X
330 FOR X=1 TO N-1:X1=X2(X):Y1=Y2(X):X
2=X2(X+1):Y2=Y2(X+1):GOSUB 1000:NEXT X
340 X1=X2(N):Y1=Y2(N):X2=X2(1):Y2=Y2(1
):GOSUB 1000
350 GOTO 260
360 DATA 4,10,10,10,-10,-10,-10,-10,10
```

•

CHECKSUM DATA
(See pgs. 7-10)

```
100 DATA 274,394,860,852,286,35,223,49
2,578,583,299,36,472,202,715,6301
250 DATA 317,285,346,819,167,988,212,4
90,135,523,722,951,5955
```

Listing 2.

```
1000 REM ****************************
1010 REM * GRAPHICS CLIPPING ROUTINE *
1020 REM *                          *
1030 REM *       BY TOM HUDSON      *
1040 REM ****************************
1050 L1=0:L2=0:R1=0:R2=0:T1=0:T2=0:B1=
0:B2=0
1060 IF X1<XL THEN L1=1:GOTO 1080
1070 IF X1>XR THEN R1=1
1080 IF Y1>YB THEN B1=1:GOTO 1100
1090 IF Y1<YT THEN T1=1
1100 IF X2<XL THEN L2=1:GOTO 1120
1110 IF X2>XR THEN R2=1
1120 IF Y2>YB THEN B2=1:GOTO 1140
1130 IF Y2<YT THEN T2=1
1140 IF L1+L2=2 OR R1+R2=2 OR T1+T2=2
OR B1+B2=2 THEN RETURN
1150 X3=X1:Y3=Y1:X4=X2:Y4=Y2:GOSUB 121
0
1160 L1=L2:R1=R2:T1=T2:B1=B2
1170 X1=XW:Y1=YW:X3=X2:Y3=Y2:X4=X1:Y4=
Y1:GOSUB 1210
1180 IF X1<XL OR X1>XR OR Y1<YT OR Y1>
YB OR XW<XL OR XW>XR OR YW<YT OR YW>YB
 THEN RETURN
1190 PLOT X1,Y1:DRAWTO XW,YW
1200 RETURN
1210 IF L1+T1+B1+R1=0 THEN XW=X3:YW=Y3
:RETURN
1220 IF L1 THEN XW=XL:YW=Y3+(Y4-Y3)*(X
L-X3)/(X4-X3):X3=XW:Y3=YW:IF Y3>=YT AN
D Y3<=YB THEN RETURN
1230 IF R1 THEN XW=XR:YW=Y3+(Y4-Y3)*(X
R-X3)/(X4-X3):X3=XW:Y3=YW:IF Y3>=YT AN
D Y3<=YB THEN RETURN
1240 IF B1 THEN YW=YB:XW=X3+(X4-X3)*(Y
B-Y3)/(Y4-Y3):X3=XW:Y3=YW:IF X3>=XR AN
D X3<=XL THEN RETURN
1250 IF T1 THEN YW=YT:XW=X3+(X4-X3)*(Y
T-Y3)/(Y4-Y3):X3=XW:Y3=YW:IF X3>=XR AN
D X3<=XL THEN RETURN
1260 RETURN
```

•

CHECKSUM DATA
(See pgs. 7-10)

```
1000 DATA 598,934,60,54,602,235,68,814
,6,822,52,814,18,822,608,6507
1150 DATA 879,618,493,947,81,785,160,9
28,947,905,960,791,8494
```

•

3-D GRAPHS MADE FAST & EASY

16K Cassette or Disk

by Tom Hudson

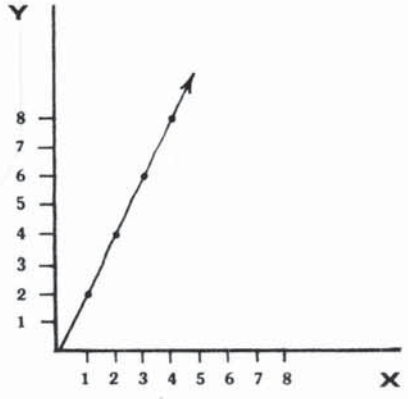
Thanks to ATARI's **Graph-It** (TM) graphics package, ATARI computer owners can generate bar charts, pie graphs, and two- and three-dimensional plots. Unfortunately, when more complex three-dimensional plots are desired, **Graph-It** can take more than an hour to complete just one plot!

In order to assist those **Graph-It** users who would like to see a quick rendition of their 3-D plot before committing themselves to a marathon wait with **Graph-It**, I have written a 3-D graph program which is easy to use and produces graphs very quickly.

By now, many readers are probably asking, "What in the world is a 3-D graph?" which is not a bad question at this point, and one I will try to answer.

We are all familiar with 2-dimensional (flat) graphs. They are usually called "line" or "bar" graphs. **Figure 1** is a line graph of the equation Y=2*X. When X is four, Y is two times four, or eight, and so on.

Figure 1.



In a 3-dimensional graph, things are a little more complicated. As the name implies, we are trying to generate a 3-dimensional form, derived from an equation. To do this, we need three coordinates. We will label these coordinates X (width), Y (depth) and Z (height).

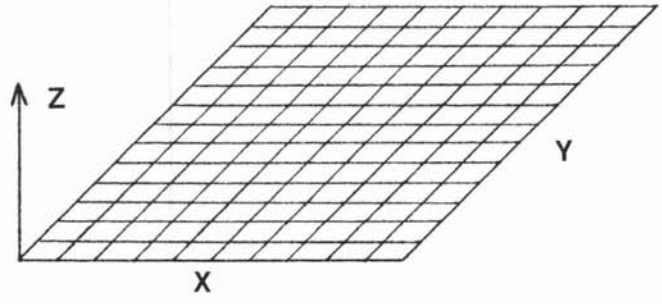


Figure 2.

We start with a grid marked with X and Y coordinates, then we lay this grid flat as in **Figure 2** (a good way to visualize this is to lay a piece of graph paper on a table in front of you). Next we use an equation to determine the Z coordinate. The Z value tells how high off the table each point on the grid is. The Z coordinate is always derived from the X and Y values. In this way, we can see how changes in the X and Y values affect the Z value. For example, in the equation Z=(X+Y)*3, when we are at the coordinates X=1 and Y=3, Z would equal 12 (4 times 3). On our graph, this would be represented as a small peak (**Figure 3**), telling us that where X=1 and Y=3, the Z value is 12. Of course, to be useful this process must be repeated for each point on the grid so that we can see the overall results. Three-dimensional graphs are useful for visualizing how an equation will act with varying X and Y values.

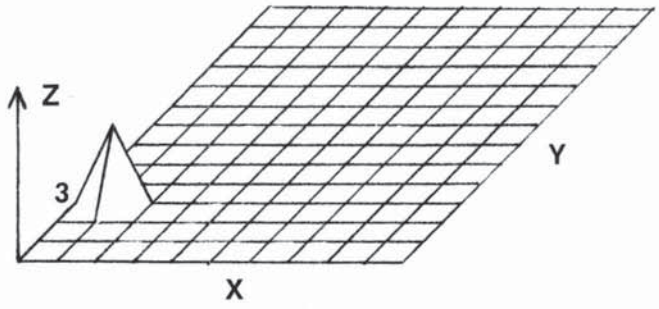


Figure 3

Listing 1 is a simple but effective 3-dimensional graph generation program. It is NOT meant as a replacement for the **Graph-It** 3-D plot program, but an enhancement.

Type **Listing 1** into your computer and check it for accuracy. When the program is correctly entered, you will be ready to start graphing in three dimensions!

Let's say you want a 3-D plot of a complex equation. You don't know what it will look like, but you'd like to get some idea before you wait an hour for **Graph-It** to process it. With this program, you can "preview" a 3-D graph, and if you want a detailed copy, the same equation can be processed by **Graph-It**. An equation requiring 70 minutes on **Graph-It** can be processed by this program in five. Of course, the **Graph-It** version is much smoother and will do such things as automatic scaling, but if you need the output quickly, this program can do it.

Let's find out what the equation $Z=(X-Y)^2$ looks like. Line 220 is where all equations must be executed, so change line 220 to read:

```
220 Z=(X-Y)^2
```

When the line is changed, RUN the program. The screen will go black for several seconds while the computer calculates the plot coordinates of the graph. When these calculations are finished, the screen will come back on, and the graph will be drawn. It's that simple.

How it works.

Line 80 — Set up the arrays needed to store the plot points.

Line 120 — Turn off the system's Direct Memory Access (DMA). This speeds up calculations considerably. The only unpleasant side effect is that the screen goes black until DMA is turned on again.

Line 160 — Set the screen limits for the graphics clipping routine (see lines 600-720).

Lines 210-230 — This is a FOR-NEXT loop for calculating the Z value for each point on the grid. Line 220 is where your equation should be placed. Just replace the existing equation with your own, starting with Z=. The program will do the rest.

Lines 270-300 — After all the Z values have been calculated, this section changes them to plot coordinates so that they can be placed on the screen.

Line 340 — This line turns DMA on again, so that we can see the graph.

Line 380 — This line draws the "zero reference" outline. This is simply the outline of the grid before the Z coordinates were calculated. It lets you know where zero is, relative to the rest of the points on the grid.

Lines 420-430 — This section actually draws the grid on the screen using the data in the GX and GY arrays, which were built in lines 270-300. It uses the graphics clipping routine in lines 600-720 just in case the lines run off the top or bottom of the screen.

Lines 470-500 — These lines draw the vertical lines from the baseline to the corners of the graph.

Line 540 — This line loops the program forever. Hit the break key to stop the program.

Line 600-720 — This is a modified graphics clipping routine. (See *A Graphics Clipping Routine*, page 44). It is modified to only clip lines that extend beyond the top and bottom of the screen, not the sides.

You can try any equation you like in line 220, just set Z to the result. Included below are a few interesting equations for you to try, along with the time required to generate the graphs. Simply replace line 220 with one of these equations. □

```
220 Z=SIN((X+Y-4)/4*30)+30
(Requires approx. 30 seconds)

220 Z=SQR(ABS(X-10.5)^2+ABS(Y-5.5)^2)^
1.7
(Requires approx. 2.25 minutes)

220 Z=(1/SQR(SQR(ABS(X-10.5)^2+ABS(Y-5
.5)^2)+4)^2)*800-50
(Requires approx. 2.75 minutes)

220 Z=70-SQR(ABS(X-10.5)^2+ABS(Y-5.5)^
2)^1.7
(Requires approx. 2.3 minutes)

220 Z=(SIN(X/10)+COS(Y/5))*30
(Requires approx. 40 seconds)

220 Z=(SIN(X/10)*SIN(Y/5))*30
(requires approx. 40 seconds)
```

```
10 REM ********************************
20 REM *      3-D GRAPH PROGRAM       *
30 REM *                              *
40 REM *         BY TOM HUDSON        *
50 REM ********************************
60 REM
```

```
70 GRAPHICS 24:SETCOLOR 2,0,0:COLOR 1
80 DIM GX(21,11),GY(21,11)
90 REM
100 REM *** DMA OFF ***
110 REM
120 POKE 559,0
130 REM
140 REM *** SET CLIPPING LIMITS ***
150 REM
160 XR=319:XL=0:YT=0:YB=191
170 REM
180 REM *** YOUR FORMULA GOES ***
190 REM *** INSIDE THIS LOOP  ***
200 REM
210 FOR X=1 TO 21:FOR Y=1 TO 11
220 Z=(X+Y)*3
230 GY(X,Y)=Z:NEXT Y:NEXT X
240 REM
250 REM *** CALC. SCREEN COORDS. ***
260 REM
270 FOR X=1 TO 21:FOR Y=1 TO 11
280 GX(X,Y)=(X-1)*10+(Y-1)*10
290 GY(X,Y)=180-(Y-1)*10-GY(X,Y)
300 NEXT Y:NEXT X
310 REM
320 REM *** DMA ON AGAIN ***
330 REM
340 POKE 559,34
350 REM
360 REM *** DRAW BASELINE ***
370 REM
380 PLOT 0,180:DRAWTO 200,180:DRAWTO 3
00,80:DRAWTO 100,80:DRAWTO 0,180
390 REM
400 REM *** PLOT THE GRAPH ***
410 REM
420 FOR X=1 TO 21:FOR Y=2 TO 11:X1=GX(
X,Y-1):Y1=GY(X,Y-1):X2=GX(X,Y):Y2=GY(X
,Y):GOSUB 600:NEXT Y:NEXT X
430 FOR Y=1 TO 11:FOR X=2 TO 21:X1=GX(
X-1,Y):Y1=GY(X-1,Y):X2=GX(X,Y):Y2=GY(X
,Y):GOSUB 600:NEXT X:NEXT Y
440 REM
450 REM *** DRAW VERTICAL LINES ***
460 REM
470 X1=0:Y1=180:X2=GX(1,1):Y2=GY(1,1):
GOSUB 600
480 X1=200:Y1=180:X2=GX(21,1):Y2=GY(21
,1):GOSUB 600
490 X1=300:Y1=80:X2=GX(21,11):Y2=GY(21
,11):GOSUB 600
500 X1=100:Y1=80:X2=GX(1,11):Y2=GY(1,1
1):GOSUB 600
510 REM
520 REM *** LOOP FOREVER ***
530 REM
540 GOTO 540
550 REM
560 REM ****************************
570 REM *  GRAPHICS CLIP ROUTINE  *
580 REM ****************************
590 REM
600 T1=0:T2=0:B1=0:B2=0:IF Y1<YT THEN
T1=1:GOTO 620
610 IF Y1>YB THEN B1=1
620 IF Y2<YT THEN T2=1:GOTO 640
630 IF Y2>YB THEN B2=1
640 IF T1+T2=2 OR B1+B2=2 THEN RETURN
650 X3=X1:Y3=Y1:X4=X2:Y4=Y2:GOSUB 690
660 T1=T2:B1=B2:X1=XW:Y1=YW:X3=X2:Y3=Y
2:X4=X1:Y4=Y1:GOSUB 690
670 IF Y1<YT OR Y1>YB OR YW<YT OR YW>Y
B THEN RETURN
680 PLOT X1,Y1:DRAWTO XW,YW:RETURN
690 IF T1+B1=0 THEN XW=X3:YW=Y3:RETURN
700 IF T1 THEN YW=YT:XW=X3+(X4-X3)*(YT
-Y3)/(Y4-Y3):X3=XW:Y3=YW:RETURN
710 IF B1 THEN YW=YB:XW=X3+(X4-X3)*(YB
-Y3)/(Y4-Y3):X3=XW:Y3=YW:RETURN
720 RETURN
```

•

CHECKSUM DATA
(See pgs. 7-10)

```
10 DATA 587,293,21,12,595,261,273,936,
267,649,77,777,83,421,89,5341
160 DATA 657,95,338,245,76,34,990,837,
88,515,94,52,420,841,520,5802
310 DATA 81,562,87,3,93,546,99,883,105
,830,83,323,366,92,349,4502
460 DATA 98,734,303,480,855,85,726,91,
723,97,368,494,374,109,536,6073
610 DATA 188,353,198,823,482,869,545,4
8,298,732,573,599,5708
```

•

Sphere Demo

```
8 SIZE=90:REM ***RADIUS***
9 CX=160:CY=96:REM **CENTER**
10 DEG :TIME=1
20 GRAPHICS 24:SETCOLOR 2,0,8:SETCOLOR
 1,0,0:COLOR 1
25 PLOT CX+SIZE,CY:REM ***START***
30 FOR Y=90 TO 0 STEP -12
40 FOR X=0 TO 360 STEP 12
50 IF TIME=1 THEN X2=CX+SIZE*COS(X):Y2
=CY-(SIZE*SIN(X)*SIN(Y)):GOTO 60
55 X2=CX-(SIZE*SIN(X)*SIN(Y)):Y2=CY+SI
ZE*COS(X)
60 DRAWTO X2,Y2:NEXT X:NEXT Y
90 TIME=TIME+1:IF TIME=2 THEN PLOT CX,
CY+SIZE:GOTO 30
100 SIZE=20+RND(1)*30:CX=SIZE+1+(RND(1
)*(318-(SIZE*2))):CY=SIZE+1+(RND(1)*(1
90-(SIZE*2))):GOSUB 1000:TIME=1:GOTO 2
5
910 REM *** ERASE HIDDEN LINES ***
1000 COLOR 0:FOR X=0 TO 90 STEP 0.5
1010 X2=SIZE*COS(X):Y2=SIZE*SIN(X)
1020 PLOT CX+X2,CY+Y2:DRAWTO CX-X2,CY+
Y2:PLOT CX+X2,CY-Y2:DRAWTO CX-X2,CY-Y2
:NEXT X:COLOR 1:RETURN
```

CHECKSUM DATA
(See pgs. 7-10)

```
8 DATA 365,712,880,195,686,399,350,614
,380,298,205,106,520,297,673,6680
1020 DATA 202,202
```

GRAPHIC VIOLENCE

16K Cassette or Disk

by Tom Hudson

When writing game programs, many programmers automatically choose assembly language over BASIC because of the obvious speed advantage. This can sometimes be a mistake, since BASIC offers some functions (such as sine, square root, etc.) not easily written in assembler. One way to take advantage of the convenience of BASIC and the speed of assembler is to combine the two languages. ATARI BASIC allows the user to "call" machine-language subroutines, which can be many times faster than the same routine in BASIC.

In order to assist those game programmers who would like to have dramatic explosion effects in their BASIC programs, I have developed **Graphic Violence**, a group of assembly-language subroutines. These routines allow BASIC to generate up to 20 simultaneous explosions in GRAPHICS 7. They can optionally generate sound effects as well as "cycle" the colors of the explosions for an interesting "radioactive glow" effect.

The first half of this article is a non-technical explanation of how to use **Graphic Violence**. The second half is an in-depth discussion of the actual assembly language code for those interested in the inner workings of the subroutines.

Using Graphic Violence.

Listing 1 is the BASIC language code necessary to set up the **Graphic Violence** subroutine. This code should be placed in any program that is to use the explosion generator. After typing this program in, SAVE it immediately, BEFORE RUNNING IT! The routine has some safeguards against typing errors in the DATA statements, but if it is executed with bad DATA, the system may crash and it will be necessary to re-type the program.

After the program is typed and SAVEd, RUN it. If it is typed correctly, the program will run for several seconds before anything happens. The screen colors will begin cycling quickly. If not, an error was made somewhere, and you should re-boot your system, load the SAVEd program, find the mistake, SAVE it and try again.

If a message such as "COORDI ERR" occurs, you have made a mistake typing in the DATA statements. "COORDI ERR" indicates that an error was made in the COORDI DATA, "INIT ERR" is an error in the INITIALIZATION CODE, etc. Find the error, fix it and re-RUN the program.

Once the computer starts cycling colors, press SYSTEM RESET before doing anything else. Whenever operating any program using the Graphic Violence subroutine, you MUST use the SYSTEM RESET key to terminate the program. The subroutine automatically disables the BREAK key since typing commands in immediate mode while the subroutine is in operation will usually cause a system crash. Pressing SYSTEM RESET will correctly terminate the subroutine and avoid any problems.

At this point, you should have a correctly operating **Graphic Violence** initialization subroutine SAVEd on tape or disk.

Program 1 Flow.

Line 80 — GOSUBs to line 10000 to initialize the subroutine.

Line 10010 — Dimensions the strings needed by Graphic Violence and RESTOREs the DATA pointer.

Line 10020-10060 — READs DATA statements into the strings used by the subroutine.

Line 10080 — POKEs graphics PLOT values into Graphic Violence.

Line 101000 — Calls the machine-language initialization routine. It is of the form:

```
A=USR(ADR(INIT$),ADR(MAIN$),ADR(COORD1
$),ADR(COORD2$),COLOR,SOUND)
```

The COLOR value tells whether or not you want the color of the explosions to cycle. In the program listing, this value is set to 1, indicating that cycling is desired. If you do not want cycling, place a 0 here.

The SOUND value tells whether or not you want the routine to generate sounds with the explosions. In the listing it is a 1, indicating that we want sound. If sound is not desired, place a 0 here.

Line 10110 — This line simply returns from the subroutine to the main program

A short demonstration.

With **Listing 1** in your computer, add **Listing 2** to the original program and RUN it. This is a short demonstration routine which simply places an

explosion at the center of the screen, then repeats.

By looking at this short routine, you will notice the USR call in line 220. This is the command which starts an explosion. Once the Graphic Violence machine-code subroutine is set up, this short operation is all you need to generate explosions.

Remember to stop the program by pressing SYSTEM RESET.

Program 2 Flow.

Line 190 — Set up a full-screen graphics mode 7.

Line 220 — Call the explosion-starting machine language routine. This line actually starts the explosion. It is of the form:

```
A=USR(ADR(EXPL$),X,Y)
```

X and Y are the screen coordinates of the center of the explosion. In the **Listing**, X=80 and Y=48, placing the explosion at the center of the screen.

This statement is the heart of the Graphic Violence routine. Once this statement is executed, it starts off an explosion while BASIC continues with whatever it is doing. In addition, the explosion handler can operate up to 20 explosions simultaneously, while BASIC does its own processing!

Line 240 — This line is a simple delay loop which allows an explosion to dissipate before generating another.

Line 260 — This line goes to start a new explosion after the wait.

In the previous example, we generated one explosion at the center of the screen, just to keep things simple. In the next example, we will see how the Graphic Violence routine will handle up to 20 simultaneous explosions without the programmer having to worry about what's going on inside the explosion handler! All the programmer needs to do is send the explosion coordinates to the routine via the USR command and let the computer do the rest. (What could be simpler?)

With **Listing 1** in your computer, add **Listing 3** to the original program and RUN it. The program will fill up most of the screen with graphics, then start dropping "bombs" from the top of the screen. As they hit the graphics area, they will explode violently, "eating" away the graphics. As soon as one of the bombs falls off the bottom of the screen, an end message will be displayed and subsequently destroyed by a number of explosions. The program will run continuously and MUST be stopped by pressing SYSTEM RESET.

Program 3 Flow.

Line 190 — Sets up graphics mode 7 and sets COLOR #2 (the explosion color) to maximum brightness.

Line 210 — Fills up the bottom section of the screen with COLOR 1 graphics.

Line 230 — Makes sure any error will cause the program to continue at line 320 (the "THE END" routine). This TRAP statement will take effect when a bomb falls off the bottom of the screen.

Line 250 — Gets the X and Y coordinates where the bomb will start its drop.

Line 270 — Erases old bomb position (using COLOR 0) and increments Y position so that bomb will "fall" toward bottom of screen.

Line 290 — Uses the LOCATE command to see if the bomb has hit anything. If the bomb hits color 1, an explosion is started at the X and Y coordinates and a new bomb is randomized.

Line 310 — If no hit is detected, the bomb is plotted in color 2, the program waits a fraction of a second, then continues at line 270.

Line 330 — When a bomb falls off the bottom of the screen, the error is TRAPped here. At this time, the computer sets up a new graphics 7 screen, sets the explosion brightness, and selects COLOR 1.

Line 350 — This line RESTOREs the DATA pointer to line 400 (THE END shape data), reads from-and-to plot data and draws the THE END message on the screen.

Line 370 — This line sets off 200 explosions, which destroy the THE END message. Note that the explosion USR call has random number functions for X and Y coordinates of the explosion center. There is also a 40 count delay after each explosion is started for a more interesting display.

Line 390 — After all explosions are generated, wait a few seconds and GOTO line 190 to re-run the demonstration continuously.

Line 410-430 — These lines contain PLOT data for the words "THE END." Each line in the letters is represented by 4 values, made up of 2 sets of X and Y coordinates, the line endpoints.

Summary.

The **Graphic Violence** explosion generator subroutine will operate in almost any game using graphics 7. Explosions overlapping the edges of the screen are automatically "clipped," but the program has minimal error-trapping. The user should take care to make sure that the coordinates supplied to the routine do not exceed the graphics 7 screen limits. The routine uses sound channel 1 when the sound generation option is requested. The Explosions use COLOR 3 (SETCOLOR 2), and will cycle the color only (not brightness) if color cycling is requested. Any program using the **Graphic Violence** routine must be terminated with SYSTEM RESET to avoid a system crash.

The following section contains a discussion of the assembly-language routines that make up **Graphic Violence**. This information is not necessary to use

the subroutine, but may assist those interested in assembly language and the inner workings of the ATARI computers.

Background information.

The Graphic Violence subroutine is made up of three program segments and two data tables. These five modules work together to provide a machine-language explosion generator for BASIC.

The first assembly program (**Listing 4**) is the Graphic Violence initialization subroutine. It is stored in the BASIC string variable INIT$. Its function is to accept the locations of the main program module, and accept the color cycling and sound generation options.

Remember that this is the routine called in the BASIC statement:

```
A=USR(ADR(INIT$),ADR(MAIN$),ADR(COORD1
$),ADR(COORD2$),COLOR,SOUND)
```

Program 4 Flow.

Line 230 — This line arbitrarily sets the location counter to $6000. Since this routine will be fully relocatable and stored in a BASIC string, this address does not matter.

Line 240 — This PLA instruction pulls the first argument off of the stack. In a BASIC USR call, this argument is always the number of arguments passed to the machine language routine. We do not use it in this case, and it is discarded.

Line 250-270 — This section zeroes out the explosion ready flag and the explosion counter.

Line 280-330 — This section pulls the low and high bytes of the address of the main routine (ADR MAIN$), transfers them to the X and Y registers, then puts a 7 in the accumulator and jumps to the SETVBV subroutine. This tells the system that we are using a vertical blank interrupt. The 7 indicates that it is a "deferred" vertical blank routine, that is, it operates after the system's vertical blank operation.

Line 340-410 — This section pulls the low and high bytes of the two sets of plot coordinates (COORD1$ and COORD2$, 4 PLA s total) and stores them on page zero ($CB-$CE) for later use by the main module.

Line 420-440 — This section pulls the color cycle indicator (COLOR) from the stack. Since this is a one-byte indicator and the system sends a two-byte argument, the first byte (high byte) is discarded and the second is stored in CYCFLG.

Line 450-470 — This section is the same as lines 420-440, except that it stores the sound indicator (SOUND) in SNDFLG.

Line 480 — This RTS (Return from Subroutine) returns control to your BASIC program after the initialization is complete.

The second assembly language program (**Listing 5**) is the explosion start routine. It is called by the BASIC statement:

```
A=USR(ADR(EXPL$),X,Y)
```

This routine simply accepts the coordinates of the explosion from BASIC. If there are 20 explosions active, it will ignore the request, otherwise it will send the coordinates to the main module, which is executing in the deferred vertical blank.

Program 5 Flow.

Line 200 — Once again, this **Listing** has its location counter set to $6000. It makes no difference, since this routine is fully relocatable.

Line 210 — As in the previous **Listings**, this line discards the first item on the stack (the number of arguments passed to the assembly routine).

Line 220-240 — These lines check the variable EXPCNT to make sure the new explosion can be started. If there are less than 20, control is passed to EXPOK (explosion OK).

Line 250-290 — These lines are used if there are already 20 explosions. The remaining 4 bytes are pulled from the stack and discarded, and the program returns to BASIC. No explosion is generated.

Line 300-350 — In a manner similar to the COLOR and SOUND parameters in **Listing #4**, this routine pulls the X and Y coordinates off of the stack and places the values in NEWX and NEWY for use by the main module.

Line 360-370 — This section places a 1 in READY flag, which tells the main interrupt routine that a new explosion is ready to start.

Line 380 — This RTS instruction simply returns control to BASIC. In this way, the interrupt can start the explosion graphics while BASIC keeps running normally.

The third assembly language routine (**Listing 6**) is the vertical blank interrupt routine, stored in MAIN$. It does all the color cycling, sound, and graphics for the explosions. Since it is an interrupt-driven program, it operates independently of BASIC, allowing BASIC to continue processing normally while the vertical blank does all the explosion work.

Since this program is stored in a BASIC string, any program editing or immediate mode operations in BASIC while the vertical blank routine is running will cause a system crash. This is due to the fact that BASIC moves its variables around in memory during editing of programs, and such movement of the interrupt routine will confuse the system. To help avoid such a problem, the **Graphic Violence**

interrupt routine disables the break key, making it necessary to press SYSTEM RESET to stop program execution. This is only a partial solution, however, since if the programmer allows his program to end with the READY prompt, then enters a program line, the crash will still occur.

The interrupt routine performs several functions. First, it disables the BREAK key and cycles the color of playfield 2 if necessary. Next, it processes sound, if required, using sound channel 1. The last major function it performs is that of explosion graphics generation.

Each explosion graphic is made up of 89 separate pixels. The routine uses the specified centerpoint of each explosion and adds X and Y offset values, which are stored in the BASIC string variables COORD1$ and COORD2$. Each of the 89 pixels are first turned on, one pixel at a time, resulting in a "growing" appearance. After all 89 pixels are on, the routine turns off one pixel at a time, causing the explosion to dissipate. Each active explosion has a pixel either turned on or off each time the interrupt is performed. Since this happens 60 times a second, each explosion takes roughly 3 seconds to expand and dissipate [(89*2)/60]. Explosions are independent of each other because of three tables. The X and Y coordinates of each explosion are stored in the XPOS and YPOS tables. The third table, CNT, holds the number of the pixel which will be turned on or off next for each explosion. This value ranges from 0 to 88 for "on" pixels, and 89 to 177 for "off" pixels. If the CNT value for an explosion exceeds 177, the explosion has dissipated completely and its values are removed from the explosion tables by a "repack" operation. That is, if explosion number 2 is finished, explosion 3 will move back to 2, 4 to 3, etc.

Program 6 Flow.

Line 500 — Clears decimal mode. This instruction is vital when writing subroutines for BASIC that do any binary arithmetic.

Line 510-540 — Disables the BREAK key by altering POKMSK and IRQEN, the interrupt request enable. This prevents the BREAK key from generating an interrupt.

Line 550-640 — Cycles colors if CYCFLG is not zero.

Line 650-770 — Processes explosion sound if SNDFLG is not zero.

Line 780-940 — Monitors the READY flag to see if there is a new explosion. If not, the program checks for any old explosions at MAIN. If there is a new explosion, the routine sets up the XPOS, YPOS and CNT tables with the new information.

Line 950 — Zeroes out COUNTR, the variable indicating which explosion is being processed.

Line 960-1000 — Increments the explosion counter. If the counter is greater than the current number of explosions active (EXPCNT), the routine jumps to XITVBV, the vertical blank exit vector. Otherwise control is passed to INDEX.

Line 1130-1350 — This section repacks the XPOS, YPOS and CNT tables to eliminate a "dead" explosion. It then branches back to RUNLP to handle the next explosion.

Line 1360-2350 — This routine turns explosion pixels on or off, depending on the PLOTCLR setting. If the pixel is off the screen, the plot is abandoned by a branch to RUNLP.

By expanding the XPOS, YPOS and CNT tables and altering the explosion call routine (**Listing 5**), advanced users can enable the **Graphic Violence** routine to handle many more explosions than it can now. However, 20 explosions are more than enough for most applications, and the routine should serve well as is.

I hope that ATARI programmers will see by this example that it is not always necessary to write game programs completely in assembly language. Just use BASIC for complicated functions difficult to write in assembler, and use assembler for things BASIC is too slow to do. □

Listing 1 (BASIC)

```
10 REM ****************************
20 REM * GRAPHIC VIOLENCE DEMO  *
30 REM * A.N.A.L.O.G. COMPUTING *
40 REM * BY TOM HUDSON          *
50 REM ****************************
60 REM
70 REM *** INITIALIZE THE GRAPHIC VIOL
ENCE SUBROUTINE ***
80 GOSUB 10010
90 REM
100 REM ******************************
110 REM ** YOUR PROGRAM GOES HERE! **
120 REM ******************************
130 GOTO 130
10000 REM *** INITIALIZATION SUBROUTIN
E ***
10010 DIM INIT$(41),EXPL$(29),MAIN$(35
5),COORD1$(89),COORD2$(89):RESTORE 110
00
10020 TOT=0:FOR X=1 TO 89:READ A:TOT=T
OT+A:COORD1$(X,X)=CHR$(A):NEXT X:IF TO
T<>9984 THEN ? "COORD1 ERR":END
10030 TOT=0:FOR X=1 TO 89:READ A:TOT=T
OT+A:COORD2$(X,X)=CHR$(A):NEXT X:IF TO
T<>9984 THEN ? "COORD2 ERR":END
10040 TOT=0:FOR X=1 TO 41:READ A:TOT=T
OT+A:INIT$(X,X)=CHR$(A):NEXT X:IF TOT<
>4237 THEN ? "INIT ERR":END
10050 TOT=0:FOR X=1 TO 29:READ A:TOT=T
OT+A:EXPL$(X,X)=CHR$(A):NEXT X:IF TOT<
>2198 THEN ? "EXPL ERR":END
10060 TOT=0:FOR X=1 TO 355:READ A:TOT=
TOT+A:MAIN$(X,X)=CHR$(A):NEXT X:IF TOT
<>36691 THEN ? "MAIN ERR":END
10070 REM *** SET UP PLOT BITS ***
10080 POKE 1568,192:POKE 1569,48:POKE
1570,12:POKE 1571,3
10090 REM *** INITIALIZE GRAPHIC VIOLE
NCE ROUTINE AND RETURN ***
10100 A=USR(ADR(INIT$),ADR(MAIN$),ADR(
COORD1$),ADR(COORD2$),1,1)
10110 RETURN
```

```
11000 REM *** COORD1 DATA ***
11010 DATA 0,1,255,0,255,0,255,2,1,1,0
,254,255,1,0,1,254,254,2,0,1,255,2,2,2
,255,254,1,253,3,3,4,252,253,254
11020 DATA 255,254,2,3,3,253,0,0,0,4,4
,252,255,2,0,3,2,1,253,254,254,252,253
,3,253,252,251,251,252,4,3,4,255
11030 DATA 5,5,5,253,1,254,0,255,252,2
53,251,253,252,3,4,3,1,255,1,2,4
12000 REM *** COORD2 DATA ***
12010 DATA 0,255,1,2,254,255,0,1,254,0
,1,0,255,1,253,253,2,255,255,254,2,3,2
,0,254,2,1,3,254,1,254,255,0,1,253
12020 DATA 253,254,3,2,0,3,252,4,3,0,2
,2,4,4,5,3,253,252,0,3,4,254,252,252,2
,1,1,0,255,254,255,1,251
12030 DATA 0,255,1,4,4,252,251,252,253
,253,255,255,3,253,253,4,251,5,5,252,3
13000 REM *** INITIALIZATION CODE ***
13010 DATA 104,169,0,141,0,6,141,1,6,1
04,170,104,168,169,7
13020 DATA 32,92,228,104,133,204,104,1
33,203,104,133,206,104,133,205
13030 DATA 104,104,141,11,6,104,104,14
1,12,6,96
14000 REM *** EXPLOSION CALL CODE ***
14010 DATA 104,173,1,6,201,20,48,5,104
,104,104,104,96,104,104
14020 DATA 141,2,6,104,104,141,3,6,169
,1,141,0,6,96
14990 REM *** MAIN INTERRUPT CODE ***
15000 DATA 216,165,16,41,127,133,16,14
1,14,210,173,11,6,240,20
15010 DATA 173,14,6,24,105,16,141,14,6
,173,198,2,41,15,13
15020 DATA 14,6,141,198,2,173,12,6,240
,22,173,13,6,240,17
15030 DATA 56,233,1,141,13,6,74,74,74,
141,1,210,169,40,141
15040 DATA 0,210,173,0,6,240,31,238,1,
6,174,1,6,173,2
15050 DATA 6,157,64,6,173,3,6,157,85,6
,169,127,141,13,6
15060 DATA 169,0,157,106,6,141,0,6,141
,5,6,238,5,6,173
15070 DATA 1,6,205,5,6,16,3,76,98,228,
174,5,6,169,0
15080 DATA 141,4,6,189,106,6,201,89,48
,51,238,4,6,56,233
15090 DATA 89,201,89,48,41,138,168,232
,236,1,6,240,2,16,21
15100 DATA 189,64,6,153,64,6,189,85,6,
153,85,6,189,106,6
15110 DATA 153,106,6,200,208,227,206,1
,6,206,5,6,169,0,240
15120 DATA 176,254,106,6,168,189,64,6,
24,113,203,141,6,6,201
15130 DATA 160,176,159,189,85,6,24,113
,205,141,7,6,201,96,176
15140 DATA 146,10,133,207,169,0,240,2,
240,137,133,208,165,207,10
15150 DATA 133,207,165,208,42,133,208,
165,207,10,133,207,141,9,6
15160 DATA 165,208,42,133,208,141,8,6,
165,207,10,133,207,165,208
15170 DATA 42,133,208,165,207,10,133,2
07,165,208,42,133,208,165,207
15180 DATA 24,109,9,6,133,207,165,208,
109,8,6,133,208,165,88
15190 DATA 24,101,207,133,207,165,89,1
01,208,133,208,173,6,6,41
15200 DATA 3,168,190,32,6,142,10,6,173
,6,6,74,74,24,101
15210 DATA 207,133,207,165,208,105,0,1
33,208,160,0,173,4,6,208
15220 DATA 11,173,10,6,81,207,145,207,
169,0,240,132,173,10,6
15230 DATA 73,255,49,207,145,207,169,0
,240,241
```

•

CHECKSUM DATA

(See pgs. 7-10)

```
10 DATA 280,324,225,872,288,261,725,83
7,267,778,948,784,701,830,321,8441
10020 DATA 807,814,290,322,416,442,758
,706,200,43,332,901,920,385,338,7674
12010 DATA 966,265,36,858,239,907,884,
831,543,392,825,377,13,7,217,7360
15040 DATA 450,996,743,441,863,301,958
,239,326,614,853,887,899,169,370,9109
15190 DATA 615,960,409,269,122,2375
```

•

Listing 2.

```
130 REM ****************************
140 REM *  GRAPHIC VIOLENCE DEMO  *
150 REM *         NUMBER 1         *
160 REM ****************************
170 REM
180 REM *** SET UP GRAPHIC MODE 7 ***
190 GRAPHICS 7+16
200 REM *** SET OFF AN EXPLOSION ***
210 REM *** AT SCREEN CENTER      ***
220 A=USR(ADR(EXPL$),80,48)
230 REM *** WAIT A FEW SECONDS ***
240 FOR WAIT=1 TO 2000:NEXT WAIT
250 REM *** DO EXPLOSION AGAIN ***
260 GOTO 220
```

•

Listing 3.

```
130 REM ****************************
140 REM *  GRAPHIC VIOLENCE DEMO  *
150 REM *         NUMBER 2         *
160 REM ****************************
170 REM
180 REM *** SET UP GRAPHICS 7 FULL SCR
EEN AND EXPLOSION COLOR ***
190 GRAPHICS 7+16:SETCOLOR 2,15,15
200 REM *** DRAW THE 'GROUND' ***
210 COLOR 1:FOR Y=20 TO 95:PLOT 0,Y:DR
AWTO 159,Y:NEXT Y
220 REM *** TRAP ANY ERRORS TO 'THE EN
D' ROUTINE ***
230 TRAP 320
240 REM *** RANDOMIZE START POINT FOR
DROPPING BOMBS ***
250 X=5+RND(0)*149:Y=RND(0)*3
260 REM *** ADVANCE THE BOMB AS IT DRO
PS ***
270 COLOR 0:PLOT X,Y:Y=Y+3
280 REM *** IF THE BOMB HITS COLOR 1,
SET OFF EXPLOSION ***
290 LOCATE X,Y,Z:IF Z=1 THEN A=USR(ADR
(EXPL$),X,Y):GOTO 250
300 REM *** NO HIT, CONTINUE DROP ***
310 COLOR 2:PLOT X,Y:FOR DELAY=1 TO 10
:NEXT DELAY:GOTO 270
320 REM *** 'THE END' ***
330 GRAPHICS 7+16:SETCOLOR 2,15,15:COL
OR 1
340 REM *** PLOT 'THE END' ***
350 RESTORE 400:FOR X=1 TO 22:READ FRX
,FRY,TUX,TUY:PLOT FRX,FRY:DRAWTO TUX,T
UY:NEXT X
360 REM *** SET OFF 200 RANDOM EXPLOSI
ONS ***
370 FOR EXPL=1 TO 200:A=USR(ADR(EXPL$)
,40+RND(0)*75,20+RND(0)*55):FOR DELAY=
1 TO 40:NEXT DELAY:NEXT EXPL
380 REM *** LET EXPLOSIONS DIE, THEN R
E-RUN THE DEMO ***
390 FOR DELAY=1 TO 2000:NEXT DELAY:GOT
O 190
400 REM *** 'THE END' DATA ***
410 DATA 50,25,67,25,59,25,59,45,72,25
,72,45,72,35,88,35,88,25,88,45,93,25,9
3,45,93,25,109,25,93,35,109,35
```

```
420 DATA 93,45,109,45,50,50,50,70,50,5
0,67,50,50,60,67,60,50,70,67,70,72,70,
72,50,72,50,88,70,88,70,88,50
430 DATA 93,50,93,70,93,50,102,50,102,
50,109,56,109,56,109,64,109,64,102,70,
102,70,93,70
```

•

CHECKSUM DATA

(See pgs. 7-10)

```
130 DATA 351,454,438,360,95,403,617,10
0,539,885,711,340,552,331,470,6646
280 DATA 421,589,835,842,98,463,638,15
3,787,999,122,753,603,401,961,8665
430 DATA 292,292
```

•

Listing 4.

```
0100 ; GRAPHIC VIOLENCE
0110 ;
0120 ; A.N.A.L.O.G. COMPUTING
0130 ;
0140 ; INITIALIZATION CODE
0150 ;
0160 READY = $600
0170 EXPCNT = $601
0180 CYCFLG = $60B
0190 SNDFLG = $60C
0200 COORD1 = $CB
0210 COORD2 = $CD
0220 SETVBV = $E45C
0230  *= $6000
0240 INIT PLA          ;DISCARD
0250  LDA #0           ;ZERO OUT:
0260  STA READY        ;READY FLAG
0270  STA EXPCNT       ;# OF EXPL.
0280  PLA              ;INTERRUPT HI
0290  TAX              ;PUT IN X
0300  PLA              ;INTERRUPT LO
0310  TAY              ;PUT IN Y
0320  LDA #7           ;DEFERRED VBI
0330  JSR SETVBV       ;SET IT!
0340  PLA              ;COORD1 HI
0350  STA COORD1+1     ;SAVE IT
0360  PLA              ;PULL COORD1 LO
0370  STA COORD1       ;SAVE IT
0380  PLA              ;PULL COORD2 HI
0390  STA COORD2+1     ;SAVE IT
0400  PLA              ;PULL COORD2 LO
0410  STA COORD2       ;SAVE IT
0420  PLA              ;DISCARD
0430  PLA              ;PULL COLOR CYCLE FLAG
0440  STA CYCFLG       ;SAVE IT
0450  PLA              ;DISCARD
0460  PLA              ;PULL SOUND FLG
0470  STA SNDFLG       ;SAVE IT
0480  RTS              ;FINISHED!
0490  .END
```

•

Listing 5.

```
0100 ; GRAPHIC VIOLENCE
0110 ;
0120 ; A.N.A.L.O.G. COMPUTING #8
0130 ;
0140 ; EXPLOSION CALL ROUTINE
0150 ;
0160 READY = $600
0170 EXPCNT = $601
0180 NEWX = $602
0190 NEWY = $603
0200  *=$6000
0210  PLA              ;DISCARD
0220  LDA EXPCNT       ;# OF EXPL.
0230  CMP #20          ;20 ACTIVE?
0240  BMI EXPOK        ;NO, IT'S OK!
0250  PLA              ;YES, DISCARD
0260  PLA              ;BOTH COORDS
0270  PLA
0280  PLA
0290  RTS              ;AND EXIT
0300 EXPOK PLA         ;DISCARD HIGH
0310  PLA              ;GET X-COORD
0320  STA NEWX         ;STORE IT
0330  PLA              ;DISCARD HIGH
0340  PLA              ;GET Y-COORD
0350  STA NEWY         ;STORE IT
0360  LDA #1           ;TELL INTERRUPT
0370  STA READY        ;WE'RE READY!
0380  RTS              ;AND EXIT BACK
0390 ;                  TO BASIC!
0400  .END
```

•

Listing 6.

```
0100 ; GRAPHIC VIOLENCE
0110 ;
0120 ; A.N.A.L.O.G. COMPUTING
0130 ;
0140 ; VBLANK INTERRUPT ROUTINE
0150 ;
0160 READY = $600
0170 EXPCNT = $601
0180 NEWX = $602
0190 NEWY = $603
0200 PLOTCLR = $604
0210 COUNTR = $605
0220 PLOTX = $606
0230 PLOTY = $607
0240 HIHLD = $608
0250 LOHLD = $609
0260 PLOTBYT = $60A
0270 CYCFLG = $60B
0280 SNDFLG = $60C
0290 SNDCNT = $60D
0300 COLOR = $60E
0310 PLOTBL = $620
0320 XPOS = $640
0330 YPOS = XPOS+21
0340 CNT = YPOS+21
0350 LO = $CF
0360 HI = $D0
0370 COORD1 = $CB
0380 COORD2 = $CD
0390 ;
0400 ;SYSTEM EQUATES
0410 ;
0420 XITVBV = $E462
0430 COLPF2 = $2C6
0440 AUDC1 = $D201
0450 AUDF1 = $D200
0460 SAVMSC = $58
0470 POKMSK = $10
0480 IRQEN = $D20E
0490  *=$6000
0500  CLD              ;CLEAR DECIMAL
0510  LDA POKMSK       ;GET IRQ INT.
0520  AND #$7F         ;NO BREAK KEY
0530  STA POKMSK       ;THE BREAK KEY
0540  STA IRQEN        ;IS NOW OFF!
0550  LDA CYCFLG       ;CYCLING COLOR?
0560  BEQ CONT         ;NO, CONTINUE
```

```
0570  LDA COLOR       ;GET LAST COLOR
0580  CLC             ;INCREMENT IT
0590  ADC #16         ;BY 16
0600  STA COLOR       ;AND SAVE IT
0610  LDA COLPF2      ;GET COLOR REG.
0620  AND #$0F        ;GET BRIGHTNESS
0630  ORA COLOR       ;ADD THE COLOR
0640  STA COLPF2      ;AND SAVE IT!
0650 CONT LDA SNDFLG  ;SOUND ON?
0660  BEQ GO          ;NO, SKIP IT!
0670  LDA SNDCNT      ;MORE SOUND?
0680  BEQ GO          ;NO, SKIP IT!
0690  SEC             ;DECREMENT THE
0700  SBC #1          ;SOUND COUNTER
0710  STA SNDCNT      ;AND STORE IT
0720  LSR A           ;SHIFT DOWN TO
0730  LSR A           ;DERIVE VOLUME
0740  LSR A           ;FROM COUNTER
0750  STA AUDC1       ;SET UP SOUND
0760  LDA #40         ;CHANNEL 1...
0770  STA AUDF1       ;FINISHED!
0780 GO LDA READY     ;NEW EXPLOSION?
0790  BEQ MAIN        ;NO, CONTINUE
0800 ;
0810 ;AT THIS POINT, THERE IS A
0820 ;NEW EXPLOSION!
0830 ;
0840  INC EXPCNT      ;ONE MORE EXPL
0850  LDX EXPCNT      ;PUT IN INDEX
0860  LDA NEWX        ;GET X-COORD,
0870  STA XPOS,X      ;PUT IN TABLE
0880  LDA NEWY        ;GET Y-COORD,
0890  STA YPOS,X      ;PUT IN TABLE
0900  LDA #127        ;INITIALIZE THE
0910  STA SNDCNT      ;SOUND COUNTER
0920  LDA #0          ;INIT COUNTER
0930  STA CNT,X       ;FOR EXPL IMAGE
0940  STA READY       ;AND READY FLAG
0950 MAIN STA COUNTR  ;ZERO COUNTER
0960 RUNLP INC COUNTR ;NEXT EXPLOSION
0970  LDA EXPCNT      ;GET # OF EXPL.
0980  CMP COUNTR      ;ANY MORE EXPL?
0990  BPL INDEX       ;YES, CONTINUE
1000  JMP XITVBV
1010 INDEX LDX COUNTR ;GET INDEX
1020  LDA #0          ;SET PLOTCLR
1030  STA PLOTCLR     ;0=PLOT A BLOCK
1040  LDA CNT,X       ;GET COUNTER
1050 ;                 FOR EXPLOSION
1060  CMP #89         ;ALL DRAWN?
1070  BMI DOPLOT      ;NO, DO IT NOW
1080  INC PLOTCLR     ;1=ERASE BLOCK
1090  SEC             ;GET READY FOR
1100  SBC #89         ;ERASE CYCLE
1110  CMP #89         ;ERASE DONE?
1120  BMI DOPLOT      ;NO,ERASE BLOCK
1130  TXA             ;MOVE INDEX
1140  TAY             ;TO Y REGISTER
1150 ;
1160 ;THE FOLLOWING ROUTINE REPACKS
1170 ;THE EXPLOSION TABLE TO GET RID
1180 ;OF EXPLOSIONS THAT ARE DONE.
1190 ;
1200 REPACK INX       ;NEXT EXPLOSION
1210  CPX EXPCNT      ;DONE?
1220  BEQ RPK2        ;NO,REPACK MORE
1230  BPL RPKEND      ;YES, EXIT!
1240 RPK2 LDA XPOS,X  ;NO, START RPK
1250  STA XPOS,Y      ;MOVE BACK X
1260  LDA YPOS,X
1270  STA YPOS,Y      ;MOVE BACK Y
1280  LDA CNT,X
1290  STA CNT,Y       ;MOVE BACK CNT
1300  INY
1310  BNE REPACK      ;NEXT REPACK
1320 RPKEND DEC EXPCNT ;DEC POINTERS
1330  DEC COUNTR      ;DUE TO REPACK
1340  LDA #0          ;FORCE BRANCH
1350  BEQ RUNLP       ;TO NEXT EXPL.
1360 DOPLOT INC CNT,X ;INC COUNTER
1370  TAY             ;EXP PHASE IN Y
1380  LDA XPOS,X      ;GET X-COORD
1390  CLC
1400  ADC (COORD1),Y  ;ADD X OFFSET
1410  STA PLOTX       ;STORE IT
1420  CMP #160        ;OFF SCREEN?
1430  BCS RUNLP       ;YES,DON'T PLOT
1440  LDA YPOS,X      ;GET Y-COORD
1450  CLC
1460  ADC (COORD2),Y  ;ADD Y OFFSET
1470  STA PLOTY       ;STORE IT
1480  CMP #96         ;OFF SCREEN?
1490  BCS RUNLP       ;YES,DON'T PLOT
1500 ;
1510 ;THE FOLLOWING SECTION IS A
1520 ;DEDICATED MULTIPLY ROUTINE
1530 ;WHICH MULTIPLIES THE A REGISTER
1540 ;BY 40, WITH RESULT IN LO & HI
1550 ;
1560  ASL A
1570  STA LO
1580  LDA #0
1590  BEQ X2
1600 JRUNLP BEQ RUNLP
1610 X2 STA HI        ;*2
1620  LDA LO
1630  ASL A
1640  STA LO
1650  LDA HI
1660  ROL A
1670  STA HI         ;*4
1680  LDA LO
1690  ASL A
1700  STA LO
1710  STA LOHLD
1720  LDA HI
1730  ROL A
1740  STA HI
1750  STA HIHLD      ;*8
1760  LDA LO
1770  ASL A
1780  STA LO
1790  LDA HI
1800  ROL A
1810  STA HI         ;*16
1820  LDA LO
1830  ASL A
1840  STA LO
1850  LDA HI
1860  ROL A
1870  STA HI         ;*32
1880  LDA LO
1890  CLC
1900  ADC LOHLD
1910  STA LO
1920  LDA HI
1930  ADC HIHLD
1940  STA HI         ;+*8=*40
1950 ;
1960 ;AT THIS POINT, THE MULTIPLY BY
1970 ;40 IS FINISHED, AND WE NEED TO
1980 ;GET AN OFFSET INTO THE SCREEN
1990 ;MEMORY
2000 ;
2010  LDA SAVMSC     ;ADD THE DISPLAY
2020  CLC             ;ADDRESS TO GET
2030  ADC LO          ;THE ACTUAL
2040  STA LO          ;ADDRESS OF THE
2050  LDA SAVMSC+1    ;BYTE THAT WILL
2060  ADC HI          ;BE ALTERED FOR
2070  STA HI          ;THE PLOT.
2080  LDA PLOTX       ;MASK PLOTX FOR
2090  AND #3          ;THE PLOT BITS,
2100  TAY             ;PLACE IN Y...
```

```
2110  LDX PLOTBL,Y    ;GET PLOT BITS,
2120  STx PLOTBYT     ;AND SAVE'
2130  LDA PLOTX       ;GET PLOTX AND
2140  LSR A           ;DIVIDE
2150  LSR A           ;BY 4
2160  CLC             ;AND ADD TO
2170  ADC LO          ;PLOT ADDRESS
2180  STA LO          ;FOR FINAL PLOT
2190  LDA HI          ;ADDRESS.
2200  ADC #0
2210  STA HI
2220  LDY #0          ;ZERO OUT Y REG.
2230  LDA PLOTCLR     ;ERASING?
2240  BNE CLEARIT     ;YES,GO CLEAR IT
2250  LDA PLOTBYT     ;GET PLOT BITS,
2260  EOR (LO),Y      ;ALTER DISPLAY,
2270  STA (LO),Y      ;AND PLOT IT!
2280  LDA #0          ;FORCE BRANCH
2290 JRUNLP2 BEQ JRUNLP ;AND EXIT!
2300 CLEARIT LDA PLOTBYT ;PLOT BITS
2310  EOR #$FF        ;FLIP 'EM
2320  AND (LO),Y      ;ALTER DISPLAY
2330  STA (LO),Y      ;AND ERASE IT!
2340  LDA #0          ;FORCE BRANCH
2350  BEQ JRUNLP2     ;AND EXIT!
2360  .END
```

Graphics 11 GTIA Demo

```
10 REM GRAPHICS 11 GTIA DEMO
20 REM
30 GRAPHICS 11
40 CI=1:C=0:SETCOLOR 4,0,2
50 FOR Y=0 TO 191
60 FOR X=0 TO 79
70 C=C+1:IF C=16 THEN C=0
80 COLOR C
90 PLOT X,Y
100 NEXT X
110 LC=LC+1:IF LC=16 THEN CI=-CI:LC=1
120 C=C+CI:IF C=16 THEN C=0
130 NEXT Y
140 GOTO 140
```

CHECKSUM DATA

(See pgs. 7-10)

```
10 DATA 990,253,995,374,128,296,319,77
0,619,758,988,438,769,707,8404
```

ATARI 1020 PRINTER DEMO

```
10 REM ************************
20 REM *  ATARI 1020 PLOTTER  *
30 REM * SPHERE DEMONSTRATION *
40 REM *     BY TOM HUDSON     *
50 REM ************************
60 REM
70 REM *** OPEN IOCB 1 TO PLOTTER ***
80 REM
90 OPEN #1,8,0,"P:"
100 REM
110 REM *** SET SPHERE RADIUS ***
120 REM
130 SIZE=150
140 REM
150 REM *** INITIALIZE PLOTTER ***
160 REM
170 ? #1;"E\*H*I*M0,";-SIZE-20;"*I"
180 REM
190 REM *** SET SPHERE CENTER ***
200 REM
210 CX=240:CY=0
220 REM
230 REM *** START PLOTTING! ***
240 REM
250 DEG :TIME=1
260 ? #1;"M";CX+SIZE;",";CY:REM *** ST
ART THE PLOT ***
270 FOR Y=90 TO 0 STEP -12
280 FOR X=0 TO 360 STEP 12
290 IF TIME=1 THEN X2=CX+SIZE*COS(X):Y
2=CY-(SIZE*SIN(X)*SIN(Y)):GOTO 340
300 X2=CX-(SIZE*SIN(X)*SIN(Y)):Y2=CY+S
IZE*COS(X)
310 REM
320 REM *** DRAW LINE OF SPHERE ***
330 REM
340 ? #1;"D";X2;",";Y2
350 NEXT X:NEXT Y
360 REM
370 REM *** DO NEXT DIRECTION ***
380 REM
390 TIME=TIME+1:IF TIME=2 THEN ? #1;"M
";CX;",";CY+SIZE:GOTO 270
400 REM
410 REM *** MOVE PAPER UP AT END ***
420 REM
430 ? #1;"H*M0,";-SIZE-20;"*I"
440 CLOSE #1:END
```

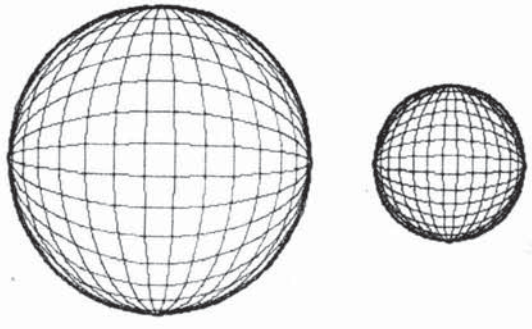
CHECKSUM DATA
(See pgs. 7-10)

```
10 DATA 267,70,613,909,275,261,719,265
,505,74,225,80,748,86,609,5706
160 DATA 92,755,98,248,76,28,82,288,88
,84,840,182,274,562,515,4212
310 DATA 81,293,87,768,536,96,322,102,
57,80,395,86,576,131,3610
```

```
10 REM **********************
20 REM * ATARI 1020 PLOTTER *
30 REM * "SQUARE-WEB" DEMO  *
40 REM *   BY TOM HUDSON    *
50 REM **********************
60 REM
70 REM *** OPEN IOCB 1 TO PLOTTER ***
80 REM
90 OPEN #1,8,0,"P:"
100 REM
110 REM *** INITIALIZE PLOTTER ***
120 REM
130 ? #1;"E\*H*I*M0,-400*I"
140 REM
150 REM *** START PLOT LOOP ***
160 REM
170 FOR X=20 TO 380 STEP 20
180 REM
190 REM *** DRAW 4 LINES ***
200 REM
210 ? #1;"M";X;",380*D380,";400-X;";";
400-X;",20;20,";X;";";X;",380"
220 NEXT X
230 REM
240 REM *** ALL DONE! ***
250 REM
260 ? #1;"H"
270 CLOSE #1:END
```

CHECKSUM DATA
(See pgs. 7-10)

```
10 DATA 831,647,628,440,839,261,719,26
5,505,74,597,80,923,86,212,7107
160 DATA 92,302,98,591,76,229,766,85,1
75,91,499,136,3140
```



DISK UTILITIES

DISK FILES: USING NOTE & POINT

32K Disk

by Jerry White

This is a demonstration program that creates a 100 record inventory file and permits the user to update the file using random access. Random access allows immediate access to any given record in a file without reading each record again and again.

The rest of this article assumes you have typed in the program. If there were no errors in typing, you should now have 3 options on the screen. First, we must create a data file, so type 1. This will send the program to line 100. We will now create a 100 record file to work with. Each record will contain a record number, an item count and an item description field. Each field is separated by a comma. As the file is created, it will be displayed on the screen. After record 100 is written the file is closed, and you will be returned to the 3 original options.

Now type the number 2. In order to use random access updating, we must know exactly where each record begins on the diskette. Before updating, the program will create an index using an array in memory. Once this is done, we can instantly find any record using the POINT instruction. But first, we must NOTE the location by storing the sector number and byte in our arrays. We only have to do this once. Then we can inspect or change as many records as needed.

The index is created using the routine starting at line 300 and ending at 420. Line 500 is the beginning of the random access routine. You should be able to follow the program listing, since the variables used will all be defined at the end of this article. At this time, I will only explain how the NOTE and POINT instructions are used.

At line 310, we check a flag to see if an index has already been created. If so, we do not have to repeat this procedure and go to line 500. To create the index, we read the datafile. Before reading each record, we NOTE the sector and byte position and put it into our SEC and BYT arrays.

Once the array is complete, we are ready to display or update any record in the datafile using the POINT instruction to locate the record we want. Let's start by displaying record 50. Type D and press RETURN. Then type 50 and press RETURN. At line 760, we POINT to the sector and byte of record 50. At line 780, we INPUT that record, clear the screen, and display record 50 on the screen.

Remember the number of items in this record, press any key, and you will be returned to the option routine at line 5000. Type 2 and this time we will change record 50. Type U and press RETURN, then type 50 and press RETURN. Record 50 will again be displayed, but now we have 3 new options. Let's take them in order. Type 1, then press RETURN. To update the quantity, we merely add to it by typing the number of items to add, or subtract by typing a negative number. Remember that our quantity field is only 3 positions, so don't increase it to more than 999 items.

After reading the record from the datafile, we store it in a string call REC$. The quantity field is updated in the string. It will be updated on the disk only when we choose option 3 to exit. Before we exit, let's change the description field. Type 2 and press RETURN. Choose a new description and type it. Now type 3, and the record will be updated on the disk.

To be sure that the record has been changed properly, you can choose the display/update option then redisplay record 50. By now, you can see the advantages of random access updating. You don't have to read the first 49 records to get to record 50. Once the arrays of the sectors and bytes contain the beginning of every record, we can locate any record instantly. □

```
20 REM INVENTORY TUTORIAL PROGRAM TO D
EMONSTRATE RANDOM ACCESS UPDATING
30 REM *** BY JERRY WHITE ***
50 DIM SEC(100),BYT(100),REC$(30),DES$
(30),CHOICE$(1):CI=0:GOTO 5000
100 REM *** CREATE INITIAL DATA FILE *
**
110 FOR BLANK=1 TO 30:REC$(BLANK,BLANK
)=" ":NEXT BLANK
120 CLOSE #1:OPEN #1,8,0,"D:DATAFILE"
130 REC$(4,4)=",":REC$(8,30)=",ITEM DE
SCRIPTION FIELD"
140 FOR RECORD=1 TO 100
160 IF RECORD<10 THEN REC$(1,2)="00":R
EC$(3,3)=STR$(RECORD):GOTO 220
```

```
180 IF RECORD<100 THEN REC$(1,1)="0":R
EC$(2,3)=STR$(RECORD):GOTO 220
200 REC$(1,3)=STR$(RECORD)
220 REC$(5,7)=STR$(RND(0)*100+100)
240 PRINT #1;REC$:? :? "RECORD ";RECOR
D:? REC$:NEXT RECORD
260 CLOSE #1:GOTO 5000
300 REM *** CREATE INDEX ***
310 IF CI=1 THEN RECORD=101:GOTO 500
320 TRAP 2000:CLOSE #2:OPEN #2,4,0,"D:
DATAFILE":TRAP 40000
360 FOR ARRAY=1 TO 100:NOTE #2,SECTOR,
BYTE
380 ? :? "RECORD ";ARRAY;"  SECTOR ";S
ECTOR;"  BYTE ";BYTE
400 SEC(ARRAY)=SECTOR:BYT(ARRAY)=BYTE:
INPUT #2,REC$:NEXT ARRAY
420 CLOSE #2:CLOSE #3:CI=1
500 REM *** RANDOM ACCESS DATAFILE ***
520 CLOSE #4:OPEN #4,12,0,"D:DATAFILE"
540 ? CHR$(125):? :? "TYPE D TO DISPLA
Y A RECORD":? :? "TYPE U TO UPDATE A R
ECORD";
560 INPUT CHOICE$:IF CHOICE$="D" THEN
700
580 IF CHOICE$="U" THEN 900
600 ? CHR$(253):GOTO 540
700 ? :? "TYPE RECORD NUMBER TO DISPLA
Y";:TRAP 700:INPUT RN:TRAP 40000
720 IF RN<ARRAY AND RN>0 AND RN=INT(RN
) THEN 760
740 ? CHR$(253):? "INVALID RECORD NUMB
ER":GOTO 700
760 POINT #4,SEC(RN),BYT(RN)
780 INPUT #4,REC$:? CHR$(125):? :? "RE
CORD ";RN:? :? REC$
800 ? :? "PRESS ANY KEY FOR OPTIONS:":
POKE 764,255:CLOSE #4
820 IF PEEK(764)<>255 OR PEEK(53279)<>
7 THEN POKE 764,255:GOTO 5000
840 GOTO 820
900 ? :? "TYPE RECORD NUMBER TO BE UPD
ATED";:TRAP 900:INPUT RN:TRAP 40000
920 IF RN<ARRAY AND RN>0 AND RN=INT(RN
) THEN 960
940 ? CHR$(253):? "INVALID RECORD NUMB
ER":GOTO 900
960 POINT #4,SEC(RN),BYT(RN)
980 INPUT #4,REC$:? CHR$(125)
1000 ? :? "RECORD ";RN:? :? REC$
1010 ? :? "TYPE 1 TO UPDATE QUANTITY":
? "TYPE 2 TO CHANGE DESCRIPTION":? "TY
PE 3 TO EXIT"
1020 TRAP 1000:INPUT CHOICE:TRAP 40000
1040 IF CHOICE<1 OR CHOICE>3 OR CHOICE
<>INT(CHOICE) THEN ? CHR$(253):GOTO 10
00
1060 ON CHOICE GOTO 1100,1300,1080
1080 POINT #4,SEC(RN),BYT(RN):PRINT #4
;REC$:CLOSE #4:GOTO 5000
1100 ? :? "TYPE POSITIVE NUMBER TO INC
REASE ITEMS":? "TYPE NEGATIVE NUMBER T
O DECREASE ITEMS"
1140 TRAP 1100:INPUT NUMBER:TRAP 40000
1160 ITEMS=VAL(REC$(5,7)):ITEMS=ITEMS+
NUMBER
1180 IF ITEMS>999 THEN ? CHR$(253):? "
ITEMS CANNOT EXCEED 999":GOTO 1100
1200 IF ITEMS<0 THEN ? CHR$(253):? "IT
EMS CANNOT BE A LESS THAN ZERO":GOTO 1
120
1220 IF ITEMS<10 THEN REC$(5,6)="00":R
EC$(7,7)=STR$(ITEMS):GOTO 1000
1240 IF ITEMS<100 THEN REC$(5,5)="0":R
EC$(6,7)=STR$(ITEMS):GOTO 1000
1260 REC$(5,7)=STR$(ITEMS):GOTO 1000
1300 ? CHR$(125):? :? "RECORD ";RN:? :
? REC$
1320 ? :? "TYPE NEW DESCRIPTION UP TO
22 POSITIONS"
1340 INPUT DES$:LD=LEN(DES$)
1360 IF LD>22 THEN ? CHR$(253):? "FIEL
D TOO LONG, EXTRA IGNORED"
1380 IF LD=22 THEN 1420
1400 FOR BLANK=LD TO 22:DES$(LEN(DES$)
+1)=" ":NEXT BLANK
1420 REC$(9,30)=DES$:GOTO 1000
2000 ? CHR$(253):? :? "DATAFILE NOT ON
 DISK:TRAP 40000"
2010 FOR WAIT=1 TO 500:NEXT WAIT:GOTO
5000
5000 REM *** INITIAL DISPLAY OF OPTION
S ***
5010 GRAPHICS 18:? #6:? #6;" INVENTORY
 OPTIONS:":? #6:? #6;"  1= CREATE FILE
"
5020 ? #6:? #6;"  2= DISPLAY/UPDATE":?
 #6:? #6;"  3= END PROGRAM"
5040 CLOSE #5:OPEN #5,4,0,"K:":GET #5,
GC:CLOSE #5:GC=GC-48
5060 IF GC<1 OR GC>3 THEN 5000
5080 GRAPHICS 0:POKE 82,1:SETCOLOR 2,0
,0:ON GC GOTO 100,300,6000
6000 GRAPHICS 0:POKE 82,2:END
```

•

CHECKSUM DATA

(See pgs. 7-10)

```
20 DATA 161,467,887,260,998,773,777,82
0,204,222,294,135,704,49,680,7431
310 DATA 667,231,617,590,580,241,701,6
58,461,208,528,24,420,307,305,6538
760 DATA 887,785,571,427,729,704,315,3
11,891,490,709,443,866,682,198,9008
1080 DATA 743,395,936,860,817,197,63,8
5,456,987,634,949,21,624,90,7857
1420 DATA 626,794,547,747,299,492,995,
631,886,917,6934
```

•

DISK DIRECTORY DUMP

16K Disk

by Tony Messina

This utility is rather simple in nature, but can prove quite helpful when trying to remember what program is on which diskette. In order for this utility to work, you need the following items: 1) a disk drive, 2) a printer (40 or 80 column), 3) an ATARI computer with at least 16K of memory. The utility itself will give you a neat, formatted hardcopy of your disk directory (I told you it was simple!). The following article should also give you a general idea about IOCBs and the OPEN/CLOSE statements which are part of the BASIC repertoire.

IOCBs.

Many programs appearing in this book use OPEN and CLOSE statements to perform a particular function. I'm sure such questions as "What is being opened/closed," "How/Why is it being opened/closed," and "How can I open/close my own things?" have crossed your mind, so now would be a good time to find out what it's all about!!

One of the most difficult things to do on any computer is INPUT/OUTPUT, or I/O for short. Would you like to write the program (commonly called a driver) to print to the printer or list to the disk or input a character from the keyboard? It really isn't all that fun. Thanks to those great ATARI folks who designed our systems (the operating system in particular), we don't have to worry too much about the above-mentioned items. We can control out I/O through an IOCB or Input/Output Control Block.

The operating system has eight IOCBs. Each IOCB contains information as to the nature of the device we want to communicate with, where the driver for the device is located, where the buffer for the device is located, the length of the buffer, the command we are trying to execute on the device (OPNE, CLOSE, PUT CHARACTER, GET CHARACTER, etc.), timeout values (i.e., how long do we try to execute a command before we decide to give up), etc. This information is used by the Central Input/Output (CIO) portion of the operating system when communicating with the device on the IOCB specified.

Now that we know something about IOCBs, let's look at how we set them up.

OPEN and CLOSE.

The OPEN command allows us to communicate with a device using the CIO facility. We don't have to know machine language to access a device...we can use BASIC instead! OPEN just dedicates an IOCB to perform our command. We can think of it as opening a hotline to our device. The line will stay open until we hang up or CLOSE it. The form of the OPEN command is as follows:

OPEN #IOCB,I/O CODE, SPECIAL, DEVICE

Parameters can take on the following values:

IOCB — Any number from 0-7. Usually only 1-5 is best, since the operating system uses IOCB 0 for the screen/editor, 6 for any graphics window (I'm sure you all have used a PRINT #6 statement), and 7 for LPRINT and Cassette I/O.

I/O CODE — 4=INPUT, 8=OUTPUT, 12=INPUT and OUTPUT, 6=DISK DIRECTORY INPUT and 9=OUTPUT (APPEND TO END OF FILE).

SPECIAL — Is usually 0 but can be filled in based on the device you are using. If you are opening a screen mode other than GR.0, you would need to put the GR. mode number in the SPECIAL parameter. If you have a sideways printing printer (say that 10 times quickly), you could get it to print sideways by putting 83 as the SPECIAL parameter. When in doubt, use 0.

DEVICE — Devices which we can control and which BASIC knows about are the KEYBOARD "K:", GRAPHICS WINDOW "S:", PRINTER "P:", CASSETTE "C:", DISK FILE "D:filename.ext", SCREEN EDITOR "E:" and RS232 PORTS "R:".

When opening a device, we must make sure that the parameters make sense. We wouldn't want to open a printer for INPUT and OUTPUT, since most printers only allow OUTPUT. It also wouldn't make sense to open the graphics window for DISK DIRECTORY INPUT. See...it's not all that complicated.

Once we have opened a device, there are many things which can be done. Commands such as PUT #, GET #, PRINT #, etc. can be executed by BASIC directly to the device we have opened. The only thing we have to remember is not to use an invalid command for the I/O CODE selected. If we opened the GRAPHICS WINDOW for OUTPUT, for example, then we could not use the GET command. Experiment using OPEN with its associated commands and you'll soon become proficient in the mysterious world of ATARI I/O.

How DDD uses IOCBs.

This utility opens 2 IOCBs. IOCB 1 is opened for output to the printer in LINE 220, and IOCB 2 is opened for disk directory input in LINE 230. The filename to get has been set to "D:**", since we want to see all of the files. DEV$ is simply set to "P:" for the printer. I also set all my codes to constants for easier reading. The values can be found in LINES 115-125.

With these two IOCBs open, the rest of the utility is a snap. We input a file name in LINE 380 and output it to the printer. A nice thing about the directory input command is that it also returns the number of FREE SECTORS after the last filename has been input. LINE 385 checks for this and routes us to LINE 420 when we are done. Another item to note is that the printer now recognizes ";" and "," so that we can format our output. LPRINT, under certain circumstances, will recognize these two characters, but it's best to open a channel to the printer and do a PRINT # instead.

The remainder of the utility performs error checks and issues prompts for the user. All of the major sections have been block commented and should present no major problems when you try and figure out what is being done.

How to use it.

Type in the listing and save it to your disk. You can now RUN the program. If you forget to turn on your printer or disk, you will be razzed until you do. Just follow the prompts and you'll soon have a listing of all your directories. You can even print a title (18 characters max) for each of your directories to help jog your memory.

One last note. If your printer doesn't support the expanded print mode, then you must change LINE 305 by deleting the ESC/ESC/CNTL N sequence and also deleting the "*2)" from the centering calculation. If you have an EPSON printer, just change the code for expanded print to the appropriate code. □

```
10 REM ******************************
15 REM *        UTILITY  #4           *
20 REM *      DISKCAT VER.1           *
25 REM *      BY TONY MESSINA         *
30 REM * FOR A.N.A.L.O.G. COMPUTING *
35 REM ******************************
40 REM *
45 REM ***********************
50 REM * MAKE SCREEN TITLE *
55 REM ***********************
60 REM *
65 GRAPHICS 2:START=PEEK(560)+PEEK(561
)*256:POKE START+9,6:POKE START+10,6:P
OKE START+11,5
70 POKE 712,32:POKE 711,10
75 ? #6;"      Utility # 4":? #6:? #6;"
    DisK Dir DuMP":? #6;" ============
======":? #6;"              BY"
80 ? #6;"          tony messina":? #6;"   Co
pyright  A.N.A.L.O.G  Computing":? #6;
"       1983"
85 ? #6;" ================="
90 REM *
95 REM ******************
100 REM * VARIABLE INIT *
105 REM ******************
110 REM *
115 DIM DEV$(2):DIM TAB$(40):DIM DIREC
TORY$(5):DIM FILENAME$(19):DIM AN$(1)
120 DIRECTORY$="D:*.*":TAB$="
                            "
125 DISK=2:PRNTER=1:DIRTAB=10:COLWID=4
0:OUTPUT=8:NULL=0:DIRIN=6:COUNT=3:SPAC
E=3
130 REM *
135 REM ********************
140 REM * GET USER INPUT *
145 REM ********************
150 REM *
155 ? "R"
160 DEV$="P:":TRAP 505:? "COLUMN WIDTH
 (40 OR 80) ";:INPUT WIDTH
165 IF WIDTH<>40 AND WIDTH<>80 THEN GO
TO 160
170 IF WIDTH=80 THEN COLWID=WIDTH:GOTO
 215
175 DIRTAB=1:COUNT=2:SPACE=2
180 ?
185 REM *
190 REM **********************
195 REM * OPEN DEVICES FOR *
200 REM *   INPUT/OUTPUT   *
205 REM **********************
210 REM *
215 TRAP 495:LPRINT
220 OPEN #PRNTER,OUTPUT,NULL,DEV$
225 TRAP 500
230 OPEN #DISK,DIRIN,NULL,DIRECTORY$
235 REM *
240 REM ************************
245 REM * ASK FOR HEADER NAME *
250 REM ************************
255 REM *
260 ? "ENTER DISK TITLE ";:INPUT FILEN
AME$
265 IF FILENAME$="" THEN FILENAME$="-D
EFAULT NAME-"
270 REM *
275 REM **********************
280 REM * PRINT TITLE OUT *
285 REM **********************
290 REM *
295 IF LEN(FILENAME$)>18 THEN GOTO 510
300 TRAP 510
305 PRINT #PRNTER;TAB$(1,INT((COLWID-(
LEN(FILENAME$)*2))/2));"E_";FILENAME$
310 ? #PRNTER:? #PRNTER;TAB$(1,DIRTAB)
;
315 REM *
320 REM **********************
325 REM * PRINT COLUMN ID *
330 REM **********************
335 REM *
340 FOR HEADCNT=1 TO COUNT:? #PRNTER;"
  FILNAME/EXT LEN";TAB$(1,SPACE);:NEXT
 HEADCNT:? #PRNTER
345 ? #PRNTER:? #PRNTER;TAB$(1,DIRTAB)
;
```

```
350 REM *
355 REM ***************************
360 REM * GET FILENAMES AND PRINT *
365 REM ***************************
370 REM *
375 FOR X=1 TO COUNT
380 INPUT #DISK,FILENAME$
385 IF LEN(FILENAME$)<17 THEN ? #PRNTE
R:? #PRNTER;TAB$(1,((COLWID-16)/2)-1);
FILENAME$:GOTO 420
390 ? #PRNTER;FILENAME$;TAB$(1,SPACE);
:NEXT X:GOTO 345
395 REM *
400 REM ************************
405 REM * CK IF USER WANTS MORE *
410 REM ************************
415 REM *
420 CLOSE #DISK:CLOSE #PRNTER
425 ? "DO ANOTHER Y/N ";:INPUT AN$
430 IF AN$<>"Y" AND AN$<>"N" THEN GO
TO 420
435 IF AN$="N" THEN ? "DIRDMP DONE!":
GOTO 460
440 ? "USE SAME PARAMETERS (Y/N) ";:IN
PUT AN$
445 IF AN$<>"Y" AND AN$<>"N" THEN GO
TO 440
450 IF AN$="Y" THEN GOSUB 490:GOTO 22
0
455 GOSUB 490:RUN
460 END
465 REM *
470 REM ************************
475 REM *  ERROR TRAPS FOLLOW  *
480 REM ************************
485 REM *
490 ? "INSERT NEW DISK AND HIT <RETUR
N> ";:INPUT AN$:RETURN
495 ? "PRINTER DOES NOT RESPOND!!":G
OTO 160
500 ? "DISK DOES NOT RESPOND!!":CLOS
E #PRNTER:GOTO 160
505 ? "INPUT ERROR (ONLY NUMBERS PLEA
SE) ":GOTO 160
510 ? "NAME TOO LONG!!":? "MAX LENGT
H IS 18":GOTO 260
```

•

CHECKSUM DATA
(See pgs. 7-10)

```
10 DATA 771,6,123,386,973,790,73,36,95
7,38,77,571,96,789,414,6100
85 DATA 396,83,555,565,775,274,620,478
,181,280,600,328,603,286,382,6486
160 DATA 891,741,931,154,909,300,49,94
0,821,29,276,732,876,711,271,8631
235 DATA 287,288,220,291,293,179,615,2
94,811,693,814,300,828,705,370,6988
310 DATA 166,283,793,563,796,289,603,1
80,290,366,580,369,296,778,660,7012
385 DATA 23,135,307,293,438,296,285,96
5,318,322,55,17,334,135,534,4457
460 DATA 50,300,117,84,120,306,488,404
,898,979,580,4326
```

•

Rainbow Demo

```
5 SETCOLOR 2,0,0:POKE 752,1:PRINT CHR$
(125)
10 DIM C$(24)
20 FOR I=1 TO 24
30 READ D
40 C$(I,I)=CHR$(D)
50 NEXT I
60 D=USR(ADR(C$))
70 END
100 DATA 162,0,173,11,212,201,32,208,2
49,141
110 DATA 10,212,142,24,208,232,232,208
,246,142
120 DATA 24,208,240,232
```

CHECKSUM DATA
(See pgs. 7-10)

```
5 DATA 557,836,230,324,306,380,233,255
,645,111,599,4476
```

BURP!

16K Disk

by Charles Bachand

Over the years, we have all run across diskettes that just would not format. This was probably due to a scratch or dent on the disk surface, and even though the ATARI 810 disk drive returns the addresses of bad sectors to the computer (Huh, I didn't know that!), the disk operating system makes no use of them. Well, how would you like to be able to use those disks that up until now you have been feeding to the trash?

BURP (Basic Unusable-disk Reclaimer Program), is a machine language program that patches itself into DOS's disk formatting routine. Being an AUTORUN.SYS file, it is loaded when the computer is first powered on and is essentially transparent to the user. The only programming limitation is that no other program can reside within the address space $600-$694.

There are a couple of limitations involved in using BURP:

1) The program will still return a bad sector error and abort if any bad sector is in the space taken up by the disk directory (sectors 360-368) or the disk boot (sectors 1-4).

2) Do not save DOS out to a disk after BURP has run without also saving a copy of the AUTORUN.SYS file containing BURP to the same disk. The DOS will have been patched into BURP, and without BURP itself loaded into memory any attempt to format a disk will end with DOS dying a terrible death! (In other words, it bytes the dust.)

BURP is divided into three sections. The first is a group of patches that load into the exisiting DOS. These patches wedge BURP into the Disk Operating System, and reduce the error_retry count to allow the OS to say "I give up!" a lot sooner.

Next follows the main section of the program that converts the bad sector numbers returned by the 810 disk drive into the corresponding bits of the disk directory's Volume Table Of Contents (VTOC). The VTOC has a bit for every sector on the disk. If a bit is on (1), this tells DOS that it may store data in that sector; no other file is currently using it. It also follows that if the bit is off (0), the sector is currently in use and should not be touched.

We next compare these bits with those of a freshly formatted disk. If the bit is on then BURP will shut it off to mark it as being in use. However, if it was in use to begin with, then we are in trouble and BURP will produce a bad sector error.

The last part of the program will check the number of sector errors on the disk. If there were no errors encountered, the program merely writes the first directory sector. Otherwise, we build a fake file entry with the name "Bad Sectors" and a length of the number of bad sectors. This entry is used as a flag to identify which of your disks caused problems.

The completely documented Macro Assembler listing follows, as well as a BASIC program to generate BURP.

There are two more limitations of this program that have surfaced. The larger of the two problems is the fact that it will not work with Percom disk drives. This is due to the fact that the Percom drive does not return bad sector numbers to the operating system if it cannot format a disk. The second problem shows up if you try to duplicate a disk using DOS option "J." DOS 2.0S copies all sectors whose corresponding bit in the VTOC is set, it will try to copy the bad sectors which it cannot do. It will instead issue an error message. A way around this would be to copy individually every file on the disk.

To generate the BURP program and have it SAVEd to a file, run the BURP maker program written in BASIC. The object program will be stored on file D:AUTORUN.SYS, which will automatically load the program after DOS is loaded.

If you want the option of using BURP or not using it, simply change the file name specified in the opening statement to something other than D:AUTORUN.SYS. To run BURP now, it will be necessary to call up the DOS Menu and perform a binary load from your chosen new file. □

BASIC Listing.

```
100 REM +++ BURP +++
110 REM BASIC UNUSEABLE-DISK
120 REM RECLAIMER PROGRAM
130 REM
140 OPEN #1,8,0,"D:AUTORUN.SYS"
150 TRAP 170
160 READ A:PUT #1,A:GOTO 160
170 CLOSE #1:END
200 DATA 255,255,140,7,141,7,169,0
210 DATA 76,13,78,13,76,82,13,142
220 DATA 13,144,13,32,0,6,165,13
230 DATA 169,13,208,248,32,101,6,0
240 DATA 6,144,6,169,0,141,148,6
250 DATA 141,147,6,172,147,6,177,71
260 DATA 141,145,6,200,177,71,141,146
270 DATA 6,200,201,255,208,8,205,145
280 DATA 6,208,3,76,149,16,140,147
290 DATA 6,169,0,160,3,78,146,6
300 DATA 110,145,6,106,136,208,246,42
310 DATA 42,42,42,168,169,0,56,106
320 DATA 136,16,252,170,173,145,6,105
330 DATA 10,168,138,49,69,208,5,104
340 DATA 104,76,181,18,138,81,69,145
350 DATA 69,160,3,177,69,56,233,1
360 DATA 145,69,238,148,6,76,8,6
370 DATA 172,148,6,240,21,162,10,189
380 DATA 134,6,9,128,157,6,20,202
390 DATA 16,245,169,96,141,1,20,140
400 DATA 2,20,32,113,16,76,25,18
410 DATA 96,66,97,100,32,83,101,99
420 DATA 116,111,114,115,224,2,225,2
430 DATA 133,6
```

•

CHECKSUM DATA
(See pgs. 7-10)

```
100 DATA 347,105,821,83,280,726,842,13
4,963,782,486,970,524,825,266,8154
270 DATA 5,7,697,221,975,239,813,55,76
1,548,9,748,798,505,21,6402
420 DATA 948,740,1688
```

•

Assembly listing.

```
;       BURP - Bad Disk Reclaimer Program

;       Written by Charles Bachand

;       This program patches itself into
;       ATARI's DOS 2.0S to allow for the
;       formatting of physically damaged
;       and previously unuseable diskettes

;       Note:
;               This program will not allow
;               the formatting of a disk
;               with damaged disk boot sectors,
;               or damaged directory sectors.
;               Sorry.

;       System Equates

VTC     =       $45     ;directory's VTOC pointer
BAD     =       $47     ;bad sector buffer pointer
WRTDIR  =       $1071   ;write directory sector
WRVTC   =       $1095   ;write volume table of contents
DELDOS  =       $1219   ;set no DOS
ERDBAD  =       $12B5   ;normal bad disk sector exit
DIR     =       $1401   ;file directory buffer

;       Patches to DOS 2.0S

        ORG     $078C
        LDA     #0      ;no retry on errors

        ORG     $0D4C
        JMP     $0D52   ;bypass bad sector errors

        ORG     $0D8E
        JSR     BD5     ;patch new error handler

        ORG     $0DA5
        BNE     $0D9F   ;do all but first sector
        JSR     WRTD0   ;write first directory sector

;       Routine to mark bad sectors as being
;       in use on the disk's VTOC. Patched into
;       the DOS at address $0D8E.

        ORG     $0600   ;we had to put it someplace!

BD5     LDA     #0      ;initialize
        STA     BADCNT  ;bad sector counter
        STA     BDSPT   ;and bad sector index
BD5LP   LDY     BDSPT   ;load index
        LDA     (BAD),Y ;get bad sector (low)
        STA     BSNUM   ;store it for later
        INY             ;increment pointer
        LDA     (BAD),Y ;get bad sector (high)
        STA     BSNUM+1 ;store it too
        INY             ;increment pointer again
        CMP     #$FF    ;end of data?
        BNE     BDCONT  ;No. not yet
        CMP     BSNUM   ;is low byte $FF?
        BNE     BDCONT  ;No. not at end yet.
        JMP     WRVTC   ;Yes. Write VTOC
BDCONT  STY     BDSPT   ;save index
        LDA     #0      ;Zero accumulator
        LDY     #3      ;shift sector number
BS1     LSR     BSNUM+1 ;3 bits to the right
        ROR     BSNUM   ;through high and low bytes
        ROR     A       ;rem goes in A as XXX00000
        DEY             ;decrement count
        BNE     BS1     ;Done 3 times? No.
        ROL     A       ;Yes. rotate a left 4
        ROL     A       ;times, so that it will
        ROL     A       ;have data in low bits
        ROL     A       ;and look like 00000XXX
        TAY             ;use value as counter
        LDA     #0      ;Zero accumulator
        SEC             ;set carry flag
BS3     ROR     A       ;rotate carry through Acc
        DEY             ;decrement counter
        BPL     BS3     ;At Y'th position? No.
        TAX             ;Yes. save bit mask in X
        LDA     BSNUM   ;get byte number
        ADC     #10     ;add offset to sector map
        TAY             ;use as VTOC index
        TXA             ;get bit mask back
        AND     (VTC),Y ;AND with VTOC
        BNE     BS4     ;Bad sector in use? No.
        PLA             ;YES! We are in trouble!
        PLA             ;pull return stack address
        JMP     ERDBAD  ;and report error condition
BS4     TXA             ;get bit mask again
        EOR     (VTC),Y ;invert allocation bit
        STA     (VTC),Y ;and store it back in VTOC
        LDY     #3      ;point to free sectors
        LDA     (VTC),Y ;byte in VTOC
        SEC             ;set carry for subtract
        SBC     #1      ;decrement number by one
        STA     (VTC),Y ;save it out again
        INC     BADCNT  ;increment bad sector count
        JMP     BD5LP   ;loop back and do it again.

;       If no bad sector errors, then just write
;       out the first directory sector to disk.
```

```
;       If there are errors, we will put a file
;       entry into the directory telling how many
;       sectors are bad and then write it to disk.

WRTD0   LDY     BADCNT  ;get bad sector count
        BEQ     NOERRS  ;Bad sectors? NO.
        LDX     #10     ;Yes. File name to entry
MOVFN   LDA     BADFN,X ;from BADFN
        ORA     #$80    ;inverse video, WOW!
        STA     DIR+5,X ;to directory buffer area
        DEX             ;decrement counter
        BPL     MOVFN   ;Done 11 bytes? No.
        LDA     #$60    ;mark file as locked
        STA     DIR     ;and in use
        STY     DIR+1   ;store bad sector count
NOERRS  JSR     WRTDIR  ;write sector to disk
        JMP     DELDOS  ;mark disk with no DOS

RETURN  RTS             ;return after patching

BADFN   DB      'Bad S' ;file name used to mark
        DB      'ectors';the disk as damaged.

BSNUM   DS      2       ;bad sector number
BDSPT   DS      1       ;bad sector pointer
BADCNT  DS      1       ;bad sector count

        END     RETURN  ;just return after loading
```

●

Swirl Demo

```
10 C=0:Q=1:SETCOLOR 1,5,5:DEG
20 XI=80:YI=50:GRAPHICS 23
30 PLOT XI,YI
40 FOR I=1 TO 1000 STEP 5
50 Q=Q+1:IF Q>3.5 THEN Q=1
60 COLOR Q:R=I/10:T=I
70 X=R*COS(T):Y=R*SIN(T)
80 IF Y+YI<0 THEN 140
90 PLOT X+XI,Y+YI
100 X=(I+C)/16*COS(I+C+90)
110 Y=(I+C)/16*SIN(I+C+90)
120 DRAWTO X+XI,Y+YI
130 NEXT I
140 SETCOLOR 2,8,2:SETCOLOR 1,8,5
150 SETCOLOR 0,8,8:GOSUB 210
160 SETCOLOR 0,8,2:SETCOLOR 2,8,5
170 SETCOLOR 1,8,8:GOSUB 210
180 SETCOLOR 1,8,2:SETCOLOR 0,8,5
190 SETCOLOR 2,8,8:GOSUB 210
200 GOTO 140
210 FOR K=1 TO 13:NEXT K:RETURN
```

CHECKSUM DATA

(See pgs. 7-10)

```
10 DATA 977,537,75,330,279,7,362,850,4
82,117,132,836,737,370,877,6968
160 DATA 373,886,376,895,697,163,3390
```

THE BLACK RABBIT 2.0

48K Disk

by Brian Moriarty

Let's face it. Backing up disks with a single drive is a dull and time-consuming chore. Even with a 48K system, ATARI DOS will make you swap at least three times to copy a reasonably full disk. And then there are those disks DOS won't copy — boot-load programs, **Letter Perfect** files, FORTH screens, anything recorded with a non-DOS file structure.

One day I got sick of disk-swapping and decided to write a more efficient disk backup system. I wanted to be able to duplicate all 720 sectors of a disk with no more than two read/write passes. To accomplish this,I had to find a way to cram 360 sectors worth of data into RAM at once — 46080 bytes!

A 48K ATARI contains 49152 bytes of user RAM. But the first four pages (1024 bytes) are reserved for use by the operating system ROM routines. A graphics mode 0 screen and display list require an additional 993 bytes. This leaves a maximum of 1055 bytes for the disk copier.

The Black Rabbit fits into this cramped space with room to spare. Version 2.0 features simple one-button operation with audio/visual prompting, automatic formatting of the destination disk and a "Visible VTOC" (Volume Table of Contents) that lets you check the distribution of data on the source disk and monitor the progress of the copy. It "skips over" empty sectors and will not crash if it encounters an unreadable sector.

Typing it in.

Listing 1 is an ATARI BASIC program that will create an auto-booting image of the Black Rabbit on any disk. **Listing 2** is the assembly-language source code, created with the **MAC/65** Macro Assembler by OSS. This listing is only provided to show you how the program works; you do NOT have to type it in to use the Rabbit.

Enter each line of the BASIC program carefully. Be especially careful with the DATA statements in lines 1000-1290. When you're finished, LIST the program out to disk and use D:CHECK2 (see page 9) to verify the accuracy of your typing. Use the following procedure to write your copy of Black Rabbit 2.0:

1. Load the BASIC program into memory and type RUN. The line numbers between 1000-1290 will be displayed as each DATA statement is checked. If bad data is encountered, the program will list the line containing the error and stop so that you can correct it. Re-RUN the program until all data lines are thoroughly debugged.
2. You will next be prompted to insert a blank disk into drive #1. *Make sure this disk contains no important programs or data, because it is about to be completely erased.*
3. Press the START key. The destination disk will be formatted and a copy of the Black Rabbit will be written out to the first six sectors. An error message will result if the disk is write-protected or cannot be formatted.
4. The prompt "Rabbit disk okay" means success! Remove the Rabbit disk from the drive, replace it with one of your regular DOS disks and SAVE the BASIC program. You can use it to make extra back-up copies of the Rabbit.

Rabbit, Run.

Now it's time to test the Black Rabbit. Re-insert the Rabbit disk in drive #1, turn off your computer, let it rest for a moment and turn it back on.

If you see "Remove cartridge; requires 48K RAM" on your screen, you forgot to remove the BASIC cartridge. The Black Rabbit needs every byte your computer can spare, and the cartridge de-selects an 8K block of RAM. So pull the cartridge out and power-up again. You should now be looking at the Rabbit's title screen (**Figure 1**).

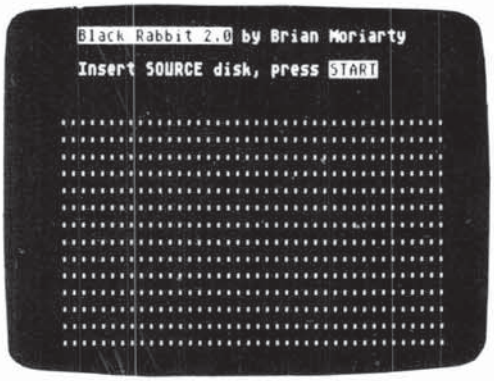


Figure 1

The 18x40 dot matrix on the bottom half of the screen is the Rabbit's "Visible VTOC." Each dot represents one of the 720 sectors on a standard ATARI disk.

Put the disk you want to copy into drive #1 and press the START key. The drive will begin spinning and you will hear the familiar beep-beep-beep of sectors being read into memory. As each sector is read, the corresponding dot in the Visible VTOC will change to a different character:

□ indicates a data sector
0 indicates an empty sector
? indicates an unreadable sector.

The Rabbit will beep again when his memory buffer is full. Remove the source disk, insert a blank copy disk and press START. Your copy disk will be formatted and the source data will be written out, one sector at a time. Each written sector will change its corresponding dot in the VTOC to an inverse dot character. Note that the Rabbit always uses the write-with-verify function of the 810 disk drive. It's slower than writing without verify, but more reliable.

The prompt "Insert source disk, press START" will re-appear at the end of the first read/write pass. Repeat the procedure outlined above. At the end of the final read/write pass, the Rabbit will offer to make another copy. Press START to re-run the Rabbit or press OPTION to boot the copy disk.

Empty and/or unreadable source sectors do not take up any memory in the Rabbit's disk buffer. So if the source disk has lots of empty or bad sectors, the Rabbit may be able to duplicate the whole thing with a single read/write pass. In any case, it will never take more than two swaps to copy an entire disk. □

Listing 1.

```
100 REM ******************************
110 REM *  BLACK RABBIT 2.0 MAKER   *
120 REM *  BY BRIAN MORIARTY        *
130 REM *  ANALOG COMPENDIUM V.1    *
140 REM ******************************
150 REM
160 CLR :DIM BUF$(768),ML$(4):FOR I=1
TO 4:READ BYTE:ML$(I)=CHR$(BYTE):NEXT
I:POKE 752,1
170 BUF$(1)="♥":BUF$(768)="♥":BUF$(2)=
BUF$:? "↰↓Verifying DATA lines.":? "↓R
eading Line ";
180 B=0:TOTAL=B:LINE=990:RESTORE 1000:
TRAP 250
190 LINE=LINE+10
200 POSITION 15,3:? LINE
210 FOR I=1 TO 25:B=B+1:READ BYTE:TOTA
L=TOTAL+BYTE:BUF$(B,B)=CHR$(BYTE):NEXT
 I
220 IF PEEK(183)+256*PEEK(184)<>LINE T
HEN ? "↓Line ";LINE;"missing.":END
230 READ CHECKSUM:IF CHECKSUM=TOTAL TH
EN 190
240 GOTO 360
250 POKE 752,0:IF PEEK(195)<>6 THEN 36
0
260 ? "↓DATA lines verified.":? "↓Inse
rt a blank disk in Drive #1."
270 ? "↓Press START to write disk.■"
280 IF PEEK(53279)<>6 THEN 280
290 POKE 769,1:POKE 770,33:? "↓Formatt
ing disk.":X=USR(ADR(ML$))
300 IF PEEK(771)<>1 THEN ? "↰↓Format e
rror!":? "Remove write-protect tab or"
:? "replace disk.":GOTO 270
310 ? "↓Writing data.":POKE 770,87:POK
E 779,0:BUFFER=ADR(BUF$)
320 FOR SECTOR=1 TO 6
330 POKE 778,SECTOR:POKE 773,INT(BUFFE
R/256):POKE 772,BUFFER-(256*PEEK(773))
:X=USR(ADR(ML$))
340 BUFFER=BUFFER+128:NEXT SECTOR
350 ? "↓Rabbit disk okay.":END
360 ? "Bad data at line ■";LINE:LIST L
INE:END
370 DATA 104,76,83,228
380 REM * M/L DATA
1000 DATA 0,6,128,4,134,4,169,0,168,14
5,94,32,77,6,165,106,201,192,176,12,16
9,158,133,134,169,2582
1010 DATA 6,32,131,6,76,157,4,162,0,14
2,198,2,134,129,134,131,142,68,2,232,1
34,128,134,130,134,5130
1020 DATA 9,142,1,3,134,144,24,165,88,
105,239,133,136,133,138,165,89,105,0,1
33,137,133,139,32,77,7734
1030 DATA 6,169,192,133,134,169,6,32,1
31,6,24,165,88,105,122,133,140,144,2,2
30,141,230,138,208,2,10584
1040 DATA 230,139,162,2,160,0,169,14,1
45,138,200,192,240,208,249,24,165,138,
105,240,133,138,144,2,230,14151
1050 DATA 139,202,16,231,32,143,6,24,1
65,136,101,128,133,138,165,137,101,129
,133,139,169,226,133,134,169,17380
1060 DATA 6,32,131,6,32,91,6,169,82,14
1,2,3,165,128,141,10,3,165,129,141,11,
3,32,83,228,19320
1070 DATA 173,3,3,16,4,169,31,208,15,1
60,127,177,132,208,7,136,16,249,169,16
,208,2,169,128,133,21979
1080 DATA 143,160,0,132,77,145,138,230
,138,208,2,230,139,230,128,208,2,230,1
29,165,129,201,2,208,6,25359
1090 DATA 165,128,201,209,240,31,165,1
43,201,128,208,181,24,173,4,3,105,128,
141,4,3,133,132,173,5,28387
1100 DATA 3,105,0,141,5,3,133,133,201,
188,208,156,24,165,136,101,130,133,138
,165,137,101,131,133,139,31296
1110 DATA 169,4,133,134,169,7,32,131,6
,32,91,6,198,144,208,32,32,66,6,169,33
,141,2,3,32,33276
1120 DATA 83,228,173,3,3,201,1,240,14,
169,72,133,134,169,7,32,131,6,32,91,6,
240,224,32,143,35843
1130 DATA 6,169,87,141,2,3,165,130,141
,10,3,165,131,141,11,3,160,0,132,77,17
7,138,133,143,201,38312
1140 DATA 128,208,5,32,83,228,48,251,1
69,142,160,0,145,138,230,138,208,2,230
,139,230,130,208,2,230,41796
1150 DATA 131,165,131,201,2,208,6,165,
130,201,209,240,30,165,143,201,128,208
,193,24,173,4,3,105,128,45090
```

```
1160 DATA 141,4,3,173,5,3,105,0,141,5,
3,201,188,208,172,76,1,5,169,38,133,13
4,169,7,32,47206
1170 DATA 131,6,173,31,208,201,6,240,1
0,201,3,208,245,32,123,6,76,119,228,32
,123,6,76,160,4,49854
1180 DATA 169,0,141,4,3,169,4,141,5,3,
96,24,165,88,105,42,133,140,165,89,105
,0,133,141,96,52015
1190 DATA 169,100,141,0,210,169,170,14
1,1,210,169,0,133,20,165,20,201,15,208
,250,169,0,141,1,210,55028
1200 DATA 173,31,208,201,6,208,249,173
,31,208,201,7,208,249,96,133,135,160,3
3,177,134,145,140,136,16,58486
1210 DATA 249,96,169,128,133,132,141,4
,3,169,7,133,133,141,5,3,96,50,101,109
,111,118,101,0,99,60917
1220 DATA 97,114,116,114,105,100,103,1
01,27,0,114,101,113,117,105,114,101,11
5,0,20,24,43,0,50,33,62844
1230 DATA 45,162,236,225,227,235,128,1
78,225,226,226,233,244,128,146,142,144
,0,98,121,0,34,114,105,97,66563
1240 DATA 110,0,45,111,114,105,97,114,
116,121,41,110,115,101,114,116,0,51,47
,53,50,35,37,0,100,68466
1250 DATA 105,115,107,12,0,112,114,101
,115,115,0,179,180,161,178,180,0,0,0,4
1,110,115,101,114,116,70837
1260 DATA 0,35,47,48,57,0,100,105,115,
107,12,0,112,114,101,115,115,0,179,180
,161,178,180,0,0,72898
1270 DATA 0,0,0,179,180,161,178,180,0,
116,111,0,114,101,13,114,117,110,12,0,
175,176,180,169,175,75459
1280 DATA 174,0,116,111,0,98,111,111,1
16,0,0,0,50,101,112,108,97,99,101,0,98
,97,100,0,35,77294
1290 DATA 47,48,57,0,100,105,115,107,1
2,0,112,114,101,115,115,0,179,180,161,
178,180,0,0,0,0,79320
```

•

CHECKSUM DATA

(See pgs. 7-10)

```
100 DATA 539,353,257,477,551,89,898,13
3,283,432,225,174,837,123,719,6090
250 DATA 919,811,216,854,128,274,351,7
80,597,782,905,252,816,404,37,8126
1010 DATA 166,300,388,806,148,586,481,
553,689,946,564,821,75,559,705,7787
1160 DATA 357,84,666,352,37,172,294,89
0,262,573,939,204,656,993,6479
```

•

Listing 2.

```
0100 ; **********************
0110 ; *  Black Rabbit 2.0  *
0120 ; **********************
0130 ;
0140 ; Highspeed sector copier
0150 ; for single-drive systems
0160 ;
0170 ; by Brian Moriarty
0180 ; ANALOG Compendium Volume 1
0190 ;
0200 ; OS disk handler equates
0210 ;
0220 DEVNUM = $0301
0230 DCOMND = $0302
0240 DSTATS = $0303
0250 DBUFLO = $0304
0260 DBUFHI = $0305
0270 SECTLO = $030A
0280 SECTHI = $030B
0290 DVSTAT = $02EB
0300 DISKIO = $E453
0310 ;
0320 ; Disk handler commands
0330 ;
0340 READ =  $52
0350 WRITE = $57
0360 FORMAT = $21
0370 ;
0380 ; Misc. system equates
0390 ;
0400 COLDST = $0244
0410 BOOT? = $09
0420 SAVMSC = $58
0430 COLOR2 = $02C6
0440 OLDADR = $5E
0450 CONSOL = $D01F
0460 RAMTOP = $6A
0470 AUDF1 = $D200
0480 AUDC1 = $D201
0490 RTCLOK = $14
0500 ATRACT = $4D
0510 COLDSV = $E477
0520 ;
0530 ; Internal program equates
0540 ;
0550 RTOTAL = $80
0560 WTOTAL = $82
0570 BPOINT = $84
0580 PPOINT = $86
0590 SCREEN = $88
0600 VTOC =  $8A
0610 LINE =  $8C
0620 SAVEY = $8E
0630 SBYTE = $8F
0640 FFLAG = $90
0650 ;
0660 ; Characters for "Visible VTOC"
0670 ;
0680 DOT =   $0E
0690 DATA =  $80
0700 BAD =   $1F
0710 WRITTEN = $8E
0720 NOTHING = $10
0730 ;
0740 ; Memory usage
0750 ;
0760 DUMMY = $0400   ; Dummy buffer
0770 ORIGIN = $0480  ; Program start
0780 BUFFER = $0780  ; Data buffer
0790 ;
0800     *=  ORIGIN
0810 ;
0820 ; 6 bytes to control boot-up
0830 ;
0840     .BYTE $00,$06 ; # boot sects
0850     .BYTE ORIGIN&255,ORIGIN/256
0860     .BYTE ENTRY&255,ENTRY/256
0870 ;
0880 ENTRY
0890 ;
0900 ; Init screen line pointer
0910 ;
0920     LDA #0
0930     TAY
0940     STA (OLDADR),Y ; Kill cursor
0950     JSR TOPLINE
0960 ;
0970 ; Check for 48K RAM
0980 ;
0990     LDA RAMTOP
1000     CMP #$C0    ; $C0 = 48K
1010     BCS RABBIT  ; ) OR = 48K
1020 ;
1030 ; Print RAM warning
1040 ;
```

```
1050     LDA #WARNING&255
1060     STA PPOINT
1070     LDA #WARNING/256
1080     JSR MESSAGE
1090 ;
1100 FREEZE
1110     JMP FREEZE  ; Infinite loop
1120 ;
1130 ; ********************
1140 ; *  Initialize R/W  *
1150 ; ********************
1160 ;
1170 RABBIT
1180 ;
1190 ; Initialize important things
1200 ;
1210     LDX #0      ; Black
1220     STX COLOR2  ; Background
1230     STX RTOTAL+1 ; Clear MSB
1240     STX WTOTAL+1 ; Ditto
1250     STX COLDST  ; Coldstart flag
1260     INX         ; X = 1
1270     STX RTOTAL  ; LSB
1280     STX WTOTAL  ; Ditto
1290     STX BOOT?   ; Boot flag
1300     STX DEVNUM  ; Drive #1
1310     STX FFLAG   ; Format enable
1320 ;
1330 ; Setup VTOC screen pointer
1340 ;
1350     CLC
1360     LDA SAVMSC  ; Addr of screen
1370     ADC #239    ; 6 lines down
1380     STA SCREEN
1390     STA VTOC
1400     LDA SAVMSC+1
1410     ADC #0
1420     STA SCREEN+1
1430     STA VTOC+1
1440 ;
1450 ; Print title
1460 ;
1470     JSR TOPLINE
1480     LDA #TITLE&255
1490     STA PPOINT
1500     LDA #TITLE/256
1510     JSR MESSAGE
1520 ;
1530 ; Reset screen pointer
1540 ;
1550     CLC
1560     LDA SAVMSC
1570     ADC #122    ; X=2, Y=3
1580     STA LINE
1590     BCC DODOTS
1600     INC LINE+1
1610 ;
1620 ; Init VTOC display matrix
1630 ;
1640 DODOTS
1650     INC VTOC
1660     BNE MATRIX
1670     INC VTOC+1
1680 MATRIX
1690     LDX #2
1700 LOOP1
1710     LDY #0
1720     LDA #DOT
1730 LOOP2
1740     STA (VTOC),Y
1750     INY
1760     CPY #240
1770     BNE LOOP2
1780     CLC
1790     LDA VTOC
1800     ADC #240
1810     STA VTOC
1820     BCC MORE
1830     INC VTOC+1
1840 MORE
1850     DEX
1860     BPL LOOP1
1870 ;
1880 ; ********************
1890 ; *  READ Routine  *
1900 ; ********************
1910 ;
1920 READER
1930 ;
1940 ; Reset buffer addr pointers
1950 ;
1960     JSR REPOINT
1970 ;
1980 ; Update VTOC pointer
1990 ;
2000     CLC
2010     LDA SCREEN
2020     ADC RTOTAL
2030     STA VTOC
2040     LDA SCREEN+1
2050     ADC RTOTAL+1
2060     STA VTOC+1
2070 ;
2080 ; Print READ prompt
2090 ;
2100     LDA #RPROMPT&255
2110     STA PPOINT
2120     LDA #RPROMPT/256
2130     JSR MESSAGE
2140 ;
2150     JSR WAIT    ; START key
2160 ;
2170     LDA #READ
2180     STA DCOMND  ; Set READ mode
2190 ;
2200 ; ************************
2210 ; *  Start of READ loop  *
2220 ; ************************
2230 ;
2240 RLOOP
2250 ;
2260 ; Update sector #
2270 ;
2280     LDA RTOTAL
2290     STA SECTLO
2300     LDA RTOTAL+1
2310     STA SECTHI
2320 ;
2330     JSR DISKIO  ; Fetch sector
2340     LDA DSTATS  ; Check status
2350     BPL SECSTAT ; Branch if okay
2360     LDA #BAD
2370     BNE SHOWSTAT
2380 ;
2390 ; Check sector data for status
2400 ;
2410 SECSTAT
2420     LDY #$7F
2430 NEXTBYTE
2440     LDA (BPOINT),Y
2450     BNE DATAID
2460     DEY
2470     BPL NEXTBYTE
2480     LDA #NOTHING
2490     BNE SHOWSTAT
2500 DATAID
2510     LDA #DATA
2520 SHOWSTAT
2530     STA SBYTE
2540     LDY #0
2550     STY ATRACT  ; Attract off
2560     STA (VTOC),Y
2570 ;
2580 ; Update VTOC addr pointer
2590 ;
```

```
2600      INC VTOC
2610      BNE UPCOUNT
2620      INC VTOC+1
2630 UPCOUNT
2640      INC RTOTAL
2650      BNE SECTMAX
2660      INC RTOTAL+1
2670 ;
2680 ; End of disk?
2690 ;
2700 SECTMAX
2710      LDA RTOTAL+1
2720      CMP #$02
2730      BNE DATACHECK
2740      LDA RTOTAL
2750      CMP #$D1
2760      BEQ WRITER
2770 ;
2780 ; Check for data sector
2790 ;
2800 DATACHECK
2810      LDA SBYTE
2820      CMP #DATA
2830      BNE RLOOP
2840 ;
2850 ; Add 128 to buffer pointers
2860 ;
2870      CLC
2880      LDA DBUFLO
2890      ADC #$80
2900      STA DBUFLO
2910      STA BPOINT
2920      LDA DBUFHI
2930      ADC #0
2940      STA DBUFHI
2950      STA BPOINT+1
2960 ;
2970 ; Check if buffer full
2980 ;
2990      CMP #$BC    ; Top of buffer?
3000      BNE RLOOP   ; No; keep going
3010 ;
3020 ; ******************
3030 ; *  WRITE Routine  *
3040 ; ******************
3050 ;
3060 WRITER
3070 ;
3080 ; Init VTOC pointer
3090 ;
3100      CLC
3110      LDA SCREEN
3120      ADC WTOTAL
3130      STA VTOC
3140      LDA SCREEN+1
3150      ADC WTOTAL+1
3160      STA VTOC+1
3170 ;
3180 ; Print WRITE prompt
3190 ;
3200      LDA #WPROMPT&255
3210      STA PPOINT
3220      LDA #WPROMPT/256
3230      JSR MESSAGE
3240 ;
3250      JSR WAIT    ; START key
3260 ;
3270      DEC FFLAG
3280      BNE NOFORM  ; Skip if Pass 2
3290 ;
3300 ; Format disk
3310 ;
3320 ERASE
3330      JSR DUMPOINT ; buffer addr
3340      LDA #FORMAT
3350      STA DCOMND  ; format cmnd
3360      JSR DISKIO  ; Do it!
3370 ;
3380 ; Check for okay format
3390 ;
3400      LDA DSTATS
3410      CMP #1
3420      BEQ NOFORM
3430 ;
3440 ; Print bad format warning
3450 ;
3460      LDA #BADFORM&255
3470      STA PPOINT
3480      LDA #BADFORM/256
3490      JSR MESSAGE
3500      JSR WAIT
3510      BEQ ERASE
3520 ;
3530 NOFORM
3540 ;
3550      JSR REPOINT ; Reset pntrs
3560 ;
3570      LDA #WRITE
3580      STA DCOMND  ; WRITE command
3590 ;
3600 ; ************************
3610 ; *  Start of WRITE loop  *
3620 ; ************************
3630 ;
3640 WLOOP
3650 ;
3660 ; Update setor #
3670 ;
3680      LDA WTOTAL
3690      STA SECTLO
3700      LDA WTOTAL+1
3710      STA SECTHI
3720 ;
3730 ; Get status of next read
3740 ;
3750      LDY #0
3760      STY ATRACT
3770      LDA (VTOC),Y
3780      STA SBYTE
3790 ;
3800 ; Branch depending on status
3810 ;
3820      CMP #DATA
3830      BNE SKIPSECT ; If no data
3840 ;
3850 DWRITE
3860      JSR DISKIO  ; Write sector
3870      BMI DWRITE
3880 ;
3890 ; Display write status
3900 ;
3910 SKIPSECT
3920      LDA #WRITTEN
3930      LDY #0
3940      STA (VTOC),Y
3950 ;
3960 ; Update VTOC, WTOTAL
3970 ;
3980      INC VTOC
3990      BNE WRUP
4000      INC VTOC+1
4010 WRUP
4020      INC WTOTAL
4030      BNE WSECTMAX
4040      INC WTOTAL+1
4050 WSECTMAX
4060      LDA WTOTAL+1
4070      CMP #$02
4080      BNE BUFLOOK
4090      LDA WTOTAL
4100      CMP #$D1
4110      BEQ FINISHED
4120 ;
4130 ; Should buffer addr be updated?
4140 ;
```

```
4150 BUFLOOK
4160     LDA SBYTE
4170     CMP #DATA   ; Update bufadr?
4180     BNE WLOOP   ; No; next sect
4190 ;
4200 ; Update buffer address
4210 ;
4220     CLC
4230     LDA DBUFLO
4240     ADC #$80
4250     STA DBUFLO
4260     LDA DBUFHI
4270     ADC #0
4280     STA DBUFHI
4290 ;
4300 ; Buffer full?
4310 ;
4320 FULBUF
4330     CMP #$BC
4340     BNE WLOOP
4350     JMP READER  ; Next pass
4360 ;
4370 ; *****************
4380 ; *  End routine  *
4390 ; *****************
4400 ;
4410 FINISHED
4420     LDA #COMPLETE&255
4430     STA PPOINT
4440     LDA #COMPLETE/256
4450     JSR MESSAGE
4460 DECIDE
4470     LDA CONSOL
4480     CMP #6      ; START press?
4490     BEQ RERUN
4500     CMP #3      ; OPTION?
4510     BNE DECIDE
4520     JSR LETGO
4530     JMP COLDSV  ; Cold boot
4540 RERUN
4550     JSR LETGO
4560     JMP RABBIT  ; Re-run Rabbit
4570 ;
4580 ; *****************
4590 ; *  Subroutines  *
4600 ; *****************
4610 ;
4620 ; Point to dummy buffer
4630 ;
4640 DUMPOINT
4650     LDA #DUMMY&255
4660     STA DBUFLO
4670     LDA #DUMMY/256
4680     STA DBUFHI
4690     RTS
4700 ;
4710 ; Point to top screen line
4720 ;
4730 TOPLINE
4740     CLC
4750     LDA SAVMSC
4760     ADC #42     ; X=2, Y=1
4770     STA LINE
4780     LDA SAVMSC+1
4790     ADC #0
4800     STA LINE+1
4810     RTS
4820 ;
4830 ; Beep and wait for START key
4840 ;
4850 WAIT
4860     LDA #100    ; Freq = 100
4870     STA AUDF1
4880     LDA #$AA    ; D & V = 10
4890     STA AUDC1
4900     LDA #0
4910     STA RTCLOK  ; Clear count
4920 BEEP
4930     LDA RTCLOK
4940     CMP #15     ; 1/4 sec
4950     BNE BEEP
4960     LDA #0
4970     STA AUDC1   ; Silence!
4980 ;
4990 ; Check key
5000 ;
5010 HOLDIT
5020     LDA CONSOL
5030     CMP #6
5040     BNE HOLDIT  ; Pressed?
5050 LETGO
5060     LDA CONSOL
5070     CMP #7
5080     BNE LETGO   ; Till released
5090     RTS
5100 ;
5110 ; Print text messages
5120 ;
5130 MESSAGE
5140     STA PPOINT+1
5150     LDY #33
5160 NEXTPRINT
5170     LDA (PPOINT),Y
5180     STA (LINE),Y
5190     DEY
5200     BPL NEXTPRINT
5210     RTS
5220 ;
5230 ; Set buffer pointers
5240 ;
5250 REPOINT
5260     LDA #BUFFER&255
5270     STA BPOINT
5280     STA DBUFLO
5290     LDA #BUFFER/256
5300     STA BPOINT+1
5310     STA DBUFHI
5320     RTS
5330 ;
5340 ; *******************
5350 ; *  Message texts  *
5360 ; *******************
5370 ;
5380 WARNING
5390     .SBYTE "Remove cartridge; requires 48K RAM"
5400 ;
5410 TITLE
5420     .SBYTE "Black Rabbit 2.0 by Brian Moriarty"
5430 ;
5440 RPROMPT
5450     .SBYTE "Insert SOURCE disk, press START   "
5460 ;
5470 WPROMPT
5480     .SBYTE "Insert COPY disk, press START     "
5490 ;
5500 COMPLETE
5510     .SBYTE "START to re-run, OPTION to boot   "
5520 ;
5530 BADFORM
5540     .SBYTE "Replace bad COPY disk, press START"
5550 ;
5560     .END
```

●

DISKTOOL REV.3

32K Disk

by Tony Messina

Disk Tool is designed to work with an ATARI 400/800/1200 with at least 32K of memory and up to 4 single-density disk drives. The key is SINGLE density. PERCOM, RANA, MICRO-MAINFRAME and other *double* density drives can run **Disk Tool**, but only in the single-density mode. Sorry, but **Disk Tool** was designed and written back in the olden days BD (before double density), and would require a complete overhaul in every aspect.

Disk Tool history.

My need for a disk utility made its appearance shortly after my disk drive arrived in March, 1981. I was plagued with disk link errors and crashed files all over the place. To put it mildly, "Boy, was I really mad!" It was then I decided to write a program that would allow me to access any sector on the disk. To make a long story short, I got a copy of the DOS 1 source listing and ATARI Tech Manual. I then locked myself in the den and proceeded to work. 50 gallons of coffee, two power outages and 5 billion phone calls to ATARI later, I emerged victorious. I had actually managed to READ and WRITE to a disk sector without using the File Management System (FMS) or Utility Code in DOS 1. Yaaayy!!

When DOS 2 arrived on the scene, I converted the **Tool**. Some letter I had received prompted me to organize the **Tool** and publish it as a 2-part article in **A.N.A.L.O.G. Computing**. Response to the program and the article was outrageous. When **A.N.A.L.O.G.** editor Lee Pappas mumbled something about a **Compendium**, I saw the opportunity not only to improve the article and documentation, but also the method by which I could include the most requested enhancements to the **Tool**. So here it is, everything you ever wanted to know about disk structures and **Disk Tool**. And away we go...

Disk sector structure.

The ATARI 810 disk drive, in conjunction with the File Management System (FMS), organizes data on a diskette into blocks called sectors. There are 720 sectors (numbered from 0-719) on each diskette after it is formatted by the Disk Operating System. The sectors are laid out in what are known as tracks. There are 40 tracks per diskette, each containing 18 sectors. To clarify the last two statements, I have my patented "formatted diskettes are like onions" dog and pony show. Next time you cut an onion in half (when you make onion rings, mushrooms and onions, etc.), lop off a hunk in the middle about 1/4 inch wide. Now turn the onion so that the big round part faces you. Each individual ring of that onion is exactly similar to a track on the diskette. Go ahead, pull off the outer ring. Now, if you cut that ring into 18 equal pieces, each piece would represent a sector. The outer ring is track 0. As you move inward, the next ring is track 1 and so forth until you reach track 39. Each track would contain 18 sectors. Track 0 contains sectors 0-17, track 1 has sectors 18-35, etc.

Now you have an idea of how a diskette is organized. **Disk Tool** is designed to work at the sector level. Although there are 720 sectors on each diskette, not all sectors are available to you, the user.

You've just formatted a diskette. Ahhh, the feeling of power, 720 sectors to store all of your programs. You hit the A OPTION in DOS (just to see that magic number 719). Upon hitting RETURN, the number 707 appears when using DOS 2 and 709 appears when using DOS 1. What! What happened? Well, it's quite simple, friends. Although there are 720 sectors, only 707 are available for your use with DOS 2 and 709 sectors with DOS 1. The other sectors are reserved for use by DOS. The disk directory steals 8 sectors starting at sector 361 and running to 368. One sector (360) is allocated for the VTOC (Volume Table of Contents, pronounced "Vee-Talk"). The boot portion of FMS also occupies 3 sectors (1,2,3) for DOS 2, only 1 sector for DOS 1. That's what happened to your 12 missing sectors for DOS 2 and 10 missing sectors for DOS 1, so don't be alarmed.

With that out of the way, it's time to discuss the different types of sectors. Yes, I know it sounds confusing. . . after all, isn't a sector a sector? The answer is yes. Each sector is capable of holding 128 bytes of data. The manner in which the data is structured on a sector is dependent on a particular sector's purpose or type. I like to define sectors as being of 4 types:

1.) **Data Sector:** Containing program information, text files, etc.

2.) **Boot Sector:** Containing ML program data.

3.) **Directory Sector:** Containing program names and associated data.

4.) **VTOC Sector:** Sector containing free count and disk bit map.

Let's take a look at the differences and similarities of each type of sector.

Data sectors.

This is the most common type of sector on your disk. Technically, all the sectors are data sectors. I use this name only to distinguish its format from other types of sectors.

Whenever you use the commands SAVE "D:XXX", LIST "D:XXX" or invoke the Binary Save option from DOS, the actual programs are written to the disk in data sector format. The format is quite simple. Bytes 0-124 contain actual program data. Bytes 125-127 contain sector identity data or "link data." **Figure 1** illustrates this type of format.

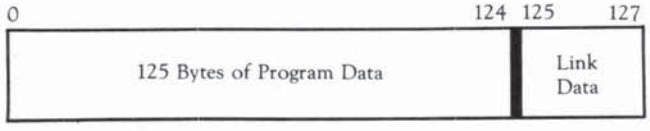


Figure 1.
Data Sector Format.

The link data for DOS 2 is formatted as in **Figure 2**, while link data for DOS 1 is as per **Figure 3**.

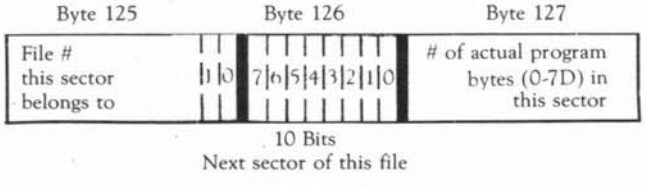
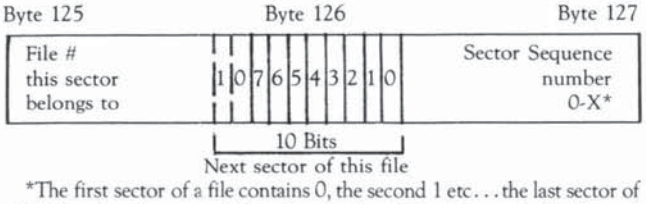


Figure 2.

DOS 2 link structure.



*The first sector of a file contains 0, the second 1 etc. . . the last sector of a file is unique however. The value in Byte 127 will contain (# of actual bytes used + 1) + $80 for the last sector of a file.

Figure 3.

DOS 1 link structure.

Notice that the lower two bits of byte 125 and all of byte 126 combined point to the next physical sector of this file. A zero (0) indicates that this is the last sector of a file.

One variation in data sector format occurs when the Binary Save option is used to save an area of memory to the disk. The variation occurs with the first 6 bytes of the first sector of the binary file. Those 6 bytes are commonly referred to as the "binary file header." The header is formatted as per **Figure 4.**

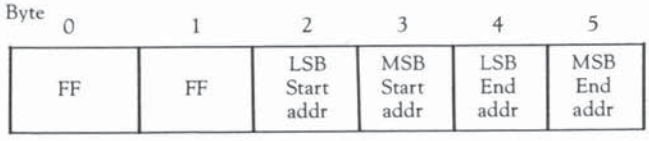


Figure 4.

If, for example, you answer the Binary Save Prompt DOS with "MLPROG,0600,065F," then the first 6 bytes of the first sector of disk storage for this program would look like **Figure 5.**

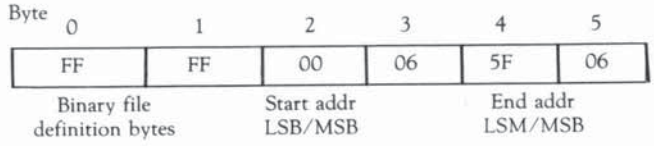


Figure 5.
Binary program save example.

Directory sectors.

There are 8 directory sectors starting at sector 361 and running sequentially to sector 368. The directory contains the names of all the programs on the diskette along with the other information about the program. Each directory entry uses 16 Bytes. There is enough room to hold 8 program names (and associated data) on 1 sector. (16 Bytes * 8 names + 128 Bytes or 1 sector.) Therefore with 8 sectors available, we can have (8 sectors *8 names per sector) = 64 possible file names total. On a directory read, DOS starts at sector 361 and keeps reading sectors until there are no more names. Directory entries have the following format:

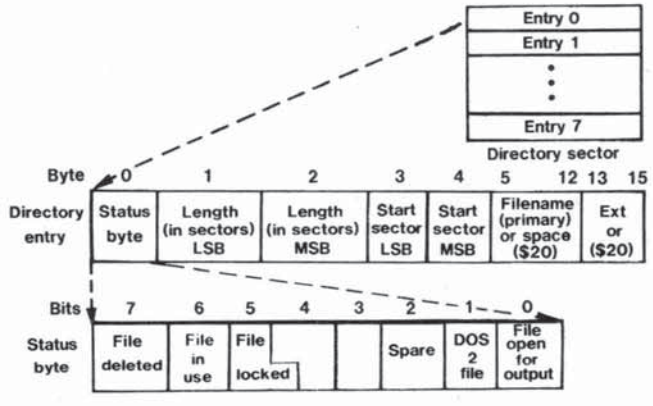


Figure 6.

NOTE: Bits set (+1) indicate condition listed.
Bit 6 set indicates the file is in use.
Bits can be combined for multiple status.

Example: Bits 1,5 and 6 set would mean file was created by DOS 2, it is locked and in use.

Here is a quick reference to the possible status values.

$00=File is never used.
$01=File open for output.
$02=File created by DOS 2 (if bit not set, assume DOS 1).
$20=File locked.
$40=File in use.
$80=Entry deleted.

How does DOS use the information we have discussed so far? In simple terms, when you type LOAD "D:XXX" in BASIC, the FMS opens the directory for input, reads in the directory sectors starting at sector 361 and searches for a match. If it finds an entry that matches the program name you asked for, FMS extracts the starting sector from bytes 3 and 4 of the entry and also the length from bytes 1 and 2. FMS then positions the read/write head of the disk drive at that sector, reads in the sector, extracts the link information (to find the next sector) and checks to see if this sector actually belongs to the file you wanted. If it does, then FMS checks to see if this is the last sector to load. (Remember, the next sector to load is in the link bytes.) FMS keeps loading until the next sector to load is 0.

If, during this process, the file number of the sector just loaded does not match the one you are looking for, a file number mismatch error (#164) occurs. This usually means that either the disk link information of the previous sector was incorrect, or possibly the link data of the current sector is incorrect. We'll discuss how to fix this later.

Boot sectors.

I use the term "boot sector format" when referring to files which start at sector 1 and run contiguously to sector X, where sector X is the ending sector. These files do not need any language cartridges or DOS. They are completely self-contained programs which load and execute upon powering up the computer. Do not confuse these with AUTORUN.SYS files.

Remember the header bytes for binary files saved using the Binary Save Option of DOS? Well, boot sectors have a similar structure. Sector 1 of the disk contains the magic header information which is structured as per **Figure 7.**

Byte 0	1	2	3	4	5
0 by tradition	# of sectors to load	LSB Load addr	MSB Load addr	LSB Init addr	MSB Init addr

Boot sector header (sector 1)

Figure 7.

Whenever you turn on your computer, a check is made to verify if any cartridges are present. If a cartridge is present, the "Allow Disk Boot Bit" (Bit 0 of location $BFFD) is checked. If it is zero (as it would be if no cartridge were present) then the ROM boot routine is invoked. This routine goes out to sector 1 of the disk, reads in the data contained there and interprets it. Byte 1 tells the system how many sectors to read. Bytes 2 and 3 tell the system where to load the data, and bytes 4 and 5 tell the system where to start executing the ML program once it is loaded.

Boot sectors do not have any link data. Consequently, each boot-type sector can contain 128 bytes of program information. I said "can" because the last sector may be a short sector containing less than 128 bytes. The FMS for DOS 2 contains 3 boot sectors worth of program data, while the FMS for DOS 1 has only 1 boot sector.

VTOC sector structure.

Sector 360 contains the VTOC or Volume Table of Contents. The purpose of the VTOC is to keep track of which sectors on the disk are or are not being used.

There are basically two important parts of the VTOC: the miscellaneous portion (bytes 0-4) and the sector use map (also called sector bit map). Bytes 0-4 are used as follows:

Byte 0=Use byte (2 for DOS 2, 1 for DOS 1).
Bytes 1 & 2=Total # of sectors (LO/HI format).
Bytes 3 & 4=Free sector count (LO/HI format).

The sector use map begins at byte $0A and runs to byte $63. Each bit of each byte represents one sector on the disk. If the bit is zero, then that sector is being used. If the bit is 1, then that sector is available for free. Bit 7 of byte $0A represents sector 0, which does not exist (see experiment 5 for explanation). Bit 6 of byte $0A represents sector 1, etc., all the way down to bit 1 of byte $63 which represents sector 719. Use **Disk Tool** to examine the maps of your own diskettes.

Well, that about wraps up our discussion on disk data structures. I realize I've clobbered you with many new concepts and material. The best way to digest this information is to use the **Disk Tool** experiments which follow.

Disk Tool structure.

You may have noticed that **Disk Tool** consists of 3 programs: an AUTORUN.SYS creator program, a machine language loader and the actual BASIC code. Why 3? Well, the original intent was to make **Disk Tool** fit into a 16K disk system. That was when the **Tool** was small. Now it's so huge that it won't all fit,

so I kept it at three programs. **Disk Tool** sets itself up as follows:

1.) Protect 3000 bytes of low end memory and disable the break key (via AUTORUN.SYS);

2.) Load the ML portion of **Disk Tool** into the protected area, and load the **Disk Tool** BASIC program;

3.) Execute **Disk Tool** from BASIC.

Since I believe it is more important to know how to use this utility, I won't get into a long-winded dissertation about how it works (as I haven't been long-winded already in this article). If you study the listings along with the documentation, you should be able to get a very good understanding of what is going on.

Warnings.

Disk Tool will happily allow you to wipe out your directory, the VTOC, DOS boot sectors or any other sector on your disk. It will ask you to verify prior to writing, but once a sector has been written, it may be too late. You don't need to be an advanced systems programmer to use **Disk Tool** — only a careful programmer. It is suggested that you read the descriptions of each function as presented, and then perform all of the experiments in order to become familiar with **Disk Tool** and its capabilities.

OK, warnings are behind us. Let's move on.

Getting things together.

The first thing to do is to get a new diskette, format it and write out DOS 2 to the diskette. Type the listings in order from program 1 to 3. SAVE the three programs to your new disk. Suggested filenames follow:

1.) **AUTORUN.SYS maker**→MAKAUTO.UTL

2.) ML loader→DSKTOOL.PT1

3.) Disk Tool BASIC→DSKTOOL.PT2

These are only suggestions. If you decide to rename the BASIC **Disk Tool** portion, you must change the RUN command in the ML Loader so that you don't get a file not found error. Run the AUTORUN.SYS maker first so it can create the AUTORUN.SYS file. Power down, power up with the same disk and type RUN "D:DSKTOOL.PT1."

A note on typing.

The program listings for **Disk Tool** are fairly large (that's an understatement). Suffice it to say if any data is missing or erroneously typed in, the **Tool** will not work correctly. I suggest that you purchase the disk version of this *Compendium*. You'll not only save yourself hours of typing, but you will be assured that all programs will work correctly. I have spent over 200 hours debugging, testing and ensuring that the listings presented here are exact duplicates of my working copy of **Disk Tool**. It really does work! And now back to our regularly scheduled **Disk Tool.**

Using Disk Tool (finally!).

I know everyone has **Disk Tool** running. (Those of you who don't keep trying.) The first thing you will see is the Command Menu and a "COMMAND OR SECTOR NUMBER" prompt. To examine any sector, just type in the number and hit RETURN. Only sectors 1-720 can be examined. Any number < 1 or > 720 will generate an error message. Sector numbers can be entered in either decimal or hex (if preceded by a $). Let's try it out. Put in any of your program diskettes.

Experiment #1: Look at Directory Sector.

Answer the prompt with 361 and hit RETURN. You will see the first sector of the directory. Compare each entry with the format of Figure 5. Once you feel comfortable with the format of the directory, move on to the next experiment.

Experiment #2: Look at Formatted Directory Output with "D" Command.

Answer the COMMAND OR SECTOR NUMBER prompt with a D and hit RETURN. A formatted display should appear. All numbers appear in hex notation. This option displays 2 sectors worth of directory data (16 program names). The sector number is the actual sector at which that directory entry resides. FILENAME is self-explanatory. START is the first disk sector which contains data pertaining to that program. LEN is the length or number of sectors that file contains. FIL# is the entry number in the directory for that file; and STAT is the file status in human readable form where:

*=File locked.
U= in use.
D=File has been deleted.
1=File created by DOS 1.
2=created by DOS 2.

To examine more directory sectors, hit "+" and press RETURN. The new sectors will appear. To abort the directory format, just hit RETURN and our friend "COMMAND OR SECTOR NUMBER" will appear.

Experiment #3: Trace/Examine a File.

Now find a file you want to examine from the directory listing. (Try one other than DOS.SYS or DUP.SYS.) Find the start sector number for that file under the START column. Since the start number is in hex, type $ followed by the number. You don't need to type in leading zeros. If the start number was 00BF, then type $BF, for 01CD type $1CD, etc. Then hit RETURN. The sector will appear in HEX/ATASCII format along with the sector number, next sector and file information. SECTOR NUMBER indicates the current sector number being

displayed. NEXT SECTOR points to the next sector containing data for this file. FILE NUMBER is the file number to which this sector belongs. The next sector does not have to be the current sector number +1 (more on this later).

When you're ready to look at the next sector, you can enter the number and hit RETURN. If the next sector happens to be the current sector +1, just hit RETURN or "+" and RETURN. If you want to look at the current sector -1, type "-" and RETURN. Trace your file, examining the format of the data, etc. Remember **Figure 1**. Try to look at all types of files: Binary, SAVE files, ASCII files, etc., and compare these with the appropriate figures. When you hit the end of a file, you'll see that the next sector pointer will equal zero.

Experiment #4: Change Bytes with "C" Command.

Call up sector 720 on the disk. If it is all zeroes then you can use it. If it isn't, type "-" and hit RETURN until a sector is displayed with all zeros. At the prompt COMMAND OR SECTOR NUMBER, type in C and hit RETURN. The screen should change to yellow and a prompt should appear. Move the cursor (CTRL up, down, left, right, arrow, etc.) to the 1st hex value in byte 00 line. Replace the 00 values with the following:

44 49 53 4B 54 4F 4F 4C

Then hit RETURN. Make sure you overwrite each value of 00 and space between each byte. If you have done everything correctly, you should see a "secret message."

The C function only changes memory locations. Nothing has been written to the disk. You can only change one display line at a time. RETURN must be hit after your line changes are satisfactory. If you wish to change more data on the sector, simply hit C again, make your change, hit RETURN, etc.

Experiment #4A: Change Bytes (ATASCII method).

Follow the procedure in Experiment #4. To change bytes, move the cursor over to the hex parameter to change. Hit the space bar to blank out the first parameter of the hex number. Now type the ATASCII letter or number you want. Continue with the rest of the line, always remembering to precede the character you want with a space. Hit return and check your work.

Experiment #5: Writing to Disk with "W" Command.

As I mentioned previously, writing to the disk can be dangerous. Be careful! Sector 720 should be safe. Why? Well, there is a bug in DOS. DOS can only handle sector numbers from 0 to 719. The disk drive, however, will only accept commands for sectors 1-720. Some software developers have taken advantage of this useful quirk to protect their disks. So don't write to 720 if something was there. If all was OK and you did Experiment #4, hit RETURN. Now type W and hit RETURN again. The screen will turn red and a verify prompt will appear. Answer Y to the prompt if you are sure you want to write to the disk. When the write is complete, the screen will turn green again and we're back to the COMMAND OR SECTOR NUMBER prompt. Recall sector 720 just to check what was written.

Experiment #6: Trace File with "T" Command.

Now that you've traced a file the hard way (if you didn't do Experiment #3, then shame on you), we'll do it using the T command. Call up the directory and pick a file (any file). Note the file number in the FIL# column. Hit T and RETURN. Enter the selected file number (hex or decimal) and hit RETURN. The computer should be busily grinding away, spewing out hexidecimal numbers along with the filename and start sector. When done, the word END should appear. This function shows you exactly which sectors on the disk the file you selected occupies.

Trace will scream if it encounters any file number mismatch errors or short file errors. A short file error means that the length of the file in the directory does not match the number of sectors traced. If this happens for every file you trace, then a possible typing error exists in the ML Loader portion of **Disk Tool.**

Experiment #7: Set Drive Number with "S" Command.

This straightforward command was a heavily requested addition to **Disk Tool**. At the COMMAND OR SECTOR NUMBER prompt, type "S" and hit RETURN. The current working drive number will appear as well as a prompt for the new drive number. Drive numbers 1-4 will be accepted and processed; anything else will produce a RAZZ and an error message. If you change to a drive that does not exist, trying to execute a command will again cause the infamous RAZZ/error message combination.

Experiment #8: Print Screen with "P" Command.

Another straightforward command. If you don't have a printer, you may skip to the next experiment. If you do have a printer, then pick a screen which you would like a hard copy of and answer the COMMAND OR SECTOR NUMBER prompt with a "P" and RETURN. The message PRINTING SCREEN will appear and the screen will be dumped. If you fail to turn on your printer or interface, you will obtain an error message.

You cannot print the HELP screen as the dump routine is only set up to dump Graphics 0. If you try to dump the HELP screen, you will get an IMPROPER SCREEN CONDITION error message.

Experiment #9:
Modify links with "M" Command.

The modify links command is very powerful and one should exercise EXTREME CAUTION in its use. Improper use could cause you to destroy the integrity of a file or files and is guaranteed to make you exclaim that famous all-American expression "Awww Jeepers!" if used incorrectly. Since you have your experiment disk loaded, it won't matter if we mess up a file and then fix it using the Tool.

Find a nice, long file on your experiment disk by scanning the directory. Aha!, there's one. OK, go to the starting sector of the file (indicated under the START column). Manually trace the file for about 4 or 5 sectors and stop. REMEMBER this sector number. Answer the COMMAND OR SECTOR NUMBER prompt with M and RETURN. When the next prompt appears, type in the sector number which you remembered. The sector will be read in and the file number and next sector will be displayed. A prompt asking you for the new file number will appear. Type in a number other than what is displayed but REMEMBER the old file number. Another prompt will appear asking you for the new next sector pointer. Type in a number which is 1 more than the number being displayed but REMEMBER the old number. Boy, we really messed up this file, huh?

A message indicating the new links and a prompt to write the sector to the disk if correct will appear. We will now destroy your disk! No, only kidding. Hit W and RETURN. The screen will turn RED and the verify prompts will appear. Answer Y and write out the sector. Now, if you still remember the file number, hit T and RETURN. Enter the file number at the prompt and watch Trace in action. You should get an error message which indicates a FILE MISMATCH ERROR AT SECTOR $XXX where X is the sector number of the sector which you clobbered. If you didn't, then you probably typed in the wrong file number. OK. Let's fix the error.

Experiment #9A:
Fix error from last experiment.

Hit M and RETURN and recall the sector you clobbered. Change the file number back to what it was. Do NOT fix the next sector number yet. Type in the same number when prompted for the new sector pointer. After all the messages come up, write this sector back out again. Trace the same file. Everything will seem to be fine until the trace realizes that there are some sectors (1 sector in our case) missing. Trace will tell you how many sectors there should be as well as how many it found. The number of "should be" sectors minus the number of "found" sectors should equal the number of missing sectors. Fix the error by recalling that same messed-up sector and replacing the next sector pointer with the original value. Write it back out and re-trace.

Experiment #10:
Recover a deleted file with "R" Command.

Recovering a deleted file is no simple task using manual methods. This was the most requested function to be added to Disk Tool, so here are the steps.

Find a deleted file entry by scanning the directory. Answer the COMMAND OR SECTOR NUMBER prompt with R and RETURN. Answer the next prompt with the file number (hex or decimal) you want to recover and hit RETURN. Disk Tool will now be busy recovering the file. It will keep you informed with messages as it proceeds. Soon you will see the FILENAME.EXT RECOVERED message. Magic, huh? Now, before you go scrambling for those diskettes with deleted files, I must say that there are certain file conditions which must exist or RECOVER will not work — as a matter of fact, NOTHING will work! Let me explain.

Recover file restrictions.

In order for a file to be recovered, it cannot have any sectors which have been written on by other saves. When a file is deleted, DOS sets the file deleted flag in the file status byte of the directory sector where the name resides. It then traces that file to obtain the sector numbers which that file occupied. DOS sets the bits in the VTOC bit map, thus marking the sectors occupied by the file being deleted as now being available. On any subsequent saves to the disk, DOS first searches for an empty file entry in the directory sectors and places the new name and file status in that slot. DOS then examines the VTOC bit map searching for sectors which can be allocated to the new file being saved. If the sectors that it finds available are the same sectors belonging to a previously deleted file, DOS doesn't care and the data belonging to the new file will overlay the deleted filed data. Once this is done, there is no way that the deleted file can be recovered.

Now that the explanation is out of the way (did it make any sense?), let me just say that the recover function of Disk Tool makes extensive checks for file integrity, proper link structure and available sectors. If anything in the file being recovered is goofy, the message FILENAME.EXT CANNOT BE RECOVERED, along with the appropriate reason will be displayed. The recover function will work with both DOS 1 and DOS 2 files, so that some of those oldies but goodies can possibly be rescued from oblivion.

The listings.

Listing 1 — contains the data statements needed to create the AUTORUN.SYS file for Disk Tool.

Listing 2 — is the assembly language source code listing for the AUTORUN.SYS file. This does NOT need to be typed in for Disk Tool to work. The AUTORUN.SYS creator (**Listing 1**) will create the appropriate file. **Listing 2** is

there for reference only. This should give you a pretty good idea of how to reserve some low-end memory, and also how to disable the break key prior to BASIC gaining control of the system.

Listing 3 — is the ML loader program for Disk Tool. This program loads in all of the machine language instructions needed by the Disk Tool BASIC program.

Listing 4 — (the huge one!) is the assembly language source code for Disk Tool utility. In it, you will find how to put a character on the screen, how to convert binary numbers to hex and hex to binary, how to display messages on the screen, how to go crazy trying to read an assembly listing and other common routines. I must say that this code is not the most efficient. Things can be done to improve it, so feel free. I will be glad to answer any questions or comments about it. My address is at the top of the listing (please send a SASE if you write).

Listing 5 — is the Disk Tool BASIC code. I have completely overhauled the code and commented it like a maniac. The documentation following **Listing 5** gives all the addresses, label names and a complete cross-reference to the BASIC code. There is also a memory map which is valid only after Disk Tool has been loaded.

Hints on using Disk Tool.

In these modern times, with DOS 2 being available and all that, it is very rare to come up with link errors and crashed files. Some errors occur, however, when you try to copy DOS 1 files using DOS 2, or you may even run across an old program by some obscure out-of-business company that is loaded with crashed sectors (probably why they are out of business). Whatever the reason, if you have run into Error 164 here is one procedure to follow.

1. Isolate the file causing the problem. It this isn't obvious, call up the directory and trace each file (using the T function) until the culprit is caught. Dump the trace to the printer.

2. Remember the file number. Go to the sector previous to the one in error. This is where some detective skills will pay off. Examine the sectors from your current location to current sector +10, noting which file they belong to. You will probably find your missing sector within this range. I have not failed yet. This usually works on diskettes that have not had too much disk activity; i.e., a lot of file deletions and new file saves. If you run into a toughy, don't give up! You WILL find your missing sector.

3. Once found, note the sector number and the next sector number. Manually trace it to verify the integrity of the file.

4. Call up the original sector which had the incorrect pointer using the M command. Change the pointer to the missing sector and write out the sector using the W command.

This sounds like an involved process, and in some extreme cases it may be, but it sure beats retyping the original file.

Other uses.

CHANGING HEADER BYTES ON ML OBJECT FILES: You have a relocatable ML file which you assembled on page 6. You now want to move it someplace else. The old procedure would be to load in the assembler, load in the source file, change the origin of the file, re-assemble, save the object code. Bah-Humbug to that. With Disk Tool simply call up the directory and find the start of the object file. Call up that sector and change the header information as per **Figure 4.** Re-write the sector and your file will now be loaded at the new address.

The uses for Disk Tool are left to your imagination. It's saved me a lot of time by allowing me direct access to the disk sectors and the information on them. I've patched ML programs directly, added code and allocated new sectors for that code, changed file names that refused to be changed by DOS and recovered many valuable files that were crashed. Let your imagination run wild. □

Listing 1.

```
10 GRAPHICS 2+16
15 ? #6;" ++++++++++++++++++"
20 ? #6;" + ANALOG 400/800 +"
25 ? #6;" +  DSKTOOL.RV3   +"
30 ? #6;" +  autorun.sys   +"
35 ? #6;" + CREATOR PROG.  +"
40 ? #6;" +   for dos ii   +"
45 ? #6;" ++++++++++++++++++"
50 ? #6;"    hit any key to":? #6;" cre
ate AUTORUN.SYS":? #6;"          file"
60 OPEN #1,4,0,"K:"
65 GET #1,A
70 CLOSE #1
75 ? #6;"    creating file"
80 OPEN #1,8,0,"D:AUTORUN.SYS"
85 PUT #1,255:REM HEADER $FF
90 PUT #1,255:REM HEADER $FF
100 PUT #1,0:REM LOAD START LSB $00
105 PUT #1,6:REM LOAD START MSB $06
110 PUT #1,74:REM LOAD END   LSB $4A
115 PUT #1,6:REM LOAD END    MSB $06
120 READ A:IF A=999 THEN GOTO 140
123 REM ** NOW PUT OUT REST OF PROG **
125 PUT #1,A
130 GOTO 120
140 CLOSE #1
160 POSITION 3,10:? #6;" FILE WRITTEN"
170 GOTO 170
1000 DATA 24,173,231,2,105,184,141,231
,2,173
1002 DATA 232,2,105,11,141,232,2,169,0
,133
1004 DATA 8,32,27,6,76,0,160,120,173,2
2
1006 DATA 2,141,60,6,173,23,2,141,61,6
1008 DATA 169,52,141,22,2,169,6,141,23
,2
1010 DATA 88,96,72,173,14,210,16,4,104
,76
1012 DATA 59,6,169,127,141,14,210,165,
16,141
```

```
1014 DATA 14,210,104,64,0,226,2
1016 DATA 227,2,0,6,224,2,225,2,0,6
1018 DATA 999
1020 REM ******************
1022 REM * END AUTORUN.SYS*
1024 REM *   LOADER PROG   *
1026 REM ******************
```

●

CHECKSUM DATA

(See pgs. 7-10)

```
10 DATA 442,342,782,4,723,347,422,348,
971,480,388,504,332,40,762,6887
90 DATA 749,325,357,409,116,197,906,59
4,698,647,891,725,587,482,235,7918
1006 DATA 54,497,359,628,586,702,885,7
89,726,285,801,6312
```

●

Listing 2.

```
0005 ;*******************************
0010 ;* AUTORUN.SYS SOURCE CODE     *
0015 ;* FOR DSKTOOL UTILITY RV3     *
0020 ;* BY: TONY MESSINA            *
0025 ;*     48 DUDLEY AVE           *
0030 ;*     NEWPORT,RI 02840        *
0035 ;*                             *
0040 ;* THIS FILE RESERVES 3000     *
0045 ;* BYTES OF RAM BY MOVING THE *
0050 ;* MEMLO POINTER UP BEFORE     *
0055 ;* THE BASIC OR ASSEMBLER CART*
0060 ;* GETS CONTROL OF THE SYSTEM.*
0065 ;*                             *
0070 ;* THIS CODE ALSO DISABLES THE*
0075 ;* BREAK KEY TO PREVENT ANY    *
0080 ;* POSSIBLE USER ERRORS FROM   *
0085 ;* HAPPENING.                  *
0090 ;
0095 ;* PROGRAM BASED ON IDEAS      *
0100 ;* PRESENTED IN  DE RE ATARI   *
0105 ;* PGS 8-11 & 8-15             *
0110 ;*******************************
0115 ;
0120 ;**************
0125 ;*  EQUATES   *
0130 ;**************
0135 ;
0140 MEMLOL     .DE *02E7           ; LO BYTE MEMLO
0145 MEMLOH     .DE *02E8           ; HI BYTE MEMLO
0150 WARMST     .DE *0008           ; WARSTART FLAG
0155 CARVEC     .DE *A000           ; CART START VECTOR
0160 SAVBYT     .DI 3000            ; # OF BYTES TO RESERVE
0165 POKMSK     .DE *0010           ; POKEY IRQ MASK
0170 IRQEN      .DE *D20E           ; IRQ ENABLE BITS
0175 IRQST      .DE IRQEN           ; IRQ STATUS
0180 VMIRQ      .DE *0216           ; SYSTEM IRQ VECTOR
0185 ;
0190 ;**************
0195 ;*  CONTROL   *
0200 ;**************
0205            .BA *0600           ; ORIGIN *0600
0210            .LS                 ; GIMME LISTING
0215            .OS                 ; OBJ CODE TO MEM
0220 ;
0225 ;**************
0230 ;*  PROGRAM   *
0235 ;**************
0240 ;**** RESERVE 3000 BYTES ****
0245 ;
0250 STRES      CLC                 ; CLEAR FOR ADD
0255            LDA MEMLOL          ; GET CURRENT MEMLO LO BYT
0260            ADC #L,SAVBYT       ; ADD 3000 LO
0265            STA MEMLOL          ; STORE
0270            LDA MEMLOH          ; GET CURRENT MEMLO HI
```

PAGE 02

```
0275            ADC #H,SAVBYT       ; ADD 3000 HI
0280            STA MEMLOH          ; STORE IT
0285            LDA #00             ; WARMSTART RESET
0290            STA *WARMST         ; STORE
0295            JSR SWAPEM          ; DO BREAK KEY STUFF
0300            JMP CARVEC          ; JUMP THROUGH CART
0305 ;
0310 ;*************************
0315 ;* SWAP IRQ VEC ROUTINE *
0320 ;* TO POINT TO OUR OWN  *
0325 ;* ROUTINE. WE WILL IG- *
0330 ;* NORE THE BREAK KEY.  *
0335 ;*************************
0340 ;
0345 ;*** 1ST PUT SYS IRQ IN OUR STUFF ***
0350 ;
0355 SWAPEM     SEI                 ; STOP IRQ'S FOR NOW
0360            LDA VMIRQ           ; GET SYSTEM IRQ LO ADDR
0365            STA SYSIRQ+1        ; MODIFY JMP LO
0370            LDA VMIRQ+1         ; GET SYS IRQ HI ADDR
0375            STA SYSIRQ+2        ; MODIFY JMP HI
0380 ;
0385 ;*** NOW PUT OUR IRQ HANDLER ADDRESS ***
0390 ;*** INTO THE SYSTEM VECTOR LOCATION ***
0395 ;
0400            LDA #L,OURIRQ       ; GET ADDR LO
0405            STA VMIRQ           ; STORE AS SYS VEC
0410            LDA #H,OURIRQ       ; GET ADDR HI
0415            STA VMIRQ+1         ; STORE IT TOO
0420            CLI                 ; NOW ALLOW INTERUPTS
0425            RTS                 ; AND RETURN
0430 ;
0435 ;*************************
0440 ;* THIS IS THE ACTUAL IRQ  *
0445 ;* SERVICE ROUTINE. ALL WE *
0450 ;* DO IS CHECK FOR A BREAK *
0455 ;* KEY. IF BREAK IS HIT, WE*
0460 ;* CAUSE THE SYSTEM TO JUST*
0465 ;* IGNORE IT AND THEN RETURN*
0470 ;*************************
0475 ;
0480 OURIRQ     PHA                 ; SAVE A
0485            LDA IRQST           ; WAS THIS A BREAK??
0490            BPL TISBRK          ; YES IT IS!!
0495            PLA                 ; NO SO PULL A
0500 SYSIRQ     JMP SYSIRQ          ; AND CALL SYSTEM ROUTINE
0505 ;
0510 ;*** BREAK KEY HIT SO SQUASH ***
0515 ;*** THIS MAMA & STOP DOOM!! ***
0520 ;
0525 TISBRK     LDA #$7F            ; WIPE BRK BIT
0530            STA IRQST           ; PUT IN STATUS
0535            LDA *POKMSK         ; GET POKEY MASK
0540            STA IRQEN           ; AND STUFF
0545            PLA                 ; PULL A
0550            RTI                 ; AND RETURN FROM INTERUPT
0555 ;
0560 ;**************
0565 ;* END PROG  *
0570 ;**************
0575            .EN                 ; THE END
```

Listing 3.

```
10 GRAPHICS 2+16:POKE 712,14:POKE 709,
102:POKE 708,202
15 ? #6;"  ++++++++++++++++"
20 ? #6;"  +ANALOG 400/800+"
25 ? #6;"  +   presents   +"
30 ? #6;"  +  utility #3  +"
35 ? #6;"  +  DISK TOOL   +"
40 ? #6;"  ++++++++++++++++"
45 ? #6;"       by"
50 ? #6;"    TONY MESSINA"
55 ? #6;" reading ml program"
60 AREA=7420:REM **ML SAVE AREA **
65 POKE 711,14:READ X:IF X=999 THEN PO
KE 755,2:GOTO 75
70 POKE 711,0:POKE AREA,X:AREA=AREA+1:
GOTO 65
75 ? #6;" loading dsktool.ut1":RUN "D:
DSKTOOL.PT2"
80 DATA 32,83,228,48,51,173,130,29,208
,57
85 DATA 32,133,29,32,38,30,173,22,30,3
2
90 DATA 15,30,32,175,29,162,0,160,0,18
9
95 DATA 253,3,32,243,29,32,222,29,32,2
4
100 DATA 30,32,178,29,32,31,30,200,192
,8
105 DATA 240,17,232,76,25,29,140,126,2
9,32
110 DATA 16,32,160,0,140,22,30,104,96,
142
115 DATA 23,30,138,56,233,7,170,238,23
,30
120 DATA 189,253,3,32,195,29,142,129,2
9,32
125 DATA 57,35,174,129,29,232,236,23,3
0,208
130 DATA 235,169,155,32,57,35,174,23,3
0,224
135 DATA 128,176,202,32,38,30,32,6,30,
32
140 DATA 175,29,160,0,174,23,30,76,25,
29
145 DATA 0,0,0,0,0,0,0,162,0,189
150 DATA 154,29,240,13,142,127,29,32,5
7,35
155 DATA 174,127,29,232,56,176,238,96,
125,66
```

```
160 DATA 89,84,69,35,127,127;72,69,88,
127
165 DATA 127,65,84,65,83,67,73,155,0,3
2
170 DATA 178,29,169,32,32,57,35,96,41,
15
175 DATA 201,10,48,2,105,6,105,48,96,2
01
180 DATA 32,144,20,201,125,144,18,201,
128,144
185 DATA 12,201,155,144,10,201,160,144
,4,201
190 DATA 253,144,2,169,46,96,32,24,30,
173
195 DATA 242,29,32,57,35,173,241,29,32
,57
200 DATA 35,32,31,30,96,0,0,72,74,74
205 DATA 74,74,32,184,29,141,242,29,10
4,32
210 DATA 184,29,141,241,29,96,173,22,3
0,24
215 DATA 105,8,141,22,30,32,243,29,32,
222
220 DATA 29,96,0,0,142,127,29,140,128,
29
225 DATA 96,174,127,29,172,128,29,96,1
69,62
230 DATA 32,57,35,169,36,32,57,35,96,1
04
235 DATA 104,133,206,104,133,205,160,2
,177,205
240 DATA 32,111,30,170,24,105,8,141,12
9,29
245 DATA 200,200,200,177,205,201,32,20
8,5,200
250 DATA 177,205,208,3,32,111,30,157,2
53,3
255 DATA 232,236,129,29,144,231,72,76,
6,29
260 DATA 0,56,233,48,201,10,144,2,233,
7
265 DATA 96,72,200,177,205,32,101,30,1
41,100
270 DATA 30,104,32,101,30,10,10,10,10,
13
275 DATA 100,30,96,76,52,29,32,243,29,
32
280 DATA 222,29,96,32,83,228,48,241,16
9,253
285 DATA 133,205,169,3,133,206,160,5,1
62,0
290 DATA 177,205,157,221,31,200,232,22
4,11,144
295 DATA 245,160,0,177,205,141,236,31,
200,177
300 DATA 205,141,235,31,200,177,205,14
1,234,31
305 DATA 200,177,205,141,233,31,200,17
7,205,141
310 DATA 232,31,44,236,31,16,8,169,68,
141
315 DATA 237,31,76,254,30,80,37,169,85
,141
320 DATA 237,31,169,32,44,236,31,240,5
,169
325 DATA 42,141,239,31,169,2,44,236,31
,240
330 DATA 8,169,50,141,238,31,76,254,30
,169
335 DATA 49,141,238,31,32,43,30,173,11
,3
340 DATA 32,47,35,173,10,3,32,136,30,3
2
345 DATA 178,29,162,0,189,221,31,32,24
,30
350 DATA 32,57,35,32,31,30,232,224,8,1
44
355 DATA 239,32,24,30,32,178,29,32,31,
30
360 DATA 189,221,31,32,24,30,32,57,35,
32
365 DATA 31,30,232,224,11,144,239,32,1
78,29
370 DATA 32,43,30,173,232,31,32,136,30
,173
375 DATA 233,31,32,136,30,32,178,29,32
,43
380 DATA 30,173,234,31,32,136,30,173,2
35,31
385 DATA 32,136,30,32,175,29,32,43,30,
173
390 DATA 243,31,32,136,30,32,175,29,16
2,2
395 DATA 189,237,31,32,24,30,32,57,35,
32
400 DATA 31,30,169,32,157,237,31,202,1
6,236
405 DATA 238,243,31,238,242,31,169,8,2
05,242
410 DATA 31,240,22,165,205,24,105,16,1
33,205
415 DATA 144,2,230,206,169,155,32,57,3
5,32
420 DATA 204,31,76,156,30,169,155,32,5
7,35
425 DATA 173,10,3,24,105,1,141,10,3,14
4
430 DATA 3,238,11,3,162,0,142,242,31,1
73
435 DATA 241,31,208,17,238,241,31,76,1
43,30
440 DATA 162,11,169,32,157,221,31,202,
208,250
445 DATA 96,206,241,31,76,63,29,0,0,0
450 DATA 0,0,0,0,0,0,0,0,0,0
455 DATA 0,0,32,32,32,32,32,0,0,0
460 DATA 32,83,228,16,3,76,52,29,32,16
465 DATA 32,76,63,29,173,132,29,42,42,
13
470 DATA 131,29,141,122,4,76,63,29,173
,122
475 DATA 4,72,41,3,141,131,29,104,74,7
4
480 DATA 141,132,29,96,173,254,34,240,
4,104
485 DATA 76,189,32,104,104,141,11,3,14
1,119
490 DATA 35,104,141,10,3,141,118,35,10
4,133
495 DATA 206,104,133,205,104,104,141,2
43,31,32
500 DATA 83,228,16,3,76,221,34,162,11,
160
505 DATA 15,177,205,157,220,31,136,202
,208,247
510 DATA 177,205,141,11,3,136,177,205,
141,10
515 DATA 3,136,177,205,141,235,31,136,
177,205
520 DATA 141,234,31,136,173,117,35,240
,3,76
525 DATA 252,35,177,205,240,20,141,236
,31,44
530 DATA 236,31,16,28,32,212,33,142,12
6,29
535 DATA 32,213,34,76,64,29,162,34,160
,111
540 DATA 32,159,33,173,243,31,32,136,3
0,76
545 DATA 42,33,162,33,160,232,32,159,3
3,32
550 DATA 212,33,162,33,160,239,32,159,
33,173
555 DATA 11,3,32,47,35,173,10,3,32,136
560 DATA 30,160,16,140,249,34,169,155,
32,57
565 DATA 35,162,7,142,248,34,32,83,228
,16
570 DATA 3,76,221,34,32,16,32,238,255,
34
575 DATA 208,3,238,0,35,173,132,29,205
,243
580 DATA 31,208,35,173,123,4,13,131,29
,240
585 DATA 81,32,199,33,32,1,35,206,248,
34
590 DATA 16,210,169,1,141,46,35,206,24
9,34
595 DATA 16,190,141,254,34,76,64,29,32
,204
600 DATA 31,162,34,160,20,32,159,33,32
,43
605 DATA 30,173,11,3,32,47,35,173,10,3
```

```
610 DATA 32,136,30,162,34,160,54,32,15
9,33
615 DATA 169,1,141,126,29,141,46,35,16
9,0
620 DATA 141,255,34,141,254,34,141,0,3
5,76
625 DATA 64,29,173,234,31,77,255,34,20
8,31
630 DATA 173,235,31,77,0,35,208,23,141
,255
635 DATA 34,141,0,35,162,34,160,134,32
,159
640 DATA 33,169,155,32,57,35,169,1,76,
47
645 DATA 33,162,34,160,139,32,159,33,1
73,235
650 DATA 31,32,136,30,173,234,31,32,13
6,30
655 DATA 162,34,160,165,32,159,33,173,
0,35
660 DATA 32,136,30,173,255,34,32,136,3
0,162
665 DATA 34,160,191,32,159,33,169,0,14
1,255
670 DATA 34,141,0,35,76,42,33,173,136,
29
675 DATA 141,250,34,173,137,29,141,251
,34,140
680 DATA 136,29,142,137,29,32,133,29,1
73,250
685 DATA 34,141,136,29,173,251,34,141,
137,29
690 DATA 174,252,34,172,253,34,96,173,
131,29
695 DATA 141,11,3,173,123,4,141,10,3,9
6
700 DATA 162,0,189,221,31,32,24,30,32,
57
705 DATA 35,32,31,30,232,224,11,144,23
9,96
710 DATA 70,73,76,69,58,32,0,32,32,32
715 DATA 32,32,83,84,65,82,84,32,83,69
720 DATA 67,84,79,82,58,36,0,32,73,83
725 DATA 32,68,69,76,69,84,69,68,33,33
730 DATA 253,253,155,0,155,70,73,76,69
,32
735 DATA 78,85,77,66,69,82,32,77,73,83
740 DATA 77,65,84,67,72,32,65,84,32,83
745 DATA 69,67,84,79,82,27,31,0,155,67
750 DATA 72,69,67,75,32,80,82,69,86,73
755 DATA 79,85,83,32,83,69,67,84,79,82
760 DATA 32,76,73,78,75,83,33,33,253,2
53
765 DATA 0,67,65,78,78,79,84,32,82,69
770 DATA 65,68,32,83,69,67,84,79,82,58
775 DATA 27,31,36,253,0,155,78,79,32,6
9
780 DATA 78,84,82,89,32,70,79,82,32,70
785 DATA 73,76,69,27,31,36,253,0,32,69
790 DATA 78,68,0,155,79,82,73,71,73,78
795 DATA 65,76,32,83,69,67,84,79,82,32
800 DATA 67,79,85,78,84,27,31,36,0,155
805 DATA 65,67,84,85,65,76,32,83,69,67
810 DATA 84,79,82,83,32,76,79,65,68,69
815 DATA 68,27,31,36,0,155,83,72,79,82
820 DATA 84,32,70,73,76,69,32,69,82,82
825 DATA 79,82,33,33,253,155,0,162,34,
160
830 DATA 3,32,159,33,96,162,34,160,87,
32
835 DATA 159,33,173,11,3,32,47,35,173,
10
840 DATA 3,32,136,30,169,155,32,57,35,
76
845 DATA 42,33,0,0,0,0,0,0,0,0
850 DATA 0,32,24,30,173,46,35,240,6,32
855 DATA 43,30,76,28,35,169,27,32,57,3
5
860 DATA 169,31,32,57,35,32,43,30,173,
131
865 DATA 29,32,47,35,173,123,4,32,136,
30
870 DATA 169,0,141,46,35,96,1,32,243,2
9
875 DATA 32,231,29,96,69,58,155,162,64
,32
880 DATA 86,228,96,162,64,169,12,157,6
6,3
885 DATA 32,86,228,162,64,169,3,157,66
,3
890 DATA 169,54,157,68,3,169,35,157,69
,3
895 DATA 169,8,157,74,3,32,86,228,162,
64
900 DATA 169,11,157,66,3,169,0,157,72,
3
905 DATA 157,73,3,104,96,0,0,0,0,0
910 DATA 0,0,0,0,0,0,0,0,0,0
915 DATA 0,0,0,0,0,0,0,0,0,0
920 DATA 0,0,0,0,0,0,0,0,0,0
925 DATA 0,0,0,0,0,0,0,0,0,0
930 DATA 0,0,0,0,0,0,0,0,0,0
935 DATA 0,0,0,0,0,0,0,0,0,0
940 DATA 0,0,0,0,0,0,0,0,0,0
945 DATA 0,0,0,0,0,0,0,0,0,0
950 DATA 0,0,0,0,0,0,0,0,0,0
955 DATA 0,0,0,0,0,0,0,0,0,0
960 DATA 0,0,0,0,0,0,0,0,0,0
965 DATA 0,0,0,0,0,0,0,0,0,0
970 DATA 0,0,0,0,0,0,0,0,0,0
975 DATA 0,0,140,253,34,173,11,3,141,1
16
980 DATA 35,173,10,3,141,115,35,162,35
,160
985 DATA 120,142,5,3,140,4,3,162,104,1
60
990 DATA 1,140,11,3,142,10,3,32,83,228
995 DATA 16,13,162,38,160,251,32,159,3
3,32
1000 DATA 204,31,76,221,34,32,218,37,1
72,253
1005 DATA 34,177,205,208,3,76,144,32,1
41,236
1010 DATA 31,44,236,31,16,73,162,38,16
0,34
1015 DATA 32,159,33,173,116,35,141,11,
3,173
1020 DATA 115,35,141,10,3,32,83,228,16
,16
1025 DATA 32,212,33,32,204,31,162,39,1
60,19
1030 DATA 32,159,33,76,221,34,32,16,32
,238
1035 DATA 255,34,208,3,238,0,35,173,13
2,29
1040 DATA 205,243,31,208,27,173,123,4,
13,131
1045 DATA 29,240,84,32,199,33,76,89,36
,32
1050 DATA 212,33,162,38,160,68,32,159,
33,76
1055 DATA 42,33,162,34,160,20,32,159,3
3,32
1060 DATA 43,30,173,11,3,32,47,35,173,
10
1065 DATA 3,32,136,30,169,155,32,57,35
,162
1070 DATA 33,160,232,32,159,33,32,212,
33,162
1075 DATA 38,160,88,32,159,33,32,204,3
1,240
1080 DATA 204,32,212,33,32,204,31,162,
38,160
1085 DATA 88,32,159,33,76,103,33,173,2
34,31
1090 DATA 77,255,34,208,232,173,235,31
,77,0
1095 DATA 35,208,224,141,255,34,141,0,
35,162
1100 DATA 38,160,114,32,159,33,173,118
,35,141
1105 DATA 10,3,173,119,35,141,11,3,32,
83
1110 DATA 228,16,13,162,39,160,38,32,1
59,33
1115 DATA 32,204,31,76,42,33,160,0,173
,120
1120 DATA 35,201,2,208,4,169,66,208,2,
169
1125 DATA 64,145,205,169,87,141,2,3,32
,83
1130 DATA 228,16,35,32,218,37,32,204,3
1,162
```

```
1135 DATA 39,160,62,32,159,33,173,11,3
,32
1140 DATA 47,35,173,10,3,32,136,30,162
,39
1145 DATA 160,86,32,159,33,76,42,33,32
,218
1150 DATA 37,162,38,160,152,32,159,33,
162,38
1155 DATA 160,174,32,159,33,173,115,35
,141,10
1160 DATA 3,173,116,35,141,11,3,32,83,
228
1165 DATA 16,13,162,39,160,19,32,159,3
3,32
1170 DATA 204,31,76,221,34,32,234,37,3
2,16
1175 DATA 32,173,123,4,13,131,29,240,6
,32
1180 DATA 199,33,76,115,37,162,35,160,
120,142
1185 DATA 5,3,140,4,3,162,104,160,1,14
2
1190 DATA 10,3,140,11,3,169,87,141,2,3
1195 DATA 32,83,228,16,13,162,38,160,2
27,32
1200 DATA 159,33,32,218,37,76,42,33,32
,218
1205 DATA 37,32,212,33,162,38,160,204,
32,159
1210 DATA 33,32,204,31,142,117,35,76,6
4,29
1215 DATA 162,3,160,253,142,5,3,140,4,
3
1220 DATA 162,82,142,2,3,96,169,0,160,
3
1225 DATA 78,11,3,110,10,3,106,136,208
,246
1230 DATA 160,5,106,136,208,252,168,16
9,0,56
1235 DATA 106,136,16,252,72,173,10,3,1
05,10
1240 DATA 168,104,89,120,35,153,120,35
,206,123
1245 DATA 35,173,123,35,201,255,208,3,
206,124
1250 DATA 35,96,127,80,65,83,83,49,32,
45
1255 DATA 32,67,72,69,67,75,73,78,71,3
2
1260 DATA 70,73,76,69,32,67,79,78,68,7
3
1265 DATA 84,73,79,78,155,0,32,73,83,3
2
1270 DATA 78,79,84,32,68,69,76,69,84,6
9
1275 DATA 68,33,33,155,253,0,44,32,67,
65
1280 DATA 78,78,79,84,32,66,69,32,82,6
9
1285 DATA 67,79,86,69,82,69,68,33,33,2
53
1290 DATA 155,0,70,73,76,69,32,73,78,8
4
1295 DATA 65,67,84,155,127,80,65,83,83
,50
1300 DATA 32,45,32,82,69,67,79,86,69,8
2
1305 DATA 73,78,71,32,70,73,76,69,155,
0
1310 DATA 68,73,82,69,67,84,79,82,89,3
2
1315 DATA 69,78,84,82,89,32,68,79,78,6
9
1320 DATA 155,0,82,69,65,76,76,79,67,6
5
1325 DATA 84,73,78,71,32,68,69,76,69,8
4
1330 DATA 69,68,32,83,69,67,84,79,82,8
3
1335 DATA 155,0,32,72,65,83,32,66,69,6
9
1340 DATA 78,32,82,69,67,79,86,69,82,6
9
1345 DATA 68,33,253,155,0,69,82,82,79,
82
1350 DATA 32,73,78,32,86,84,79,67,32,8
7
1355 DATA 82,73,84,69,33,33,253,155,0,
69
1360 DATA 82,82,79,82,32,73,78,32,86,8
4
1365 DATA 79,67,32,82,69,65,68,32,33,3
3
1370 DATA 253,155,0,70,73,76,69,32,82,
69
1375 DATA 65,68,32,69,82,82,79,82,33,2
53
1380 DATA 155,0,68,73,82,69,67,84,79,8
2
1385 DATA 89,32,82,69,65,68,32,69,82,8
2
1390 DATA 79,82,33,253,155,0,68,73,82,
69
1395 DATA 67,84,79,82,89,32,87,82,73,8
4
1400 DATA 69,32,69,82,82,79,82,33,27,3
1
1405 DATA 253,155,0,0,0,999
```

•

CHECKSUM DATA

(See pgs. 7-10)

```
10 DATA 601,859,577,173,880,415,850,35
0,194,519,564,841,903,755,385,8866
85 DATA 282,260,264,536,556,358,379,58
8,885,892,537,507,433,580,758,7815
160 DATA 489,349,537,548,888,644,367,4
44,23,629,624,352,589,908,609,8000
235 DATA 128,561,845,578,617,261,677,4
32,564,898,576,144,882,99,938,8200
310 DATA 357,618,605,607,626,532,321,3
90,571,519,493,851,600,415,841,8346
385 DATA 371,405,520,844,677,652,617,6
09,278,533,853,894,224,256,637,8370
460 DATA 169,591,579,294,844,885,823,1
06,379,157,678,908,831,694,564,8502
535 DATA 588,615,608,668,62,871,436,57
3,621,597,599,602,695,379,113,8027
610 DATA 592,419,577,640,578,619,573,7
22,595,606,865,893,617,898,718,9912
685 DATA 744,741,285,497,611,301,219,3
45,283,469,285,240,246,263,298,5827
760 DATA 677,264,299,389,251,231,264,2
84,230,275,289,211,232,420,581,4897
835 DATA 524,546,336,99,383,396,598,34
1,466,480,643,632,647,271,936,7298
910 DATA 254,259,257,262,260,265,263,2
68,266,271,269,274,272,549,633,4622
985 DATA 535,68,639,595,641,524,747,30
4,755,539,537,580,410,799,518,8191
1060 DATA 435,593,594,789,581,802,819,
608,840,460,797,480,322,360,779,9259
1135 DATA 332,300,577,631,844,307,546,
511,335,835,214,15,816,574,639,7476
1210 DATA 521,199,194,530,640,771,92,8
38,478,327,339,303,356,485,339,6412
1285 DATA 590,303,432,322,301,349,369,
316,353,349,292,354,503,327,550,5710
1360 DATA 339,311,471,584,326,346,518,
373,303,82,3653
```

•

Listing 4.

```
0006 ;*********************************
0008 ;* THIS FILE IS THE CONTROL    *
0010 ;* FILE USED TO ASSEMBLE ALL   *
0012 ;* PARTS OF THE ML PORTION OF  *
0014 ;* THE DISK TOOL UTILITY. IT   *
0016 ;* IS EXECUTED BY TYPING THE   *
0018 ;*      ASM "D:DMPUNIV.CTL"    *
0020 ;* COMMAND FROM THE MAE EDITOR*
0022 ;*********************************
0024            .CT
0025            .LS
0026            .FI "D:DMPUNIV.SRC"


 D:DMPUNIV.SRC

0002 ;*********************************
0004 ;*     * DSKTOOL MLLIST *      *
0006 ;*     ******************      *
0008 ;*     *   8 AUG 1981   *      *
0010 ;*     *REV6 15 JUL 1983*      *
0012 ;*     *  TONY MESSINA  *      *
0014 ;*     * 48 DUDLEY AVE  *      *
0016 ;*     *   NEWPORT,RI   *      *
0018 ;*     *      02840     *      *
0020 ;*********************************
0022 ;
0024 ;***************
0026 ;*  EQUATES   *
0028 ;***************
0030 ;
0032 DSKVEC     .DE $E453           ; READ/WRITE DISK
0034 CASBUF     .DI $03FD           ; 128 BYTE BUFFER
0036 PERIOD     .DI $2E             ; ASCII PERIOD
0038 CR         .DI $9B             ; CARRIGE RETURN
0040 SP         .DI $20             ; ASCII SPACE
0042 CLS        .DI $7D             ; CLEAR SCREEN
0044 TAB        .DI $7F             ; TAB SPACE
0046 NULL       .DI $00             ; END OF TEXT DELIMETER
0048 PAGE0      .DI $00CD           ; PG 0 WORK LOCATION
0050 LOCKED     .DI $20             ; FILE LOCKED MASK
0052 DOSMSK     .DI $02             ; DOS 2 MASK
0054 DAUX1      .DE $030A
0056 DAUX2      .DE $030B
0058 ESC        .DI $001B           ; ESC/ESC SEQ
0060 BELL       .DI $FD             ; RING BUZZER
0062 RAR        .DI $001F           ; RIGHT ARROW
0064 ;
0066 ;***************
0068 ;*  CONTROL   *
0070 ;***************
0072 ;* NOTE:      *
0074 ;* ORG DOS2= *
0076 ;*   $1CFC    ********************
0078 ;* ANY OTHER ORG WILL REQUIRE *
0080 ;* CHANGES TO THE BASIC PART  *
0082 ;* OF THIS UTILITY..BEWARE!!!!*
0084 ;* RUN ONLY UNDER UNMODIFIED  *
0086 ;* DOS2 (I.E. DOS2 THAT DOES  *
0088 ;* NOT SAVE SPACE BY DROPPING *
0090 ;* NON-EXISTING DRIVES!!!     *
0092 ;*********************************
0094 ;
0096            .PR "ORIGIN OF HEXDMP" ; ASK ORG
0098 ASTART     .IN ASTART          ; USER INPUT
0100            .BA ASTART          ; ASSIGN ORG
0102            .OS                 ; STORE OBJ CODE IN MEM
0104            .MC $A900           ; BUT PUT AT $A900
0106            .PR "ORIGIN AT INPUT"
0108            .PR "OBJ STORE AT $A900"
0110 ;
0112 ;***************
0114 ;*  PROGRAM   *
0116 ;***************
0118 ;
0120 START      JSR DSKVEC          ; GO READ OR WRITE
0122            BMI DERR            ; IF ERROR BRANCH
0124            LDA WFLAG           ; WAS IT A WRITE?
0126            BNE EXIT2           ; YES..KEEP DISPLAY!
0128 MESAGE     JSR MSG             ; PUT UP HEADER
0130            JSR PREFIX          ; PUT UP >0
0132            LDA LOADR           ; GET LOADR
0134            JSR AROUND          ; PUT UP 00
0136            JSR SPACE2          ; 2 SPACES
0138 ;*** SECTOR IS IN, HEADER UP, DUMP HEX ***
0140 ;
0142            LDX #0              ; BYTE COUNTER
0144            LDY #0              ; COUNT 8 HEX BYTES
0146 DSPHEX     LDA CASBUF,X        ; GET A BYTE
0148            JSR CONVERT         ; BREAK INTO NYBLES
0150            JSR DISPLY          ; AND DISPLAY
0152            JSR SAVXY           ; SAVE X&Y REGS
0154            JSR SPACE1          ; NOW SP 1
0156            JSR RESXY           ; RESTORE X&Y REGS
0158            INY                 ; INC COUNT
0160            CPY #8              ; TIME TO DUMP ASCII??
0162            BEQ DMPASC          ; YES..GO DO IT
0164            INX                 ; NO..INC BYTE COUNT
0166            JMP DSPHEX          ; NO..NEXT??
0168 ;
0170 ;***************
0172 ;* DSK ERRORS *
0174 ;***************
0176 ;
0178 DERR       STY ERRFLG          ; INFORM BASIC OF ERROR
0180 EXIT       JSR WEIRD           ; BREAK UP WEIRD BYTE
0182            LDY #0              ; ZERO Y
0184            STY LOADR           ; ZERO LOADR
0186 EXIT2      PLA                 ; CLEAN UP STACK
0188            RTS                 ; AND RETURN TO BASIC
0190 ;
0192 ;***************
0194 ;* ASCII DUMP *
0196 ;***************
0198 ;
0200 DMPASC     STX HIADR           ; SAVE COUNT
0202            TXA                 ; GET THIS COUNT
0204            SEC
0206            SBC #7              ; SET START FOR DUMP
0208            TAX                 ; PUT IN X
0210            INC HIADR           ; INC FOR COMPARE
0212 GETIT      LDA CASBUF,X        ; GET WHOLE BYTE
0214            JSR CKDOOM          ; STOP TROUBLE
0216            STX CHRCNT          ; SAVE THIS COUNT
0218            JSR PUTCHR          ; CHARACTER TO SCREEN
0220            LDX CHRCNT          ; RESTORE X
0222            INX                 ; AND INCREMENT
0224            CPX HIADR           ; LIMIT REACHED?
0226            BNE GETIT           ; NO..NEXT CHAR
0228            LDA #CR             ; YES..SO
0230            JSR PUTCHR          ; SKIP A LINE
0232            LDX HIADR           ; CK LIMITS
0234            CPX #128            ; DONE SECTOR?
0236            BCS EXIT            ; YES..GOODBYE
0238            JSR PREFIX          ; PUT UP >0
0240            JSR UPDATE          ; UPDATE BYTE CNT
0242            JSR SPACE2          ; SKIP 2 SPACES
0244            LDY #0              ; ZERO Y COUNTER
0246            LDX HIADR           ; GET OLD X COUNT
0248            JMP DSPHEX          ; AND GET MO STUFF
0250 ;
0252 ;***************
0254 ;* VARIABLES  *
0256 ;***************
0258 ;
0260 ERRFLG     .DS 1               ; ERROR FLAG
0262 SAVEX      .DS 1               ; X SAVE
0264 SAVEY      .DS 1               ; Y SAVE
0266 CHRCNT     .DS 1               ; STOREAGE
0268 WFLAG      .DS 1               ; WRITE FLAG
0270 TOPSEC     .DS 1
0272 FILNUM     .DS 1
0274 ;
0276 ;***************
0278 ;* SUBROUTINES*
0280 ;***************
0282 ;
0284 ;*** WRITE HEADER ***
0286 ;***  SUBROUTINE  ***
0288 ;
0290 MSG        LDX #0              ; START AT ZERO
0292 DISMSG     LDA HEADER,X        ; GET BYTE
0294            BEQ ENDMSG          ; IF ZERO SCRAM
0296            STX SAVEX           ; SAVEX
0298            JSR PUTCHR          ; DISPLAY CHAR
0300            LDX SAVEX           ; RESTORE X
0302            INX                 ; ADD 1
0304            SEC
0306            BCS DISMSG          ; ALWAYS BRANCH!!
0308 ENDMSG     RTS                 ; ADIOS!!
0310 HEADER     .BY CLS 'BYTE#'
0312            .BY TAB TAB 'HEX' TAB TAB 'ATASCI' CR NULL
0316 ;*** WRITE SPACES ***
0318 ;***  SUBROUTINE  ***
0320 ;
0322 SPACE2     JSR SPACE1          ; GO HERE FOR 2 SPACES
0324 SPACE1     LDA #SP             ; LOAD A SPACE
0326            JSR PUTCHR          ; AND DISPLAY
0328            RTS                 ; THEN RETURN
0330 ;
0332 ;*** CONVERT ASCI ***
0334 ;***  SUBROUTINE  ***
0336 ;
0338 CONASC     AND #$0F            ; CLEAR TOP NYBLE
0340            CMP #$0A            ; IS A RE0>9??
0342            BMI LT9             ; NO..ONLY ADD $30
0344            ADC #6              ; YES.ADD 6
0346 LT9        ADC #$30            ; ADD $30
0348            RTS                 ; AND RETURN
0350 ;
0352 ;*** CHECK GARBAGE ***
0354 ;***  SUBROUTINE   ***
0356 ;
0358 CKDOOM     CMP #$20            ; A < SPACE?
0360            BCC SUBPER          ; YES SUB PERIOD
0362            CMP #$7D            ; NO < 7D??
0364            BCC OUT             ; YES PRINT
0366            CMP #$80            ; NO..HOW ABOUT <$80
0368            BCC SUBPER          ; YES..SUB PERIOD
0370            CMP #CR             ; < $9B ??
0372            BCC OUT             ; YES..SCRAM
0374            CMP #$A0            ; NO..< $A0??
0376            BCC SUBPER          ; YES..SUB PERIOD
0378            CMP #$FD            ; NO..HOW BOUT $FD
0380            BCC OUT             ; YES..ELSE
0382 SUBPER     LDA #PERIOD         ; LOAD A PERIOD
0384 OUT        RTS                 ; THEN RETURN
0386 ;
0388 ;*** DISPLAY HEX BYTES ***
0390 ;***      SUBROUTINE     ***
0392 ;
0394 DISPLY     JSR SAVXY           ; SAVE X& Y REGS
0396            LDA HIHEX           ; GET HI BYTE
0398            JSR PUTCHR          ; PUT ON SCREEN
0400 ONEBYE     LDA LOHEX           ; GET LO BYTE
0402            JSR PUTCHR          ; AND DISPLAY IT TOO
0404            JSR RESXY           ; RESTORE X&Y REGS
0406            RTS                 ; THEN RETURN
0408 LOHEX      .DS 1               ; LO HEX STORE
0410 HIHEX      .DS 1               ; HI HEX STORE
0412 ;
0414 ;*** CONVERT BIN BYTE TO 2 HEX DIGITS ***
0416 ;***      SUBROUTINE ***
0418 ;
0420 CONVERT    PHA                 ; 1ST SAVE A
0422            LSR A               ; EXTRACT
0424            LSR A               ; TOP NYBLE
0426            LSR A               ; AND SHIFT TO
0428            LSR A               ; BOTTOM NYBLE
0430            JSR CONASC          ; CONVERT BIN TO ASCII
0432            STA HIHEX           ; STORE IT
```

```
0434            PLA                     ; GET ORIGINAL BYTE
0436            JSR CONASC              ; CONVERT IT
0438            STA LOHEX               ; STORE IT
0440            RTS                     ; AND RETURN
0442 ;
0444 ;*** UPDATE BYTE COUNT ***
0446 ;***    SUBROUTINE     ***
0448 ;
0450 UPDATE     LDA LOADR               ; GET LO BYTE
0452            CLC                     ; CLEAR CARRY
0454            ADC #$08                ; ADD 8
0456            STA LOADR               ; STORE AWAY
0458 AROUND     JSR CONVERT             ; CONVERT IT
0460            JSR DISPLY              ; AND DISPLAY
0462            RTS                     ; THEN RETURN
0464 LOADR      .DS 1                   ; BYTE COUNT LO
0466 HIADR      .DS 1                   ; BYTE COUNT HI
0468 ;
0470 ;
0472 ;*** SAVE X&Y REGISTERS ***
0474 ;***    SUBROUTINE     ***
0476 ;
0478 SAVXY      STX SAVEX
0480            STY SAVEY
0482            RTS
0484 RESXY      LDX SAVEX
0486            LDY SAVEY
0488            RTS
0490 ;
0492 ;*** PREFIX BYTES WITH >0 ***
0494 ;***      SUBROUTINE     ***
0496 ;
0498 PREFIX     LDA #'>                 ; LOAD A CARAT
0500            JSR PUTCHR              ; DISPLAY IT
0502 JUSHEX     LDA #'$                 ; LOAD HEX DESIGNATOR
0504            JSR PUTCHR              ; DISPLAY IT
0506            RTS                     ; RETURN
0508 ;
0510 ;*** ML CHANGE BYTE ROUTINE ***
0512 ;*** BASIC ENTERS HERE      ***
0514 CHNGBY     PLA                     ; PULL OFF # VARS PASSED
0516            PLA                     ; PULL OFF HI ADR
0518            STA *PAGE0+1            ; STUFF IT
0520            PLA                     ; NEXT??
0522            STA *PAGE0              ; STUFF IT
0524 INDEX      LDY #2                  ; SKIP >0
0526            LDA (PAGE0),Y           ; HI HEX
0528            JSR MAKBIN              ; HEX ASCI TO BIN BYTE
0530            TAX                     ; SAVE START INDEX
0532            CLC
0534            ADC #$8                 ; GET MAX COUNT
0536            STA CHRCNT              ; STORE IT
0538            INY
0540 HX2BIN     INY                     ; SKIP TO
0542            INY                     ; NXT USEABL BYTE
0544            LDA (PAGE0),Y           ; HI CHAR
0546            CMP #$P                 ; IS IT SPACE?
0548            BNE NOTASC              ; NO..HEX
0550            INY                     ; YES..GET CHAR
0552            LDA (PAGE0),Y           ; IN A
0554            BNE STUFIT              ; AND STORE DIRECTLY
0556 NOTASC     JSR MAKBIN              ; HEX ASCII TO BIN BYTE
0558 STUFIT     STA CASBUF,X            ; STUFF IN BUFFER
0560            INX
0562            CPX CHRCNT              ; DONE 8 BYTES
0564            BCC HX2BIN              ; NO.GET NXT
0566            PHA                     ; PUSH FOR EXIT
0568            JMP MESAGE              ; PUT UP NEW SCREEN
0570 TEMP       .DS 1                   ; LO BIN VAL STORE
0572 ;*** ASCI HEX TO BIN ***
0574 ;***   SUBROUTINE    ***
0576 ;
0578 AS2BIN     SEC                     ; SUBTRACT
0580            SBC #'0                 ; ASCII 0
0582            CMP #10                 ; A<10?
0584            BCC ASBIN1              ; YES JMP
0586            SBC #7                  ; ELSE SUB 7 MORE
0588 ASBIN1     RTS                     ; AND RETURN
0590 ;
0592 ;*********************************
0594 ;* CONVERT 2 ASCII HEX DIGITS    *
0596 ;* TO A BINARY #. THE HI DIGIT   *
0598 ;* COMES IN THE A REG. THE LO    *
0600 ;* DIGIT IS EXTRACTED FROM THE   *
0602 ;* PAGE 0 PTR + Y REG. ROUTINE   *
0604 ;* EXITS WITH BIN NUMBER IN A    *
0606 ;*********************************
0608 ;
0610 MAKBIN     PHA                     ; SAVE HI HEX DIGIT
0612            INY
0614            LDA (PAGE0),Y           ; GET LO HEX DIG
0616            JSR AS2BIN              ; CONVERT
0618            STA TEMP                ; STORE
0620            PLA                     ; GET BACK HI
0622            JSR AS2BIN              ; CONVERT
0624            ASL A                   ; SHIFT
0626            ASL A                   ; IT
0628            ASL A                   ; UP
0630            ASL A                   ; TOP
0632            ORA TEMP                ; OR IN LO BYTE
0634            RTS                     ; RETURN WITH BIN IN A
0636 ;*** DIRECTORY DUMP STARTS HERE ***
0638 JPONT      JMP DERR                ; LONG BRANCH
0640 DOBOTH     JSR CONVERT             ; CON VAL IN A TO HEX
0642            JSR DISPLY              ; AND DISPLAY
0644            RTS
0646 ;
0648 REDIR      JSR DSKVEC              ; READ SECTOR
0650            BMI JPONT               ; JUMP LONG ON ERROR
0652            LDA #L,CASBUF           ; LO BUF START
0654            STA *PAGE0              ; STUFF
0656            LDA #H,CASBUF           ; HI START
0658            STA *PAGE0+1            ; STUFF
0660 FILOOP     LDY #5                  ; NAME START
0662            LDX #0                  ; BUF INDEX
0664 ;*** NOW GET FILE NAME ***
0720            LDA #'U                 ; USED LOAD 'U'
0722            STA FILSTA+1            ; STUF
0724 CKLOCK     LDA #LOCKED             ; LOCK MASK
0726            BIT FILSTA              ; IS IT?
0728            BEQ CKDOS               ; NO CK DOS
0730            LDA #'*                 ; YES. LOAD '*'
0732            STA FILSTA+3            ; STUF
0734 CKDOS      LDA #DOSMSK             ; DOS MASK
0736            BIT FILSTA              ; DOS?
0738            BEQ DOS1                ; NO DOS1
0740            LDA #'2                 ; YES LOAD 2
0742            STA FILSTA+2            ; STUF
0744            JMP OUTFIL              ; AND OUT
0746 DOS1       LDA #'1                 ; LOAD 1
0748            STA FILSTA+2            ; AND STUF
0750 ;*** FILE FORMATTED..DUMP IT ***
0752 OUTFIL     JSR JUSHEX              ; PUT UP A $
0754            LDA DAUX2               ; GET SEC HI
0756 PUTSEC     JSR DOONE               ; DISPLAY LO NYBLE
0758            LDA DAUX1               ; DSK SEC LO
0760            JSR DOBOTH              ; DISPLAY
0762            JSR SPACE1              ; SKIP SPACE
0764            LDX #0                  ; SET INDEX
0766 DISPFL     LDA NAMBUF,X            ; GET LTR
0768            JSR SAVXY               ; SAVE IDX'S
0770            JSR PUTCHR              ; PUT ON SCRN
0772            JSR RESXY               ; RESTORE X/Y
0774            INX
0776            CPX #8                  ; DONE NAME?
0778            BCC DISPFL              ; NO.GET MO
0780            JSR SAVXY               ; SAVE X&Y
0782            JSR SPACE1              ; 1 SPACE
0784            JSR RESXY               ; RESTORE X&Y
0786 DOEXT      LDA NAMBUF,X            ; GET EXTENSION
0788            JSR SAVXY               ; SAVE EM
0790            JSR PUTCHR              ; DISPLAY
0792            JSR RESXY               ; RESTORE
0794            INX                     ; INC COUNT
0796            CPX #11                 ; GOT EXTENSION?
0798            BCC DOEXT               ; NO..GET ALL
0800            JSR SPACE1              ; PUT UP SP
0802            JSR JUSHEX              ; AND $
0804            LDA STASEC              ; LOAD START
0806 PUTSTA     JSR DOBOTH              ; TO HEX
0808            LDA STASEC+1            ; START LO
0810            JSR DOBOTH              ; DISPLAY
0812            JSR SPACE1              ; SKIP 3
0814            JSR JUSHEX              ; PUT UP $
0816            LDA FILEN               ; FILE LEN HI
0818 PUTLEN     JSR DOBOTH              ; DISPLAY
0820            LDA FILEN+1             ; FILE LEN LO
0822            JSR DOBOTH              ; DISPLAY
0824            JSR SPACE2              ; SKIP 3
0826            JSR JUSHEX              ; PUT UP $
0828            LDA FINUMB              ; FILE NUMBER
0830            JSR DOBOTH              ; DISPLAY
0832            JSR SPACE2              ; SKIP 2
0834 PSTATU     LDX #2                  ; CNT 2
0666 GETNAM     LDA (PAGE0),Y           ; GET CHAR
0668            STA NAMBUF,X            ; STUFF
0670            INY                     ; INC THE
0672            INX                     ; COUNTERS
0674            CPX #11                 ; DONE WITH NAME?
0676            BCC GETNAM              ; NO.GET MORE
0678            LDY #0                  ; YES
0680 GETSTA     LDA (PAGE0),Y           ; GET STATUS BYTE
0682            STA FILSTA              ; STORE
0684            INY                     ; NEXT BYTE
0686 GETLEN     LDA (PAGE0),Y           ; LEN LO
0688            STA FILEN+1             ; STORE
0690            INY
0692            LDA (PAGE0),Y           ; LEN HI
0694            STA FILEN               ; STORE
0696            INY
0698 FISTAR     LDA (PAGE0),Y           ; GET FI START
0700            STA STASEC+1            ; STUF
0702            INY
0704            LDA (PAGE0),Y           ; START HI
0706            STA STASEC              ; STUF
0708 CKSTA      BIT FILSTA              ; ENTRY DELETED?
0710            BPL CKUSED              ; NO. CK IF USED
0712            LDA #'D                 ; DELETED LOAD 'D'
0714            STA FILSTA+1            ; STUF
0716            JMP OUTFIL              ; THEN OUT
0718 CKUSED     BVC OUTFIL              ; IF NOT USED, OUT
0836 PUTLTR     LDA FILSTA+1,X          ; PUT UP STATUS
0838            JSR SAVXY               ; SAVE X&Y
0840            JSR PUTCHR              ; PUT UP ASCII
0842            JSR RESXY               ; RESTORE X&Y
0844            LDA #$P                 ; LOAD SPACE
0846            STA FILSTA+1,X          ; CLEAR THIS STATUS
0848            DEX                     ; DECREMENT COUNT
0850            BPL PUTLTR              ; BRANCH TILL DONE
0852            INC FINUMB              ; INC FILE NUMBER
0854            INC SECMAX              ; AND FILES/SECTOR CNT
0856            LDA #8                  ; CLEAR OLD FILE ITEMS
0858            CMP SECMAX              ; DONE 8 FILES?
0860            BEQ NXTSEC              ; YES-GET NEXT SECTOR
0862            LDA *PAGE0              ; ELSE INC BUFFER POINTER
0864            CLC                     ; SO THAT WE
0866            ADC #16                 ; SKIP 16 BYTES
0868            STA *PAGE0              ; STORE NEW POINTER LOW
0870            BCC NOHIBY              ; JUMP IF NO CARRY
0872            INC *PAGE0+1            ; ELSE INC HI PART OF POIN
0874 NOHIBY     LDA #CR                 ; LOAD RETURN
0876            JSR PUTCHR              ; DO IT
0878            JSR CLRNAM              ; CLEAR NAME BUFFER
0880            JMP FILOOP              ; & GET NXT FILE INFO
0882 NXTSEC     LDA #CR                 ; EXECUTE A
0884            JSR PUTCHR              ; LINE FEED
0886            LDA DAUX1               ; INC DSK IOCB
0888            CLC                     ; TO READ THE NEXT
0890            ADC #1                  ; SECTOR
0892            STA DAUX1               ; THEN STORE
0894            BCC NOFLIP              ; INC HI
0896            INC DAUX2               ; IF NEEDED
```

```
0898 NOFLIP     LDX #0              ; CLEAR FILE CNT
0900            STX SECMAX
0902            LDA TOGGLE          ; DONE 2 SECTORS?
0904            BNE CLENUP          ; YES CLEAN UP OUR ACT
0906            INC TOGGLE          ; NO..SET FOR NEXT TIME TH
0908            JMP REDIR           ; AND GO READ NEXT SECTOR
0910 ;**********************
0912 ;* CLEAR NAME BUFFER *
0914 ;*     SUBROUTINE    *
0916 ;**********************
0918 CLRNAM     LDX #11             ; 11 CHARACTERS
0920            LDA #SP             ; ONE SPACE
0922 CLRSPA     STA NAMBUF,X        ; CLEAR IT
0924            DEX                 ; GET EVERYTHING!!
0926            BNE CLRSPA          ; TILL DONE
0928            RTS                 ; THEN RETURN
0930 ;********************
0932 ;* DIRECTORY DUMP *
0934 ;*   EXITS HERE   *
0936 ;********************
0938 CLENUP     DEC TOGGLE          ; CLEAR TOGGLE LOCATION
0940            JMP EXIT2           ; AND THEN JUMP BACK TO BA
0942 ;********************
0944 ;* DIRECTORY DATA *
0946 ;********************
0948 NAMBUF     .DS 11              ; FILE NAME BUF
0950 STASEC     .DS 2               ; START SEC HI/LO
0952 FILEN      .DS 2               ; FILE LEN  HI/LO
0954 FILSTA     .BY 32 32 32 32 32 ; FILE STATUS

0956 TOGGLE     .DS 1               ; INFAMOUS TOGGLER
0958 SECMAX     .DS 1               ; FILE COUNTER
0960 FINUMB     .DS 1               ; THIS FILE #
0962 ;********************
0028            .FI "D:DMPUN1V2.SRC"
1002 ;************************
1004 ;* CHANGE SECTOR LINKS *
1006 ;************************
1008 ENTLNK     JSR DSKVEC          ; READ SECTOR
1010            BPL CHGLNK          ; JMP GOOD READ
1012            JMP DERR            ; ELSE ERROR
1014 CHGLNK     JSR WEIRD           ; BREAK UP FI&SECTOR
1016            JMP EXIT2           ; AND RETURN
1018 NEWLNK     LDA FILNUM          ; GET NEW FILE
1020            ROL A               ; ROLL IT 2
1022            ROL A               ; BITS LEFT
1024            ORA TOPSEC          ; OR IN HI SECTOR
1026            STA CASBUF+125      ; STUFF BACK
1028            JMP EXIT2           ; AND RETURN
1030 ;********************
1032 ;* EXTRACT HI/LO *
1034 ;*  NEXT SECTOR  *
1036 ;*  SUBROUTINE   *
1038 ;* THIS AVOIDS A *
1040 ;* HORRENDOUS BUG*
1042 ;* OF TYING UP   *
1044 ;* THE STACK (AT *
1046 ;* LEAST I HOPE) *
1048 ;********************
1050 ;
1052 WEIRD      LDA CASBUF+125      ; GET WEIRDO
1054            PHA                 ; STUFF AWAY
1056            AND #$03            ; MASK OFF BITS 2-7
1058            STA TOPSEC          ; STUFF HI BY
1060            PLA                 ; GET STUFFED BYTE
1062            LSR A               ; RIGHT JUSTIFY
1064            LSR A               ; FILE NUMBER
1066            STA FILNUM          ; STUFF IT
1068            RTS                 ; GO HOME
1070 ;**** TRACE SECTORS ENTERS HERE ****
1072 TRASEC     LDA CONTIN          ; FIRST TIME?
1074            BEQ NOCON           ; YES..DO PULLS
1076            PLA                 ; JUST PULL VAR PASSED
1078            JMP SETY            ; NO..NO PULLS
1080 NOCON      PLA                 ; PULL OUT VAR CNT
1082            PLA                 ; PULL START HI
1084            STA DAUX2           ; STORE FOR READ
1086            STA DINUMH          ; STO FOR RECOV
1088            PLA                 ; GET START LO
1090            STA DAUX1           ; STORE FOR READ
1092            STA DINUML          ; STO FOR RECOV
1094            PLA                 ; INDEX HI
1096            STA *PAGE0+1        ; STORE
1098            PLA                 ; INDEX LO
1100            STA *PAGE0          ; STORE
1102            PLA                 ; FILE# HI
1104            PLA                 ; FILE# LO
1106            STA FINUMB          ; STORE IT
1108            JSR DSKVEC          ; READ SECTOR
1110            BPL NOERR           ; BRANCE NO ERROR
1112 ERRTRP     JMP RDERR           ; ERROR..JUMP
1114 NOERR      LDX #11             ; NAME COUNT
1116            LDY #15             ; INDEX CNT
1118 LOADIT     LDA (PAGE0),Y       ; GET LETTER
1120            STA NAMBUF-1,X      ; STUFF IT )
1122            DEY                 ; DEC Y
1124            DEX                 ; AND X
1126            BNE LOADIT          ; GET MORE
1128            LDA (PAGE0),Y       ; GET START LO
1130            STA DAUX2           ; STUFF
1132            DEY
1134            LDA (PAGE0),Y       ; GET LO
1136            STA DAUX1           ; STUF
1138            DEY
1140            LDA (PAGE0),Y       ; GET FI LEN HI
1142            STA FILEN+1         ; SAVE FOR CKS
1144            DEY
1146            LDA (PAGE0),Y       ; FI LEN LO
1148            STA FILEN           ; SAVE IT TOO
1150            DEY
1152            LDA RECOVR          ; ARE WE DOING RECOV FIL?
1154            BEQ STCHK           ; NO..CHECK STATUS
1156            JMP DOREC           ; YES..DO RECOVER'
1158 STCHK      LDA (PAGE0),Y       ; GET STATUS
1160            BEQ NOENT           ; IF ZERO, NO ENTRY
1162            STA FILSTA          ; STORE IT
1164            BIT FILSTA          ; FILE DELETED??
1166            BPL ITSUSD          ; NO..ITS USED!!
1168            JSR DISNAM          ; YES..PUT NAME
1170            STX ERRFLG          ; NOTE ERROR
1172            JSR DELETE          ; PUT DEL MSG
1174            JMP EXIT2+1         ; AND SCRAM
1176 NOENT      LDX #H,NOMSG        ; MSG ADDR
1178            LDY #L,NOMSG
1180            JSR FLIPIT
1182            LDA FINUMB          ; FILE NUMBER
1184            JSR DOBOTH          ; DISPLAY
1186            JMP COMMEX          ; COMMON ERROR EXIT
1188 ;**** FILE USED PUT UP NAME ETC ****
1190 ITSUSD     LDX #H,NAME         ; MSG ADDR
1192            LDY #L,NAME         ; HI/LO
1194            JSR FLIPIT          ; DISPLAY
1196            JSR DISNAM          ; PUT UP NAME
1198            LDX #H,SECMSG       ; MSG ADDR
1200            LDY #L,SECMSG
1202            JSR FLIPIT          ; FLIP MSGS
1204            LDA DAUX2           ; GET TRUE START SECTOR
1206            JSR DOONE           ; DISP HEX
1208            LDA DAUX1           ; GET LO
1210            JSR DOBOTH          ; DISPLAY TOO
1212 SETY       LDY #16             ; DO 17 LINES
1214            STY YCNT            ; SAVE CNT
1216 DORETN     LDA #CR             ; CARR RET
1218            JSR PUTCHR          ; DISPLAY
1220            LDX #7              ; 8 SECTORS/LINE
1222            STX XCNT            ; SAVE IT
1224 DOREAD     JSR DSKVEC          ; READ SECTOR
1226            BPL DRIVON          ; JMP GOOD READ
1228            JMP RDERR           ; BAD READ!!
1230 DRIVON     JSR WEIRD           ; EXTRACT WEIRD BYTE
1232            INC FILCNT          ; INC COUNTER
1234            BNE GOON            ; NO WRAP YET
1236            INC FILCNT+1        ; ELSE INC HI
1238 GOON       LDA FILNUM          ; GET THIS FILE #
1240            CMP FINUMB          ; SAME AS 1 WE SEEK??
1242            BNE OOPS            ; NO!!! OOPS..
1244            LDA CASBUF+126      ; GET LO PTR
1246            ORA TOPSEC          ; OR WITH HI BYTE!!
1248            BEQ WEDONE          ; IF ZERO THEN DONE
1250            JSR SETUP           ; ELSE SETUP FOR NXT READ
1252 DOASEC     JSR SECDIS          ; DISPLAY TRACE SECTOR
1254            DEC XCNT            ; DEC CNTR
1256            BPL DOREAD          ; IF NOT DONE 8 THEN GET N
1258            LDA #1              ; ELSE SETUP NXT LINE
1260            STA FAKFLG          ; FOR ONLY #
1262            DEC YCNT            ; DEC LINE CNT
1264            BPL DORETN          ; IF NOT 16 GO BACK
1266            STA CONTIN          ; ELSE SET CONTINUE FLG
1268            JMP EXIT2+1         ; AND ESCAPE..WE WILL RETU
1270 OOPS       JSR CLRNAM          ; CLEAR NAME
1272            LDX #H,LINKER       ; MSG ADDR
1274            LDY #L,LINKER
1276            JSR FLIPIT          ; DISPLAY
1278 ONMDER     JSR JUSHEX          ; PUT UP #
1280            LDA DAUX2           ; HI SEC
1282            JSR DOONE           ; DISP JUST LO
1284            LDA DAUX1           ; LO SEC
1286            JSR DOBOTH          ; DISPLAY
1288 LASMSG     LDX #H,ODAMSG       ; MSG HI
1290            LDY #L,ODAMSG       ; MSG LO
1292            JSR FLIPIT          ; PUT IT UP
1294 COMMEX     LDA #1              ; RESTORE FAKE
1296            STA ERRFLG
1298 CLRFAK     STA FAKFLG          ; RESET FAKE
1300            LDA #0              ; CLR CONTIN
1302            STA FILCNT          ; CLEAR COUNTR
1304            STA CONTIN
1306            STA FILCNT+1        ; HI CLR
1308            JMP EXIT2+1         ; AND SCRAM
1310 WEDONE     LDA FILEN           ; GET ORIGINAL LO
1312            EOR FILCNT          ; OR WITH COUNTER
1314            BNE SORRY           ; IF NOT ZERO..SORRY
1316            LDA FILEN+1         ; DO HI NOW
1318            EOR FILCNT+1        ; OR THEM
1320            BNE SORRY           ; SAME AS ABOVE
1322            STA FILCNT          ; CLR FOR NXT TIME
1324            STA FILCNT+1
1326            LDX #H,STOPMS       ; ANNOUNCE WE DONE
1328            LDY #L,STOPMS       ; TO THE WORLD
1330            JSR FLIPIT          ; DO IT
1332            LDA #CR             ; CARR RET
1334            JSR PUTCHR          ; DO IT
1336            LDA #1              ; LOAD TO CLR
1338            JMP CLRFAK          ; EXIT GRACIOUSLY
1340 SORRY      LDX #H,FILMSG       ; MSG ADDR
1342            LDY #L,FILMSG
1344            JSR FLIPIT          ; PUT IT UP
1346            LDA FILEN+1
1348            JSR DOBOTH
1350            LDA FILEN
1352            JSR DOBOTH
1354            LDX #H,FIMSG1       ; NEXT
1356            LDY #L,FIMSG1
1358            JSR FLIPIT          ; DISPLAY
1360            LDA FILCNT+1
1362            JSR DOBOTH
1364            LDA FILCNT
1366            JSR DOBOTH
1368            LDX #H,FIMSG2       ; LAST
1370            LDY #L,FIMSG2
1372            JSR FLIPIT          ; DISPLAY
1374            LDA #0              ; CLEAR FILCNT
1376            STA FILCNT
1378            STA FILCNT+1
1380            JMP COMMEX          ; NOW EXIT W/ERROR
1382 ;********************
1384 ;* MORE SUBROUTINES *
1386 ;*      FOLLOW      *
1388 ;********************
1390 ;
1392 FLIPIT     LDA DISMSG+1        ; OLD PTR
1394            STA TEMP1
```

```
1396            LDA DISMSB+2
1398            STA TEMP2
1400            STY DISMSB+1        ; POINT TO
1402            STX DISMSB+2        ; NEW MSB
1404            JSR MSB             ; PUT UP THE MSB
1406            LDA TEMP1           ; RESTORE OLD
1408            STA DISMSB+1        ; MSB POINTER
1410            LDA TEMP2
1412            STA DISMSB+2
1414            LDX TEMPX           ; RESTORE X&Y
1416            LDY TEMPY           ; ALSO
1418            RTS                 ; THEN RETURN
1420 SETUP      LDA TOPSEC          ; SECTOR MSB
1422            STA DAUX2           ; STORE FOR NXT READ
1424 SETUP2     LDA CASBUF+126      ; SECTOR LSB
1426            STA DAUX1           ; STORE FOR NXT READ
1428            RTS                 ; AND RETURN
1430 DISNAM     LDX #0              ; START AT ZERO
1432 GETMOR     LDA NAMBUF,X        ; GET LETTER
1434            JSR SAVXY
1436            JSR PUTCHR          ; DISPLAY
1438            JSR RESXY
1440            INX
1442            CPX #11
1444            BCC GETMOR
1446            RTS                 ; GO BACK..JACK
1448 ;**********************
1450 ;* HERE ARE THE MSGS  *
1452 ;**********************
1454 ;
1456 NAME       .BY 'FILE: ' NULL
1458 SECMSG     .BY SP SP SP SP ' START SECTOR:$' NULL
1460 DELMSG     .BY ' IS DELETED!!' BELL BELL CR NULL
1462 LINKER     .BY CR 'FILE NUMBER MISMATCH AT SECTOR' ESC
1464            .BY NULL
1466 ODAMSG     .BY CR 'CHECK PREVIOUS SECTOR LINKS!!'
1468            .BY BELL BELL NULL
1470 RDMSG      .BY 'CANNOT READ SECTOR:' ESC RAR '$' BELL
1472 NOMSG      .BY CR 'NO ENTRY FOR FILE' ESC RAR '$' BELL
1474 STOPMS     .BY SP 'END' NULL
1476 FILMSG     .BY CR 'ORIGINAL SECTOR COUNT'
1478 FIMSG1     .BY CR 'ACTUAL SECTORS LOADED'
1480 FIMSG2     .BY CR 'SHORT FILE ERROR!!!'
1482 DELETE     LDX #H,DELMSG       ; POINT TO
1484            LDY #L,DELMSG       ; DELETED
1486            JSR FLIPIT          ; DISPLAY
1488            RTS                 ; RETURN
1490 RDERR      LDX #H,RDMSG
1492            LDY #L,RDMSG
1494            JSR FLIPIT          ; DISPLAY
1496            LDA DAUX2           ; SECTOR HI
1498            JSR DOONE           ; DISPLAY
1500            LDA DAUX1           ; SECTOR LO
1502            JSR DOBOTH          ; DISPLAY
1504            LDA #CR             ; LINE FEED
1506            JSR PUTCHR          ; DO IT
1508            JMP COMMEX          ; AND EXIT
1510 ;**********************
1512 ;* TRACE SECTOR VARS *
1514 ;**********************
1516 XCNT       .BY 0
1518 YCNT       .BY 0
1520 TEMP1      .BY 0
1522 TEMP2      .BY 0
1524 TEMPX      .BY 0
1526 TEMPY      .BY 0
1528 CONTIN     .BY 0
1530 FILCNT     .BY 0 0
1532 ;*****************************
1534 ;* THIS SUBROUTINE DISPLAYS *
1536 ;* THE SECTOR TRACE IN THE  *
1538 ;* FORM >$XXX. DEPENDING ON *
1540 ;* THE FAKFLG VALUE THE 1ST*
1542 ;* VALUE OF EACH LINE WILL  *
1544 ;* BE OF THE FORM $XXX.     *
1546 ;*****************************
1548 SECDIS      JSR SAVXY           ; SAVE X&Y
1550            LDA FAKFLG          ; ONLY $XXX THIS TIME?
1552            BEQ DOPREF          ; NO-DO >$
1554            JSR JUSHEX          ; YES-JUST$
1556            JMP FAKEONE         ; THEN JUMP
1558 DOPREF     LDA #ESC            ; SEND ESC FIRST
1560            JSR PUTCHR          ; SEND IT
1562            LDA #RAR            ; LOAD RIGHT ARROW CHAR
1564            JSR PUTCHR          ; SEND IT
1566            JSR JUSHEX          ; PUT UP $
1568 FAKEONE    LDA TOPSEC          ; GET UPPER NYBLE
1570            JSR DOONE           ; DISPLAY ONLY LO NYBLE
1572            LDA CASBUF+126      ; GET LO BYTE
1574            JSR DOBOTH          ; DISPLAY IT
1576            LDA #0              ; LOAD ZERO
1578            STA FAKFLG          ; CLEAR FAKFLG
1580            RTS                 ; AND RETURN
1582 ;**********
1584 FAKFLG     .BY 1               ; INITIALLY ONE
1586 ;**********
1588 ;
1590 ;******************
1592 ;* PRINT 1 BYTE  *
1594 ;******************
1596 DOONE      JSR CONVERT         ; CONVERT TO ASCII
1598            JSR ONEBYE          ; DISPLAY LO NYBLE ONLY
1600            RTS                 ; AND RETURN
2000 ;*************************
2005 ;* IOCB HANDLER CODE FOR *
2010 ;* DISKTOOL.THIS CODE WAS*
2015 ;* WRITTEN TO GET RID OF *
2020 ;* THE UNAUTHORIZED CALL *
2025 ;* TO THE PUTCHR ROUTINE.*
2030 ;* THIS CAUSED PROBLEMS  *
2035 ;* ON THE 1200XL AND WAS *
2040 ;* NOT A CORRECT METHOD  *
2045 ;* FOR WRITING A CHARAC- *
2050 ;* TER TO THE SCREEN.    *
2055 ;* THE CIO WAY IS MORE   *
2060 ;* FLEXIBLE AND SHOULD AL*
2065 ;* LOW DISKTOOL TO RUN   *
2070 ;* WITH ALL ATARI PROD-  *
2075 ;* UCTS IN THE FUTURE....*
2080 ;*************************
2085 ;
2090 ;************************
2095 ;* IOCB EQUATES FOLLOW *
2100 ;************************
2105 ;
2110 IOCB4      .DE $40             ; IOCB 4*16
2115 IOCBST     .DE $0340           ; START IOCB BLKS
2120 ICHID      .DE IOCBST          ; HANDLER ID
2125 ICDNUM     .DE ICHID+1         ; DEVICE #
2130 ICCOM      .DE ICDNUM+1        ; COMMAND BYTE
2135 ICSTA      .DE ICCOM+1         ; STATUS BYTE
2140 ICBAL      .DE ICSTA+1         ; BUFFER ADDR LOW
2145 ICBAH      .DE ICBAL+1         ; BUFFER ADDR HIGH
2150 ICPTL      .DE ICBAH+1         ; POINTER LO
2155 ICPTH      .DE ICPTL+1         ; POINTER HI
2160 ICBLL      .DE ICPTH+1         ; BUFF LEN LOW
2165 ICBLH      .DE ICBLL+1         ; BUFF LEN HI
2170 ICAX1      .DE ICBLH+1         ; AUX BYTE 1
2175 ICAX2      .DE ICAX1+1         ; AUX BYTE 2
2180 ;
2185 ; 4 SPARE UN-LABELED BYTES FOLLOW IN IOCB
2190 CIOVEC     .DE $E456           ; CIO ADDRESS
2195 CLOSE      .DE $0C             ; CLOSE COMMAND
2200 OPEN       .DE $03             ; OPEN COMMAND
2205 GETCHR     .DE $07             ; GET CHAR CMND
2210 PUTCAR     .DE $0B             ; PUT CHAR CMND
2215 GETREC     .DE $05             ; GET RECORD CMND
2220 PUTREC     .DE $09             ; PUT REC CMND
2225 WRITE      .DE $08             ; WRITE (FOR OPEN)
2230 READ       .DE $04             ; READ  (FOR OPEN)
2235 SPLIT      .DE $10             ; SPLIT SCREEN
2240 DSPEC      .BY 'E:' CR  ; EDITOR DEV
2245 ;
2250 ;**********************
2255 ;* NEW PUTCHR SUBRTN *
2260 ;**********************
2265 ;* A REG HAS CHARACTR*
2270 ;**********************
2275 ;
2280 PUTCHR     LDX #IOCB4          ; IOCB INDEX
2285 GOCIOV     JSR CIOVEC          ; GO TO CIO HANDLER
2290            RTS                 ; THEN RETURN
2295 ;
2300 ;**********************
2305 ;* INITIALIZE IOCB #4 *
2310 ;**********************
2315 ;
2320 CLRIOC4    LDX #IOCB4          ; INDEX TO #4
2325            LDA #CLOSE          ; CLOSE COMMAND
2330            STA ICCOM,X         ; STUFF COMMAND
2335            JSR CIOVEC          ; CLOSE IT
2340            LDX #IOCB4          ; RESET X
2345 OPNIOC4    LDA #OPEN           ; OPEN COMMAND
2350            STA ICCOM,X         ; STUFF IT
2355            LDA #L,DSPEC        ; LOAD ADDR TO E:
2360            STA ICBAL,X         ; STUFF IT
2365            LDA #H,DSPEC        ; ADDR H
2370            STA ICBAH,X         ; STUFF
2375            LDA #WRITE          ; WRITE ONLY
2380            STA ICAX1,X         ; STUFF
2385            JSR CIOVEC          ; GO TO CIO HANDLER
2390 ;
2395 ;*******************************
2400 ;* SET UP IOCB4 SO THAT WE CAN *
2405 ;* PASS THE CHARACTER TO DIS-  *
2410 ;* PLAY IN THE A REGISTER. THIS*
2415 ;* IS DONE BY USING THE PUTCAR *
2420 ;* COMMAND AND ENSURING THAT   *
2425 ;* THE BUFFER LENGTH IS ZERO.. *
2430 ;*******************************
2435 ;
2440            LDX #IOCB4          ; RESET X
2445 SET4       LDA #PUTCAR         ; PUT CAR CMND
2450            STA ICCOM,X         ; STUFF
2455            LDA #0              ; ZAP BUF LEN
2460            STA ICBLL,X         ; ZAP
2465            STA ICBLH,X         ; ZAP
2470            PLA                 ; CLEAR STACK
2475            RTS                 ; RETURN TO BASIC
3005 ;*******************************
3010 ;*  RECOVER FILE CODE FOLLOWS  *
3015 ;*******************************
3020 ;* THE ACTUAL RECOVER ENTRANCE *
3025 ;* IS AT LABEL NOCON. IN THE   *
3030 ;* BASIC PART OF DISKTOOL THE  *
3035 ;* RECOVR FLAG IS SET SO THAT  *
3040 ;* THE PROGRAM WILL JUMP HERE  *
3045 ;* THIS WAS DONE TO SAVE SOME  *
3050 ;* CODE REPETITION.            *
3055 ;*******************************
3060 DCOMM      .DE 770             ; COMMAND BYTE
3065 DBUFLO     .DE 772             ; DSK BUF PTR LO
3070 DBUFHI     .DE 773             ; DSK BUF PTR HI
3075 ;
3080 STSECL     .BY 0               ; SEC LO STORE
3085 STSECH     .BY 0               ; SEC HI STORE
3090 RECOVR     .BY 0               ; RECOVER FLAG
3095 DINUML     .BY 0               ; DIRECTORY NUM LO
3100 DINUMH     .BY 0               ; AND HI
3105 VTOC       .DS 132             ; 132 BYTE BUFFER
3110 ;
3115 DOREC      STY TEMPY           ; SAVE STATUS INDEX
3120            LDA DAUX2           ; GET DIR SEC HI
3125            STA STSECH          ; SAVE IT
3130            LDA DAUX1           ; GET DIR SEC LO
3135            STA STSECL          ; SAVE IT
3140            LDX #H,VTOC         ; ALT BUF HI ADDR
3145            LDY #L,VTOC         ; ALT BUF LO ADDR
3150            STX DBUFHI          ; STUF IN IOCB
3155            STY DBUFLO          ; ALSO LO
3160            LDX #$68            ; VTOC SECTOR 360
```

```
3165            LDY #$01            ; IS $0168
3170            STY DAUX2           ; STUFF MSB
3175            STX DAUX1           ; STUFF LSB
3180            JSR DSKVEC          ; LOAD VTOC INTO ALT BUFFE
3185            BPL RESBUF          ; IF NO ERROR RESTORE OLD
3190            LDX #H,VRBAD        ; MSB HI ADR
3195            LDY #L,VRBAD        ; LO ADR
3200            JSR FLIPIT          ; DISPLAY IT
3205            JSR CLRNAM          ; CLEAR NAME
3210            JMP RDERR           ; ELSE EXIT W/ERROR
3215 RESBUF     JSR RESDSK          ; RESTORE DSK IOCB
3220 ;
3225 ;*********************************
3230 ;* RESTORE INDEX AND CK STATUS   *
3235 ;*********************************
3240 ;
3245            LDY TEMPY           ; RESTORE INDEX
3250            LDA (PAGE0),Y       ; GET STATUS
3255            BNE STOFIL          ; JMP IF ENTRY
3260            JMP NOENT           ; ERROR SO JMP
3265 STOFIL     STA FILSTA          ; ELSE SAVE STATUS
3270            BIT FILSTA          ; FILE DELETED FOR SURE??
3275            BPL FIUSED          ; NO!! USER PLAYING TRICKS
3280            LDX #H,PASS1        ; OK..ITS DELETED
3285            LDY #L,PASS1        ; NOTIFY USER WHATS HAPNIN
3290            JSR FLIPIT          ; DISPLAY MSG
3295 ;
3300 ;*********************************
3305 ;* TRACE SECTORS FOR CONTINUITY*
3310 ;*********************************
3315 ;
3320            LDA STSECH          ; RESTORE SEC HI
3325            STA DAUX2           ; STUF IN IOCB
3330            LDA STSECL          ; GET SEC LO
3335            STA DAUX1           ; STUF IT TOO
3340 VERIFY     JSR DSKVEC          ; GET A SECTOR
3345            BPL V80             ; JMP GOOD READ
3350            JSR DISNAM          ; DIPLAY NAM
3355            JSR CLRNAM          ; CLR NAME
3360            LDX #H,FIRDE        ; MSB HI ADR
3365            LDY #L,FIRDE        ; MSB LO ADR
3370            JSR FLIPIT          ; DISPLAY MSG
3375            JMP RDERR           ; EXIT W/ERROR
3380 V80        JSR WEIRD           ; EXTRACT LO/HI NXT SECTOR
3385            INC FILCNT          ; INC SECTOR COUNTER
3390            BNE V80ON           ; BRANCH NO WRAP
3395            INC FILCNT+1        ; ELSE INC HI
3400 V80ON      LDA FILNUM          ; GET FILE # TO WHICH SEC
3405            CMP FINUMB          ; SAME AS I WE SEEK?
3410            BNE NOREC           ; NO..FILE NOT RECOVERABLE
3415            LDA CASBUF+126      ; YES. GET LO POINTER
3420            ORA TOPSEC          ; OR WITH HI SECTOR
3425            BEQ CKSECN          ; IF ZERO..TRACE IS DONE
3430            JSR SETUP           ; ELSE SETUP NXT READ
3435            JMP VERIFY          ; AND GET NXT SECTOR
3440 ;
3445 ;*********************************
3450 ;* PASS1 ERROR PROCESSING HERE *
3455 ;*********************************
3460 ;
3465 FIUSED     JSR DISNAM          ; DISPLAY FILE NAME
3470            LDX #H,NODEL        ; MSB HI
3475            LDY #L,NODEL        ; MSB LO
3480            JSR FLIPIT          ; DISPLAY IT
3485 EUSE       JMP COMMEX          ; EXIT WITH ERROR
3490 NOREC      LDX #H,LINKER       ; MSB HI
3495            LDY #L,LINKER       ; MSB LO
3500            JSR FLIPIT          ; DISPLAY
3505            JSR JUSHEX          ; PUT UP A $
3510            LDA DAUX2           ; HI SEC #
3515            JSR DOONE           ; DISPLAY ONLY LO NYBLE
3520            LDA DAUX1           ; LO SEC NUM
3525            JSR DOBOTH          ; DISPLAY WHOLE THING
3530            LDA #CR             ; LOAD CRET
3535            JSR PUTCHR          ; DISPLAY
3540            LDX #H,NAME         ; FILE: MSB
3545            LDY #L,NAME         ; LO ADDR
3550            JSR FLIPIT          ; PUT UP MSB
3555            JSR DISNAM          ; PUT UP FILE NAME
3560            LDX #H,NORECO       ; NON-RECOVER MSB HI
3565            LDY #L,NORECO       ; MSB LO
3570            JSR FLIPIT          ; DISPLAY
3575            JSR CLRNAM          ; CLEAR FILE NAME
3580            BEQ EUSE            ; EXIT W/ERROR
3585 DOOM       JSR DISNAM          ; DISP NAME
3590            JSR CLRNAM          ; CLR NAME
3595            LDX #H,NORECO       ; ADR HI
3600            LDY #L,NORECO       ; ADR LO
3605            JSR FLIPIT          ; DISPLAY IT
3610            JMP SORRY           ; CONTINUE ER
3615 ;
3620 ;*********************************
3625 ;* CHECK TO SEE THAT LENGTH IN *
3630 ;* DIRECTORY MATCHES LEN OF FIL*
3635 ;*********************************
3640 ;
3645 CKSECN     LDA FILEN           ; ORIGINAL LO
3650            EOR FILCNT          ; OR W/COUNT
3655            BNE DOOM            ; NO..ERROR
3660            LDA FILEN+1         ; ORIG HI
3665            EOR FILCNT+1        ; OR ALSO
3670            BNE DOOM            ; GO IF ERROR
3675            STA FILCNT          ; CLR COUNTER
3680            STA FILCNT+1        ; LO/HI
3685 ;
3690 ;*********************************
3695 ;* BEGIN PASS 2 OF FILE RECOVER*
3700 ;*********************************
3705 ;
3710 PASS2      LDX #H,OK           ; OK MSB HI
3715            LDY #L,OK           ; AND LO
3720            JSR FLIPIT          ; DISPLAY
3725 ;
3730 ;*********************************
3735 ;* GET DIRECTORY FOR OUR FILE *
3740 ;*********************************
3745 ;
3750            LDA DINUML          ; LO BYTE
3755            STA DAUX1           ; STUFF IT
3760            LDA DINUMH          ; HI BYTE
3765            STA DAUX2           ; STUFF IT
3770            JSR DSKVEC          ; GO READ IT
3775            BPL CHKDOS          ; JMP GOOD RD
3780            LDX #H,DIRDE        ; MSB HI
3785            LDY #L,DIRDE        ; MSB LO
3790            JSR FLIPIT          ; DISPLAY IT
3795            JSR CLRNAM          ; CLR NAME
3800            JMP COMMEX          ; ELSE EXIT
3805 CHKDOS     LDY #0              ; CLEAR Y REG
3810            LDA VTOC            ; GET VTOC DOS
3815            CMP #DOSMSK         ; IS IT DOS 2?
3820            BNE DOS1FL          ; NO..DOS 1
3825            LDA #$42            ; LOAD FILE USED
3830            BNE STOSTA          ; AND BRANCH
3835 DOS1FL     LDA #$40            ; ELSE ONLY USED
3840 STOSTA     STA (PAGE0),Y       ; STORE NEW STATUS
3845            LDA #'W             ; LOAD WRITE COMMAND
3850            STA DCOMM           ; STUFF IN DSK IOCB
3855            JSR DSKVEC          ; WRITE OUT DIR SEC
3860            BPL WFINE           ; JMP GOOD WRITE
3865            JSR RESDSK          ; RESTORE DSK IOCB
3870            JSR CLRNAM          ; ERROR CLR NAME
3875            LDX #H,DIRWE        ; MSB HI
3880            LDY #L,DIRWE        ; MSB LO
3885            JSR FLIPIT          ; DISPLAY
3890            LDA DAUX2           ; SECTOR MSB
3895            JSR DOONE           ; DISP 1 DIGIT
3900            LDA DAUX1           ; SECTOR LSB
3905            JSR DOBOTH          ; DISP 2 DIGS
3910            LDX #H,DIRWE2       ; PT2 HI
3915            LDY #L,DIRWE2       ; PT2 LO
3920            JSR FLIPIT          ; DISPLAY
3925            JMP COMMEX          ; EXIT W/ERROR
3930 WFINE      JSR RESDSK          ; RESTORE DSK IOCB
3935            LDX #H,DIRENT       ; DIRECTORY DONE MSB
3940            LDY #L,DIRENT       ; AND LO
3945            JSR FLIPIT          ; WRITE IT
3950 ;
3955 ;*********************************
3960 ;* NOW RETRACE EACH SECTOR AND   *
3965 ;* UPDATE THE BIT MAP TO FULLY   *
3970 ;* RECOVER THE FILE..DO NOT FOR-*
3975 ;* GET TO DECREMENT THE # OF SEC*
3980 ;* TORS AVAILABLE FOR EACH SEC-  *
3985 ;* TOR THAT IS ALLOCATED...      *
3990 ;*********************************
3995 ;
4000            LDX #H,REALO        ; REALOCATE MSB
4005            LDY #L,REALO        ; HI/LO
4010            JSR FLIPIT          ; DISPLAY
4015            LDA STSECL          ; LO SEC
4020            STA DAUX1           ; STUFF IT
4025            LDA STSECH          ; HI SEC
4030            STA DAUX2           ; STUFF IT
4035 RREC       JSR DSKVEC          ; GET A SECT
4040            BPL RGO             ; JMP GOOD RD
4045            LDX #H,FIRDE        ; MSB HI
4050            LDY #L,FIRDE        ; MSB LO
4055            JSR FLIPIT          ; DISPLAY
4060            JSR CLRNAM          ; CLEAR NAME
4065            JMP RDERR           ; GO READ ER
4070 RGO        JSR FINDIT          ; FIND VTOC BYTE
4075            JSR WEIRD           ; GET NXT SEC
4080            LDA CASBUF+126      ; GET LO PTR
4085            ORA TOPSEC          ; OR WITH HI
4090            BEQ RECDON          ; IF # WE FINEETO
4095            JSR SETUP           ; ELSE SET NXT READ
4100            JMP RREC            ; AND DO IT
4105 ;
4110 ;*********************************
4115 ;* RECOVER FILE CLEAN UP CODE. *
4120 ;* SAVE NEW VTOC, CLEAR FLAGS, *
4125 ;* PUT UP SUCCESS MSB AND RET. *
4130 ;* TO BASIC. FILE IS RECOVERED.*
4135 ;* (PLEASE LET IT BE RECOVERED)*
4140 ;*********************************
4145 ;
4150 RECDON     LDX #H,VTOC         ; BUF PTR H
4155            LDY #L,VTOC         ; BUF PTR L
4160            STX DBUFHI          ; STUFF IT
4165            STY DBUFLO          ; STUFF IT
4170            LDX #$68            ; SECTOR 360
4175            LDY #$01            ; = $0168
4180            STX DAUX1           ; STUFF
4185            STY DAUX2           ; STUFF
4190            LDA #'W             ; WRITE COM
4195            STA DCOMM           ; STUFF IT
4200            JSR DSKVEC          ; WRITE IT
4205            BPL VOUTOK          ; JMP GOOD W
4210            LDX #H,VBAD         ; GET MSB
4215            LDY #L,VBAD         ; ADDRESS
4220            JSR FLIPIT          ; DISPLAY IT
4225            JSR RESDSK          ; RESTORE DSK IOCB
4230            JMP COMMEX          ; EXIT W/ERR
4235 VOUTOK     JSR RESDSK          ; RESTORE IOCB
4240            JSR DISNAM          ; DISP NAME
4245            LDX #H,SUCCES       ; SUCCES MSB
4250            LDY #L,SUCCES       ; ADDRESS
4255            JSR FLIPIT          ; DISPLAY IT
4260            JSR CLRNAM          ; CLEAR NAME
4265            STX RECOVR          ; CLEAR RECOVER FLAG
4270            JMP EXIT2+1         ; RETURN TO BASIC!!
4275 ;
4280 ;*********************************
4285 ;* SUBROUTINE RESDSK: RESTORES   *
4290 ;* THE DISK IOCB TO A READ CON-  *
4295 ;* DITION AND POINTS THE BUFFER  *
4300 ;* TO CASBUF.                    *
4305 ;*********************************
4310 ;
4315 RESDSK     LDX #H,CASBUF       ; BUF HI
4320            LDY #L,CASBUF       ; BUF LO
```

```
4325            STX DBUFHI          ; STUFF
4330            STY DBUFLO          ; STUFF
4335            LDX #'R             ; READ COM
4340            STX DCOMM           ; STUFF
4345            RTS                 ; RETURN
4350 ;
4355 ;*********************************
4360 ;* SUBROUTINE FINDIT: FINDS THE*
4365 ;* APPROPRIATE BYTE OF THE VTOC*
4370 ;* AND THE PROPER BIT OF THAT  *
4375 ;* BYTE WHICH REPRESENTS THE   *
4380 ;* CURRENT SECTOR OF OUR FILE. *
4385 ;* THE SECTOR MAP IS UPDATED TO*
4390 ;* ALLOCATE THE CURRENT SECTOR *
4395 ;* AS BEING USED. IN ADDITION, *
4400 ;* THE NUMBER OF FREE SECTORS  *
4405 ;* AVAILABLE IS DECREMENTED SO *
4410 ;* THAT DOS DOESNT GO BANNANAS.*
4415 ;*********************************
4420 ;
4425 FINDIT     LDA #0              ; CLEAR OUT A
4430            LDY #3              ; PREP FOR /8
4435 FIND0      LSR DAUX2           ; DIVIDE BY SHIFTING
4440            ROR DAUX1           ; SAME FOR LO
4445            ROR A               ; AND ALSO A REG
4450            DEY                 ; DEC CNTR
4455            BNE FIND0           ; DO 3 SHIFTS TO DIV
4460            LDY #5              ; DETERMINE SHFT CNT
4465 FIND1      ROR A               ; ROTATE BIT IN A REG
4470            DEY                 ; DEC CNT
4475            BNE FIND1           ; DO 5 TIMES
4480            TAY                 ; USE AS CNTR IN Y
4485            LDA #0              ; CLEAR A
4490            SEC                 ; SET OUR BIT IN CARRY
4495 FIND2      ROR A               ; AND SHIFT TO PROPER
4500            DEY                 ; DEC SHFT CNT
4505            BPL FIND2           ; SHFT TIL NEGATIVE
4510            PHA                 ; SAVE THE PROPER BIT
4515            LDA DAUX1           ; GET VTOC BYTE NUMBER
4520            ADC #0A             ; ADD VTOC OFFSET
4525            TAY                 ; INDEX OF VTOC BYTE
4530            PLA                 ; PULL THE MASK
4535            EOR VTOC,Y          ; CLEAR SECTOR BIT
4540            STA VTOC,Y          ; PUT BACK IN VTOC
4545            DEC VTOC+3          ; DEC AVAILABLE SECTORS
4550            LDA VTOC+3          ; GET VAL
4555            CMP #0FF            ; DID WE FLIP?
4560            BNE NOHI            ; NO.RETURN
4565            DEC VTOC+4          ; ELSE DEC HI
4570 NOHI       RTS                 ; AND RETURN
4575 ;*********************************
4580 ;* RECOVER FILE MESSAGES FOLLOW*
4585 ;*********************************
4590 ;
4595 PASS1      .BY TAB 'PASS1 - '
4600            .BY 'CHECKING FILE CONDITION' CR NULL
4605 NODEL      .BY ' IS NOT DELETED!!' CR BELL NULL
4610 NORECO     .BY ', CANNOT BE RECOVERED!!' BELL CR NULL
4615 OK         .BY 'FILE INTACT' CR TAB 'PASS2 - '
4620            .BY 'RECOVERING FILE' CR NULL
4625 DIRENT     .BY 'DIRECTORY ENTRY DONE' CR NULL
4630 REALO      .BY 'REALLOCATING DELETED SECTORS' CR NULL
4635 SUCCES     .BY ' HAS BEEN RECOVERED!' BELL CR NULL
4640 VBAD       .BY 'ERROR IN VTOC WRITE!!' BELL CR NULL
4645 VRBAD      .BY 'ERROR IN VTOC READ !!' BELL CR NULL
4650 FIRDE      .BY 'FILE READ ERROR!' BELL CR NULL
4655 DIRDE      .BY 'DIRECTORY READ ERROR!' BELL CR NULL
4660 DIRWE      .BY 'DIRECTORY WRITE ERROR!' ESC RAR
4665 DIRWE2     .BY BELL CR NULL
0034            .EN
```

--- LABEL FILE: ---

```
AROUND =1E0F        AS2BIN =1E65        ASBIN1 =1E6E
ASTART =1CFC        BELL =00FD          CASBUF =03FD
CHGLNK =1FFC        CHKDOS =2518        CHNGBY =1E31
CHRCNT =1D01        CIOVEC =E456        CKDOOM =1DC3
CKDOS =1EEA         CKLOCK =1EDE        CKSECN =24DD
CKSTA =1ECA         CKUSED =1ED7        CLENUP =1FD7
CLOSE =000C         CLRFAK =212F        CLRIOC4 =233F
CLRNAM =1FCC        CLRSPA =1FD0        CLS =007D
COMMEX =212A        CONASC =1DB8        CONTIN =22FE
CONVERT =1DF3       CR =009B            DAUX1 =030A
DAUX2 =030B         DBUFHI =0305        DBUFLO =0304
DCOMM =0302         DELETE =22D5        DELMSG =22D3
DERR =1D34          DINUMH =2377        DINUML =2376
DIRDE =2726         DIRENT =2698        DIRWE =273E
DIRWE2 =2756        DISMSG =1D07        DISNAM =21B4
DISPFL =1F12        DISPLY =1DDE        DMPASC =1D41
DOASEC =20F2        DOBOTH =1E08        DOEXT =1F2C
DOOM =24CD          DOONE =232F         DOPREF =230F
DOREAD =20CC        DOREC =23FC         DORETN =20C2
DOS1 =1EF9          DOS1FL =2525        DOSMSK =0002
DRIVON =20D4        DSKVEC =E453        DSPEC =2336
DSPHEX =1D19        ENDMSG =1D99        ENTLNK =1FF4
ERRFLG =1D7E        ERRTRP =2048        ESC =001B
EUSE =2499          EXIT =1D37          EXIT2 =1D3F
FAKEONE =231C       FAKFLG =232E        FILCNT =22FF
FILEN =1FEA         FILMSG =2208        FILNUM =1D84
FILOOP =1E9C        FILSTA =1FEC        FIMSG1 =22A5
FIMSG2 =22BF        FIND0 =25EE         FIND1 =25FA
FIND2 =2602         FINDIT =25EA        FINUMB =1FF3
FIRDE =2713         FISTAR =1EBF        FIUSED =248F
FLIPIT =219F        GETCHR =0007        GETIT =1D4C
GETLEN =1EB3        GETMOR =21D6        GETNAM =1EA0
GETREC =0005        GETSTA =1EAD        GOCIOV =233B
GOON =20DF          HEADER =1D9A        HIADR =1E17
HIHEX =1DF2         HX2BIN =1E47        ICAX1 =034A
ICAX2 =034B         ICBAH =0345         ICBAL =0344
ICBLH =0349         ICBLL =0348         ICCOM =0342
ICDNUM =0341        ICHID =0348         ICPTH =0347
ICPTL =0346         ICSTA =0343         INDEX =1E38
IOCB4 =0040         IOCBST =0340        ITSUSD =20A0
JPONT =1E85         JUSHEX =1E2B        LASMSG =2123
LINKER =2214        LOADIT =204F        LOADR =1E16
LOCKED =0020        LOHEX =1DF1         LT9 =1DC0
MAKBIN =1E6F        MESAGE =1D06        MSG =1D85
NAMBUF =1FDD        NAME =21E8          NEWLNK =2002
NOCON =2029         NODEL =2644         NOENT =2090
NOERR =204B         NOFLIP =1FBC        NOHI =2621
NOHIBY =1F9E        NOMSG =226F         NOREC =249C
NORECO =2658        NOTASC =1E54        NULL =0000
NXTSEC =1FA9        ODAMSG =2236        OK =2672
ONEBYE =1DE7        ONMOER =2114        OOPS =210A
OPEN =0003          OPNIOC4 =234B       OUT =1DDD
OUTFIL =1EFE        PAGE0 =00CD         PASS1 =2622
PASS2 =24F3         PERIOD =002E        PREFIX =1E26
PSTATU =1F70        PUTCAR =000B        PUTCHR =2339
PUTLEN =1F58        PUTLTR =1F72        PUTREC =0009
PUTSEC =1F04        PUTSTA =1F46        RAR =001F
RDERR =22DD         RDMSG =2257         READ =0004
REALO =26AE         RECDON =2599        RECOVR =2375
REDIR =1E8F         RESBUF =2431        RESDSK =25DA
RESXY =1E1F         RGO =2585           RREC =2573
SAVEX =1D7F         SAVEY =1D80         SAVXY =1E18
SECDIS =2301        SECMAX =1FF2        SECMSG =21EF
SET4 =2364          SETUP =21C7         SETUP2 =21CD
SETY =20BD          SORRY =2167         SP =0020
SPACE1 =1DB2        SPACE2 =1DAF        SPLIT =0010
START =1CFC         STASEC =1FE8        STCHK =2078
STOFIL =243E        STOPMS =2286        STOSTA =2527
STSECH =2374        STSECL =2373        STUFIT =1E57
SUBPER =1DDB        SUCCES =26CC        TAB =007F
TEMP =1E64          TEMP1 =22FA         TEMP2 =22FB
TEMPX =22FC         TEMPY =22FD         TOGGLE =1FF1
TOPSEC =1D83        TRASEC =2020        UPDATE =1E06
VBAD =26E3          VERIFY =2459        VRO =246E
VBOON =2479         VOUTOK =25C4        VRBAD =26FB
VTOC =2378          WEDONE =2140        WEIRD =2010
WFINE =2556         WFLAG =1D82         WRITE =0008
XCNT =22F8          YCNT =22F9
//0000,2759,B35D
```

Listing 5.

```
10 REM ********************************
15 REM *  DISK TOOL BASIC PROGRAM  *
20 REM *   BY TONY MESSINA (C) 1982*
25 REM ********************************
30 POKE 82,0:REM **LFT MAR TO 0 ***
35 REM
40 REM ********************************
45 REM * VARIABLE/CONSTANT/STRINGS *
50 REM *  INITIALIZATION FOLLOWS   *
55 REM ********************************
60 REM *      CONSTANTS FOLLOW       *
65 REM * THE FOLLOWING CONSTANTS    *
70 REM * ARE USED TO MAKE THE DSK- *
75 REM * TOOL LISTING EASIER TO     *
80 REM * READ. IT ALSO PROVIDES A   *
85 REM * CENTRAL LOCATION FOR ISO- *
90 REM * LATING ADDRESS POINTERS IN*
95 REM * ORDER TO MAKE MODIFICATION*
100 REM * OF THE PROGRAM EASIER.    *
105 REM * ENTRIES ARE ALPHABETICAL. *
110 REM * SEE THE CONSTANTS DESCRIP-*
115 REM * TION SECTION OF THE DOCU- *
120 REM * MENTATION FOR DEFINITIONS.*
125 REM ********************************
130 REM
135 BACKGND=710:BLACK=0:BORDER=712:BUF
HI=773:BUFLO=772:BUFPTR=126:CASBUF=102
1:CASPTR=CASBUF+BUFPTR
140 CHAR=709:CHNGBY=7729:CKLIM=475:CKR
OLM=590:CKROLP=550:CLIOC4=9023:CNTIN=8
958:DAUX1=778:DAUX2=779
145 DBYHI=777:DBYLO=776:DCOMM=770:DECH
EX=1360:DUNIT=769:ENTLNK=8180:ERRFLG=7
550:ERTRAP=625:FILNUM=7556
150 FINUMB=8179:GREEN=214:GSEC=82:HEXD
EC=1270:HILO=515:MESAGE=7430:NWLNK=819
4:PCHANGE=1085:PDIR=1420
155 PHELP=1190:PLUSMIN=865:PMOD=1645:P
PRINT=1530:PRECOVER=1930:PROCINP=745:P
SEC=87:PSET=2030:PTRACE=1795:PWRITE=95
0
160 RECOVR=9877:RED=64:REDIR=7823:SCRO
LL=660:SETDSK=400:SETSCRN=705:START=74
20:TOPSEC=7555:TRASEC=8224
165 TURQ=186:WFLAG=7554:WHITE=10:YELLO
W=26
170 REM
175 REM ********************************
180 REM *      VARIABLES FOLLOW       *
```

```
185 REM * THE FOLLOWING ARE VARIA-  *
190 REM * BLES SET TO THEIR DEFAULT *
195 REM * VALUES INDICATED DURING   *
200 REM * PROGRAM INITIALIZATION.SEE*
205 REM * VARIABLE DESCRIPTION SEC- *
210 REM * TION OF DOCUMENTATION FOR *
215 REM * LIST OF ALL VARIABLES AND *
220 REM * THEIR PURPOSE.            *
225 REM ****************************
230 REM
235 DRIVE=1:HELP=1:SECHI=0:SECLOW=1:SE
CNUM=1
240 REM
245 REM ****************************
250 REM *   STRING INIT FOLLOWS    *
255 REM * SEE STRING DESCRIPTION   *
260 REM * SECTION OF DOCUMENTATION *
265 REM * FOR DESCRIPTION AND USES *
270 REM * OF THE FOLLOWING STRINGS.*
275 REM ****************************
280 REM
285 DIM A$(40),AN$(1),HEXREP$(4),HEXTA
B$(16)
290 HEXTAB$="0123456789ABCDEF"
295 REM
300 REM ****************************
305 REM * VARIABLE/CONSTANT/STRING *
310 REM *   INITIALIZATION END     *
315 REM ****************************
320 REM
325 TRAP ERTRAP
330 GOSUB SETDSK
335 X=USR(CLIOC4):REM CLR IOCB 4
340 GOTO PHELP
345 REM
350 REM ****************************
355 REM *COMMON PROGRAM SUBROUTINES *
360 REM ****************************
365 REM
370 REM ****************************
375 REM * SETDSK:                  *
380 REM * SET UP DISK VECTOR TABLE *
385 REM ****************************
390 REM
395 REM
400 POKE DUNIT,DRIVE:REM ** DRIVE #
405 POKE DCOMM,GSEC:REM ** FOR READ
410 POKE DAUX1,SECLOW
415 POKE DAUX2,SECHI
420 POKE BUFLO,253:REM ** LOW BUF ADR
($FD)**
425 POKE BUFHI,3:REM **HI BUF ADR ($03
) **
430 POKE DBYLO,127:REM ** GET 128 BYTE
S (1 SECTOR) **
435 POKE DBYHI,0:REM ** NO HI **
440 RETURN
445 REM
450 REM ****************************
455 REM * CKLIM:                   *
460 REM * CK SECNUM LIMITS (1 )720  *
465 REM ****************************
470 REM
475 SECNUM=VAL(A$):IF SECNUM<1 OR SECN
UM>720 THEN ? "      INVALID SECTOR. RA
NGE IS (1-720)":POP :GOTO PROCINP
480 RETURN
485 REM
490 REM ****************************
495 REM * HILO:                    *
500 REM *BREAK SECNUM TO HI/LO FORM*
505 REM ****************************
510 REM
515 SECHI=INT(SECNUM/256):SECLOW=INT(S
ECNUM-(SECHI*256)):RETURN
520 REM
525 REM ****************************
526 REM * CKROLP:                  *
530 REM * ROLL SECTOR NUM TO 1 IF *
535 REM * )720                     *
540 REM ****************************
545 REM
550 IF SECNUM>720 THEN SECNUM=1
555 RETURN
560 REM
565 REM ****************************
566 REM * CKROLM:                  *
570 REM * ROLL SECTOR NUM TO 720  *
575 REM * ( 1                      *
580 REM ****************************
585 REM
590 IF SECNUM<1 THEN SECNUM=720
595 RETURN
600 REM
605 REM ****************************
606 REM * ERTRAP:                  *
610 REM *    GO HERE ON ERROR     *
615 REM ****************************
620 REM
625 ? "▶ILLEGAL INPUT!!":POKE CHAR,BLA
CK:POKE BACKGND,GREEN:TRAP ERTRAP:GOTO
 PROCINP
630 REM
635 REM ****************************
636 REM * SCROLL:                  *
640 REM *      SCROLL 5 LINES      *
645 REM * FROM THE BOTTOM OF SCRN *
650 REM ****************************
655 REM
660 POSITION 0,17:? "↑↑↑↑↑":POSITION 0
,17:RETURN
665 REM
670 REM ****************************
671 REM * SETSCRN:                 *
675 REM *       SET SCREEN TO      *
680 REM * DEFAULT COLORS OF GREEN *
685 REM * BACKGND,WHITE BORD,BLACK*
690 REM * LETTERS.                 *
695 REM ****************************
700 REM
705 POKE BORDER,WHITE:POKE CHAR,BLACK:
POKE BACKGND,GREEN:RETURN
710 REM
715 REM ****************************
720 REM * PROCINP:                 *
725 REM * MAIN COMMAND/INPUT PROC- *
730 REM * ESSING PORTION FOLLOWS   *
735 REM ****************************
740 REM
745 ? "▶     CURRENT DRIVE= ";DRIVE:? "
▶COMMAND OR SECTOR NUMBER";:INPUT A$
750 IF A$="H" THEN HELP=1:GOTO PHELP
755 IF A$="P" THEN GOSUB PPRINT:GOTO P
ROCINP
760 IF A$="T" THEN GOTO PTRACE
765 HELP=0
770 IF A$="+" OR A$="" THEN SECNUM=SEC
NUM+1:GOSUB CKROLP:GOTO PLUSMIN
775 IF A$(1,1)="-" THEN SECNUM=SECNUM-
1:GOSUB CKROLM:GOTO PLUSMIN
780 IF A$="W" THEN GRAPHICS 0:GOSUB PW
RITE:GOTO PROCINP
785 IF A$="C" THEN GRAPHICS 0:GOSUB PC
HANGE:GOTO PROCINP
790 IF A$="D" THEN GOTO PDIR
795 IF A$="T" THEN GOTO PTRACE
800 IF A$="M" THEN GRAPHICS 0:GOSUB SE
TSCRN:GOTO PMOD
805 IF A$="R" THEN GOTO PRECOVER
810 IF A$="S" THEN GRAPHICS 0:GOSUB SE
TSCRN:GOTO PSET
815 IF A$(1,1)="$" THEN GOSUB HEXDEC
820 GOSUB CKLIM
825 REM
830 REM ****************************
835 REM * PLUSMIN:                 *
840 REM * PROCESS THE (+) (-) OR   *
845 REM * NUMERIC INPUT. PRINT SEC-*
850 REM * TOR DISPLAY & RTN TO 370 *
855 REM ****************************
860 REM
865 GRAPHICS 0:GOSUB SETSCRN:GOSUB HIL
O:GOSUB DECHEX
870 POKE DAUX1,SECLOW:POKE DAUX2,SECHI
875 X=USR(START):REM ** GO DO IT **
880 IF PEEK(ERRFLG)=138 THEN ? "DRIVE
";DRIVE;" DOES NOT RESPOND!!"
885 IF PEEK(ERRFLG) THEN ? "CAN'T READ
 SECTOR";SECNUM;" ($";HEXREP$;")":POKE
 ERRFLG,0:GOTO PROCINP
886 IF SECNUM=360 THEN ? "VTOC SECTOR=
>360 ($0168)";:? "CREATED DOS ";PEEK(
CASBUF):? "FREE SECTORS=>";
```

```
887 IF SECNUM=360 THEN ? PEEK(CASBUF+3
)+PEEK(CASBUF+4)*256:GOTO PROCINP
890 IF SECNUM<369 AND SECNUM>360 THEN
? "▶  DIRECTORY SECTOR ";SECNUM;" ($";
HEXREP$;")":GOTO PROCINP
895 ? "SECTOR ==▶";SECNUM;" ($";HEXREP
$;")▶NEXT SEC==▶";(PEEK(TOPSEC)*256)+P
EEK(CASPTR)
900 ? "FILE#==▶";PEEK(FILNUM)
905 GOTO PROCINP
910 REM
915 REM ******************************
920 REM * PWRITE:                     *
925 REM * PROCESS (W) COMMAND TO ? *
930 REM * SECTOR REQUESTED TO DISK.*
935 REM * WRITE ONLY AFTER VERIFY..*
940 REM ******************************
945 REM
950 POKE BACKGND,RED:POKE CHAR,WHITE:P
OKE BORDER,WHITE
955 X=USR(MESAGE)
960 GOSUB SCROLL:GOSUB HILO:GOSUB DECH
EX
965 ? "↖CURRENT SECTOR IS ==▶";SECNUM;
" ($";HEXREP$;")"
970 ? "↖CURRENT DRIVE IS  ==▶";DRIVE
975 ? "↖SURE ABOUT WRITE(Y/N)";:INPUT
AN$
980 IF AN$<>"Y" AND AN$<>"N" THEN ? "▶
DANGEROUS INPUT!!":GOTO 1025
985 IF AN$="N" THEN ? "WRITE ABORTED↖↖
":GOTO 1025
990 POKE DCOMM,PSEC:POKE WFLAG,1
995 X=USR(START)
1000 IF PEEK(ERRFLG)=138 THEN ? "DRIVE
 ";DRIVE;" DOES NOT RESPOND!↖":GOTO 10
20
1005 IF PEEK(ERRFLG)=144 THEN ? "DISK
IN DRIVE ";DRIVE;" IS WRITE PROTECTED!
↖":GOTO 1020
1010 IF PEEK(ERRFLG) THEN ? "ERROR - "
;PEEK(ERRFLG);" -↖":GOTO 1020
1015 ? "SECTOR ";SECNUM;" WRITTEN"
1020 POKE ERRFLG,0:POKE DCOMM,GSEC:POK
E WFLAG,0
1025 POKE CHAR,BLACK:POKE BACKGND,GREE
N
1030 AN$=""
1035 RETURN
1040 REM
1045 REM ******************************
1050 REM * PCHANGE:                    *
1055 REM * PROCESS (C) COMMAND TO   *
1060 REM * CHANGE BYTES OF SECTOR IN*
1065 REM * CURRENT BUFFER.....       *
1070 REM ******************************
1075 REM
1080 REM *** CHANGE BYTES ROUTINE ***
1085 POKE BACKGND,YELLOW:POKE CHAR,BLA
CK:POKE BORDER,WHITE
1090 X=USR(MESAGE)
1095 GOSUB SCROLL
1100 ? "↖MOVE CURSOR TO BYTES,CHANGE,H
IT RETURN"
1105 INPUT A$
1110 IF A$="" OR LEN(A$)<26 THEN ? "IL
LEGAL INPUT!!PRESS RETURN";:GOTO 1135
1115 POKE BACKGND,YELLOW
1120 LINBUF=ADR(A$)
1125 X=USR(CHNGBY,LINBUF)
1130 ? "DATA CHANGED--HIT RETURN TO CO
NTINUE↖↖";
1135 INPUT A$
1140 POKE BACKGND,GREEN
1145 RETURN
1150 REM
1155 REM ******************************
1160 REM * PHELP:                      *
1165 REM * PROCESS (H) COMMAND BY   *
1170 REM * DISPLAYING THE COMMANDS  *
1175 REM * AVAILABLE IN DISK-TOOL...*
1180 REM ******************************
1185 REM
1190 GRAPHICS 1:POKE BACKGND,GREEN:? #
6;"                    ":? #6;" disk t
ool commands"
1195 ? #6;"                    "
1200 ? #6;"+=READ NEXT SECTOR":? #6;"-
=READ PREVIOUS SEC":? #6;"C=CHANGE SEC
 BYTES"
1205 ? #6;"D=DIRECTORY LIST":? #6;"H=H
ELP"
1210 ? #6;"L=MODIFY LINKS":? #6;"P=PRI
NT SCREEN"
1215 ? #6;"R=RECOVER A FILE":? #6;"S=S
ET DRIVE #"
1220 ? #6;"T=TRACE FILE CHAIN":? #6;"W
=WRITE A SECTOR"
1225 ? "↖":GOTO PROCINP
1230 REM
1235 REM ******************************
1240 REM * HEXDEC:                     *
1245 REM * HEX-DEC CONVERSION SUBRTN*
1250 REM * HEX INPUT FROM A$ IN THE *
1255 REM * FORM $XXXX. OUTPUT DEC A$*
1260 REM ******************************
1265 REM
1270 N=0
1275 FOR I=2 TO LEN(A$)
1280 IF A$(I,I)<"0" THEN GOTO 1310
1285 IF A$(I,I)<="9" THEN N=N*16+VAL(A
$(I,I)):GOTO 1300
1290 IF A$(I,I)<"A" OR A$(I,I)>"F" THE
N 1310
1295 N=N*16+ASC(A$(I,I))-ASC("A")+10
1300 NEXT I
1305 A$=STR$(N):RETURN
1310 ? "INVALID HEX PARAMETER":POP :GO
TO PROCINP
1315 REM
1320 REM ******************************
1325 REM * DECHEX:                     *
1330 REM * DEC-HEX CONVERSION SUBRTN*
1335 REM * HI/LO OF NUMBER IN SECLOW*
1340 REM * & SECHI. HEX OUTPUT IN    *
1345 REM * HEXREP$.................. *
1350 REM ******************************
1355 REM
1360 TSECH=SECHI:SECHI=INT(SECHI/16)+1
:HEXREP$(1,1)=HEXTAB$(SECHI,SECHI)
1365 SECHI=(TSECH-(SECHI-1)*16)+1:HEXR
EP$(2,2)=HEXTAB$(SECHI,SECHI):SECHI=TS
ECH
1370 TSECL=SECLOW:SECLOW=INT(SECLOW/16
)+1:HEXREP$(3,3)=HEXTAB$(SECLOW,SECLOW
)
1375 SECLOW=(TSECL-(SECLOW-1)*16)+1:HE
XREP$(4,4)=HEXTAB$(SECLOW,SECLOW):SECL
OW=TSECL:RETURN
1380 REM
1385 REM ******************************
1390 REM * PDIR:                       *
1395 REM * PROCESS (D) COMMAND TO   *
1400 REM * DISPLAY FORMATTED DISK   *
1405 REM * DIRECTORY/FILE INFO..    *
1410 REM ******************************
1415 REM
1420 POKE DAUX2,1:POKE DAUX1,105:REM *
* SET SECTOR 361 FOR READ
1425 SECNUM=361
1430 GRAPHICS 0:GOSUB SETSCRN
1435 ? "↖SECH# FILENAME/EXT START LENGT
H FIL# STAT"
1440 X=USR(REDIR)
1445 SECNUM=SECNUM+1
1450 ? :? "     HIT RETURN TO STOP, + T
O CONT.↖↖";
1455 INPUT AN$
1460 IF AN$="+" AND SECNUM<365 THEN 14
35
1465 AN$="":POKE FINUMB,0
1470 ? "▶COMMAND OR SECTOR NUMBER";
1475 INPUT A$
1480 IF A$="" THEN GOTO 1490
1485 IF A$(1,1)="W" OR A$(1,1)="C" THE
N ? "▶IMPROPER SCREEN CONDITIONAL↖":GOT
O 1470
1490 GOTO PROCINP+5
1495 REM
1500 REM ******************************
1505 REM * PPRINT:                     *
1510 REM * PROCESS (P) COMMAND TO ? *
1515 REM * THE SCREEN TO PRINTER..  *
1520 REM ******************************
1525 REM
```

```
1530 IF (HELP) THEN ? "▶IMPROPER SCREE
N CONDITIONS◀":RETURN
1535 TRAP 1605:LPRINT :LPRINT
1540 ? "▶    PRINTING SCREEN!!!":POKE B
ACKGND,TURQ
1545 SCAND=PEEK(88)+PEEK(89)*256
1550 REM **PRINT SCREEN TO PRINTER **
1555 FOR X=1 TO 19
1560 ARPT=1
1565 FOR Y=SCAND TO SCAND+39
1570 A$(ARPT,ARPT)=CHR$((ASC(" ")+PEEK
(Y))):TEMP=ASC(A$(ARPT,ARPT))
1575 IF TEMP>128 THEN TEMP=TEMP-128:A$
(ARPT,ARPT)=CHR$(TEMP)
1580 IF A$(ARPT,ARPT)<" " OR A$(ARPT,A
RPT)>"Z" THEN A$(ARPT,ARPT)="."
1585 ARPT=ARPT+1:NEXT Y
1590 IF A$(20,21)="DR" THEN GOTO 1600
1595 LPRINT A$:SCAND=SCAND+40:NEXT X
1600 POKE BACKGND,GREEN:TRAP ERTRAP:A$
="":GOSUB SETDSK:RETURN
1605 ? "▶PRINTER DOESN'T RESPOND!!◀":
GOTO 1600
1610 REM
1615 REM ******************************
1620 REM * PMOD:                       *
1625 REM * PROCESS (M) COMMAND TO      *
1630 REM * MODIFY LINKS OF A FILE..    *
1635 REM ******************************
1640 REM
1645 ? "↰          MODIFY SECTOR LINKS":
? :? "SECTOR TO MODIFY  (HEX OR DEC)";
:INPUT A$
1650 IF A$="" THEN GOTO PROCINP
1655 IF A$(1,1)="$" THEN GOSUB HEXDEC
1660 GOSUB CKLIM
1665 GOSUB HILO:GOSUB SETDSK:GOSUB DEC
HEX
1670 X=USR(ENTLNK)
1675 IF PEEK(ERRFLG) THEN GOTO 885
1680 ? :? "FILE#=▶";PEEK(FILNUM);":NEX
T SECTOR=▶ ";(PEEK(TOPSEC)*256)+PEEK(C
ASPTR)
1685 ? :? "ENTER NEW FILE (DEC OR HEX)
";:INPUT A$
1690 IF A$="" THEN FIL=PEEK(FILNUM):GO
TO 1705
1695 IF A$(1,1)="$" THEN GOSUB HEXDEC
1700 FIL=VAL(A$)
1705 POKE (FILNUM),FIL:? :? "ENTER NEW
 SECH (HEX OR DEC)";:INPUT A$
1710 IF A$="" THEN GOTO PROCINP
1715 IF A$(1,1)="$" THEN GOSUB HEXDEC
1720 IF VAL(A$)=0 THEN SECNUM=0:GOTO 1
730
1725 GOSUB CKLIM
1730 GOSUB HILO:POKE (TOPSEC),SECHI:PO
KE CASPTR,SECLOW
1735 X=USR(NWLNK)
1740 ? "▶LINKS CHANGED!!";:? :? "▶NEW F
ILE= ";PEEK(FILNUM);"     NEW SECTOR= "
;(PEEK(TOPSEC)*256)+PEEK(CASPTR)
1745 SECNUM=PEEK(DAUX1)+PEEK(DAUX2)*25
6
1750 ? :? "  WRITE TO DISK IF CHANGES
CORRECT":? :GOTO PROCINP
1755 REM
1760 REM ******************************
1765 REM * PTRACE:                     *
1770 REM * PROCESS (T) COMMAND TO      *
1775 REM * TRACE THE SECTORS OF A      *
1780 REM * FILE FOR FILE INTEGRITY     *
1785 REM ******************************
1790 REM
1795 IF (HELP) THEN GRAPHICS 0:HELP=0:
GOSUB SETSCRN
1800 ? "▶INPUT FILE NUMBER (HEX OR DEC
)";:? "▶TO TRACE OR X TO ABORT";:INPUT
A$
1805 IF A$(1,1)="X" THEN GOTO PROCINP
1810 IF A$(1,1)="$" THEN GOSUB HEXDEC
1815 FIN=VAL(A$):STSEC=INT(FIN/8)+361:
IF STSEC<361 OR STSEC>368 THEN ? "▶BAD
 FILE NUMBER◀":GOTO PROCINP
1820 RELFI=FIN-(INT(FIN/8)*8)
1825 IDX=(RELFI*16)+CASBUF
1830 ? "↰          SECTOR TRACE"
1835 X=USR(TRASEC,STSEC,IDX,FIN)
1840 IF PEEK(ERRFLG) THEN POKE ERRFLG,
0:POKE FINUMB,0:? :GOTO PROCINP
1845 SECLOW=PEEK(CASPTR):SECHI=PEEK(TO
PSEC)
1850 IF  NOT PEEK(CNTIN) THEN POKE (FI
NUMB),0:GOTO PROCINP
1855 ? :? "+ TO CONTIN PRINT SCRN <RET
> TO STOP";:INPUT AN$
1860 IF AN$="" THEN POKE CNTIN,0:POKE
FINUMB,0:GOTO PROCINP
1865 IF AN$="P" THEN GOSUB PPRINT:GOTO
 1855
1870 IF AN$<>"+" THEN POKE CNTIN,0:POK
E FINUMB,0:GOTO ERTRAP
1875 ? "↰▶     SECTOR TRACE (CONT)"
1880 X=USR(TRASEC)
1885 GOTO 1840
1890 REM
1895 REM ******************************
1900 REM * PRECOVER:                   *
1905 REM *PROCESS (R) COMMAND WHICH *
1910 REM *WILL RECOVER A FILE WHICH *
1915 REM *HAS BEEN DELETED          *
1920 REM ******************************
1925 REM
1930 IF (HELP) THEN GRAPHICS 0:HELP=0:
GOSUB SETSCRN
1935 ? "▶INPUT FILE NUMBER (HEX OR DEC
)";:? "▶TO RECOVER OR X TO ABORT";:INPU
T A$
1940 IF A$(1,1)="X" THEN GOTO PROCINP
1945 IF A$(1,1)="$" THEN GOSUB HEXDEC
1950 FIN=VAL(A$):STSEC=INT(FIN/8)+361:
IF STSEC<361 OR STSEC>368 THEN ? "▶BAD
 FILE NUMBER◀":GOTO PROCINP
1955 RELFI=FIN-(INT(FIN/8)*8)
1960 IDX=(RELFI*16)+CASBUF
1965 ? "↰          RECOVER FILE"
1970 POKE RECOVR,1
1975 X=USR(TRASEC,STSEC,IDX,FIN)
1980 IF PEEK(ERRFLG) THEN POKE ERRFLG,
0:POKE RECOVR,0:?
1985 POKE FINUMB,0:GOTO PROCINP
1990 REM
1995 REM ******************************
2000 REM * PSET:                       *
2005 REM *PROCESS (S) COMMAND WHICH *
2010 REM *ALLOWS USER TO CHANGE THE *
2015 REM *WORKING DISK DRIVE NUMBER *
2020 REM ******************************
2025 REM
2030 ? "▶    SET DRIVE NUMBER"
2035 ? :? "▶CURRENT DRIVE IS =";DRIVE:
?
2040 ? "▶INPUT NEW DRIVE (1-4) ";:INPU
T A$
2045 IF A$="" THEN GOTO PROCINP
2050 X=VAL(A$):IF X<1 OR X>4 THEN ? "▶
INVALID DRIVE NUMBER!◀":GOTO PROCINP
2055 DRIVE=X:GOSUB SETDSK:? :GOTO PROC
INP
```

●

CHECKSUM DATA

(See pgs. 7-10)

```
10 DATA 532,703,450,549,210,270,538,17
9,914,555,376,887,737,641,650,8191
85 DATA 855,191,95,807,148,236,48,360,
789,83,25,952,556,196,492,5833
160 DATA 642,799,95,804,557,84,166,76,
445,181,335,152,404,791,85,5616
235 DATA 51,88,558,647,870,99,861,9,56
7,100,90,792,108,543,23,5406
310 DATA 549,551,84,322,598,212,91,95,
797,540,800,101,564,576,704,6584
385 DATA 572,105,110,698,738,80,820,27
9,753,491,92,599,97,799,256,6489
460 DATA 796,807,101,257,611,109,572,7
43,980,552,85,856,88,361,389,7307
```

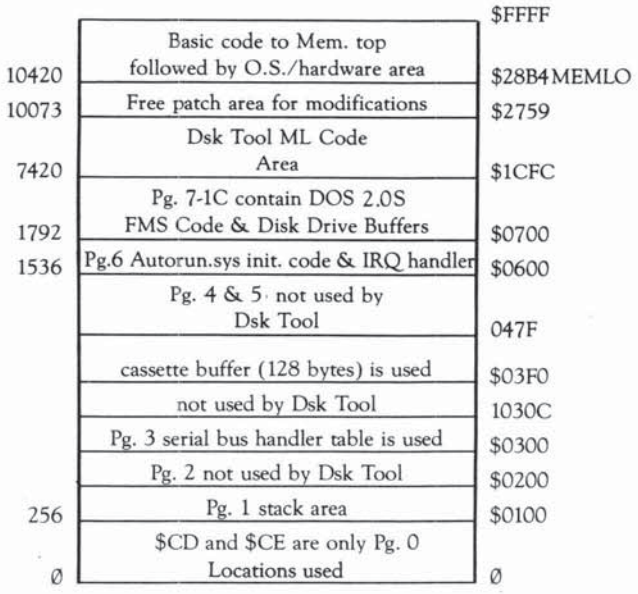
```
530 DATA 424,61,362,99,402,609,100,373
,395,365,994,374,111,514,621,5804
600 DATA 84,357,400,137,360,90,563,93,
366,426,847,509,367,104,626,5329
665 DATA 107,373,668,817,620,661,612,3
84,86,240,89,559,771,812,845,7644
735 DATA 565,98,180,704,910,950,688,99
1,728,677,415,805,964,983,204,9862
810 DATA 26,536,251,99,562,823,339,978
,539,573,106,530,364,685,652,7063
885 DATA 978,120,336,189,907,762,645,9
3,563,577,828,1,23,567,107,6696
950 DATA 658,444,958,166,775,385,533,4
92,471,648,84,189,76,285,310,6474
1025 DATA 363,200,792,279,796,756,708,
155,505,789,292,867,212,552,369,7635
1100 DATA 824,915,225,436,724,741,195,
918,166,796,283,800,264,673,15,7975
1175 DATA 989,793,296,456,964,185,508,
512,330,541,25,284,801,459,361,7504
1250 DATA 865,982,794,297,994,837,84,6
74,584,216,493,52,877,295,793,8837
1325 DATA 470,315,9,793,329,796,299,53
6,426,321,191,292,809,777,722,7085
1400 DATA 949,829,795,298,964,91,61,81
0,269,786,780,811,945,326,698,9412
1475 DATA 931,507,348,891,306,797,560,
815,954,799,302,813,92,155,821,9091
1550 DATA 180,399,480,825,450,851,286,
244,375,914,459,152,294,811,801,7521
1625 DATA 733,947,813,297,952,179,583,
434,715,624,566,467,171,583,587,8651
1700 DATA 309,455,178,582,39,443,613,3
34,577,125,246,311,809,505,738,6264
1775 DATA 704,942,821,305,807,789,730,
575,239,679,425,775,387,130,348,8656
1850 DATA 68,764,584,898,816,291,562,7
54,308,825,358,230,203,398,811,7870
1925 DATA 314,797,229,727,591,236,695,
422,696,539,394,231,592,311,828,7602
2000 DATA 781,208,18,230,786,289,784,2
87,669,172,306,731,5261
```

Cross Reference of Disk Tool BASIC program

VAR	LINE NUMBERS
BACKGND	135, 625, 705, 950, 1025, 1085, 1115, 1140, 1190, 1540, 1600
BLACK	135, 625, 705, 1025, 1085
BORDER	135, 705, 950, 1085
BUFHI	135, 425
BUFLO	135, 420
BUFPTR	135
CASBUF	135, 886, 887, 1825, 1960
CASPTR	135, 895, 1680, 1730, 1740, 1845
CHAR	140, 625, 705, 950, 1025, 1085
CHNGBY	140, 1125
CKLIM	140, 820, 1660, 1725
CKROLM	140, 775
CKROLP	140, 770
CLIOC4	140, 335
CNTIN	140, 1850, 1860, 1870
DAUX1	140, 410, 870, 1420, 1745
DAUX2	140, 415, 870, 1420, 1745
DBYHI	145, 435
DBYLO	145, 430
DCOMM	145, 405, 990, 1020
DECHEX	145, 865, 960, 1665
DUNIT	145, 400
ENTLNK	145, 1670
ERRFLG	145, 880, 885, 1000, 1005, 1010, 1020, 1675, 1840, 1980
ERTRAP	145, 325, 625, 1600, 1870
FILNUM	145, 900, 1680, 1690, 1705, 1740
FINUMB	150, 1465, 1840, 1850, 1860, 1870, 1985
GREEN	150, 625, 705, 1025, 1140, 1190, 1600
GSEC	150, 405, 1020
HEXDEC	150, 815, 1655, 1695, 1715, 1810, 1945
HILO	150, 865, 960, 1665, 1730
MESAGE	150, 955, 1090
NWLNK	150, 1735
PCHANGE	150, 785
PDIR	150, 790
PHELP	155, 340, 750
PLUSMIN	155, 770, 775
PMOD	155, 800
PPRINT	155, 755, 1865
PRECOVER	155, 805
PROCINP	155, 475, 625, 755, 780, 785, 885, 887, 890, 905, 1225, 1310, 1490, 1750, 1805, 1815, 1840, 1850, 1860, 1940, 1950, 1985, 2050, 2055
PSEC	155, 990
PSET	155, 810
PTRACE	155, 760, 795
PWRITE	155, 780
RECOVR	160, 1970, 1980
RED	160, 950
REDIR	160, 1440
SCROLL	160, 960, 1095
SETDSK	160, 330, 1600, 1665, 2055
SETSCRN	160, 800, 810, 865, 1430, 1795, 1930
START	160, 875, 995
TOPSEC	160, 895, 1680, 1730, 1740, 1845
TRASEC	160, 1835, 1880, 1975
TURQ	165, 1540
WFLAG	165, 990, 1020
WHITE	165, 705, 950, 1085
YELLOW	165, 1085, 1115
DRIVE	235, 400, 745, 880, 970, 1000, 1005, 2035, 2055
HELP	235, 750, 765, 1530, 1795, 1930
SECHI	235, 415, 515, 870, 1360, 1365, 1730, 1845
SECLOW	235, 410, 515, 870, 1370, 1375, 1730, 1845
SECNUM	235, 475, 515, 550, 590, 770, 775, 885, 886, 887, 890, 895, 965, 1015, 1425, 1445, 1460, 1720, 1745
A$	285, 475, 745, 750, 755, 760, 770, 775, 780, 785, 790, 795, 800, 805, 810, 815, 1105, 1110, 1120, 1135, 1275, 1280, 1285, 1290, 1475, 1480, 1485, 1570, 1575, 1580, 1590, 1595, 1600, 1645, 1650, 1655, 1685, 1690, 1695, 1700, 1705, 1710, 1715, 1720, 1800, 1805, 1810, 1815, 1935, 1940, 1945, 1950, 2040, 2045, 2050
AN$	285, 975, 980, 985, 1030, 1455, 1460, 1465, 1855, 1860, 1865, 1870
HEXREP$	285, 885, 890, 895, 965, 1360, 1365, 1370, 1375
HEXTAB$	285, 290, 1360, 1365, 1370, 1375
X	335, 875, 955, 995, 1090, 1125, 1440, 1555, 1595, 1670, 1735, 1835, 1880, 1975, 2050, 2055
LINBUF	1120, 1125
N	1270, 1285, 1295, 1305
I	1275, 1280, 1285, 1290, 1295, 1300
TSECH	1360, 1365
TSECL	1370, 1375
SCAND	1545, 1565, 1595
ARPT	1560, 1570, 1575, 1580, 1585
Y	1565, 1570, 1585
TEMP	1570, 1575

```
FIL             1690, 1700, 1705
FIN             1815, 1820, 1835, 1950, 1955, 1975
STSEC           1815, 1835, 1950, 1975
RELFI           1820, 1825, 1955, 1960
IDX             1825, 1835, 1960, 1975

# OF VARIABLES= 82
```

Disk Tool Memory Map (after fully loaded)



Constant Description List DSKTOOL.PT2

ID	NAME	VAL	DESCRIPTION
1	BACKGND	710	Background color register address (REGISTER 2)
	BLACK	0	Color value for black
1	BORDER	712	Border color register address (REGISTER 4)
2	BUFHI	773	Address of Disk buffer Pointer MSB
	BUFLO	772	Address of Disk buffer Pointer LSB
	BUFPTR	126	Value set to 126. Byte 126 of CASBUF contains the LSB of the next sector number of the file being examined.
	CASBUF	1021	Pointer to the start of the cassette buffer.
	CASPTR	CASBUF+	CASPTR Points to the absolute address in CASBUF of the LSB of the next sector number for the current file.
1	CHAR	709	Character color register address (REGISTER 0)
*	CHNGBY	7729	Absolute address to beginning of Change Byte ML code.
	CKLIM	475	Line number to Basic routine.
	CKROLM	590	Line number to Basic routine.
	CKROLP	550	Line number to Basic routine.
*	CLIOC4	9023	Absolute address to ML routine which sets up IOCB4 for ML output of messages etc.
*	CNTIN	8958	Absolute address to continue flag. Tells ML code if we are continuing a directory dump or sector trace.
3	DAUX1	778	Address to disk AUX value LSB.
	DAUX2	779	Address to disk AUX value MSB.

4	DBYHI	777	Address to disk byte count MSB.
	DBYLO	776	Address to disk byte count LSB.
	DCOMM	770	Address to disk command byte location. Commands used by DISK TOOL are PUT SETOR with verify and GET SECTOR.
	DECHEX	1360	Line number to Basic routine.
	DUNIT	769	Address to disk unit number. Location contains current drive being accessed.
*	ENTLNK	8180	Absolute address to the Change links ML code.
*	ERRFLG	7550	Absolute address to the error flag indicator in ML code. Flag is set by ML code to indicate any errors. Basic checks the flag to determine appropriate message.
	ERTRAP	625	Line number to Basic routine.
*	FILNUM	7556	Absolute address to ML location. Location contains the file number to which the current sector belongs.
*	FINUMB	8179	Absolute address to ML location. Location contains the directory file number of a file $0-$3F.
	GREEN	214	Value for the color green.
	GSEC	82	GET SECTOR disk command value.
	HEXDEC	1275	Line number to Basic routine.
	HILO	515	Line number to Basic routine.
*	MESAGE	7430	Absolute address to ML code which displays items in CASBUF in HEX/ATASCI format.
*	NWLNK	8194	Absolute address to ML code which changes links during a modify link operation.
	PCHANGE	1085	Line number to Basic routine.
	PDIR	1420	Line number to Basic routine.
	PHELP	1190	Line number to Basic routine.
	PLUSMIN	865	Line number to Basic routine.
	PMOD	1645	Line number to Basic routine.
	PPRINT	1530	Line number to Basic routine.
	PRECOVER	1930	Line number to Basic routine.
	PROCINP	745	Line number to Basic routine.
	PSEC	87	Disk command value for a PUT SECTOR with verify.
	PSET	2030	Line number to Basic routine.
	PTRACE	1795	Line number to Basic routine.
	PWRITE	950	Line number to Basic routine.
*	RECOVR	9077	Absolute address in ML code of recover flag. Used by ML code to distinguish a recover file from a trace file.
	RED	64	Value for the color red.
*	REDIR	7823	Absolute address in ML code to the read directory function.
	SCROLL	660	Line number to Basic routine.
	SETDSK	400	Line number to Basic routine.
	SETSCRN	705	Line number to Basic routine.
	START	7420	Absolute address to the start of DISK TOOL ML code.
*	TOPSEC	7555	Absolute address to ML location. Location contains the MSB of the sector number currently being examined.
*	TRASEC	8224	Absolute address to the start of the TRACE SECTOR ML code.
	TURQ	186	Value for the color turquoise.
*	WFLAG	7554	Absolute address to ML Write flag location. Informs the ML code if the next operation is a read or write. 1=write
	WHITE	10	Value for the color white.
	YELLOW	26	Value for the color yellow.

ID EXPLANATIONS

1= References hold true for Graphics 0. Other modes have different meanings. If confusion exists, see color register assignment table in the ATARI BASIC Reference Manual.

2= These locations point to an area in memory where we want the data on a disk sector to be placed after a read. On a disk write, these locations point to the area of memory which contains the data to be written. Disk Tool sets these pointers to the cassette buffer since it is free when using the disk drive.

3= Locations contain sector number (LSB/MSB format) of sector to read or write.

4= Locations contain number of bytes (LSB/MSB format) to be read or written.

*= Point to absolute locations in the ML code. In most cases the Basic constant name is the same as the label name in the assembly source code. Exceptions are CNTIN for CONTIN and NWLNK for NEWLNK due to BASIC not accepting CONTIN and NEWLNK. I could have used the LET statement but. . . NAAHH!!!

VARIABLE DESCRIPTION LIST

ARPT	Pointer to each item in A$
DRIVE	Current Disk Drive being used.
FIL	Temporary file number variable.
FIN	File number input for TRACE or RECOVER.
HELP	Help flag. 1=Help menu is up. 0=Help menu not up. Prevents printing the help screen since it is in the wrong graphics mode for the Print routine.
IDX	Absolute index to start of file entry in CASBUF.
LINBUF	Pointer to A$ string in memory.
RELFI	Relative file number of an entry in the directory.
SCAND	Pointer to address of the start of the screen.
SECHI	Hi byte of SECNUM.
SECLOW	Low byte of SECNUM.
SECNUM	Current sector number being read or written.
STSEC	Directory start sector number of file being requested.
TEMP	Value of ASCII character on the screen.
TSECH	Temporary value of SECHI for DEC-HEX conversion.
TSECL	Temporary value of SECLOW for DEC-HEX conversion.
X	MISC variable.
Y	MISC variable.

STRING CONSTANT LIST

NAME	DESCRIPTION
A$	Input for COMMANDS, NUMBERS (HEX or DECIMAL) and CHANGE BYTE line.
AN$	Input string for various answers to prompts.
HEXREP$	String which holds the hex value of a converted decimal value.
HEXTAB$	Table of hex string values used by the DEC-HEX routine to convert a decimal number string to it's hex equivalent.

HOME UTILITIES AND EDUCATION

HOME ENERGY CONSUMPTION ANALYSIS

16K Cassette 32K Disk

by Joe E. Harb, Jr.

"Thermowatts" and "Kilowatts" are ATARI BASIC programs which require 16K RAM with cassette and 32K RAM with disk. "Thermowatts" analyzes yearly, monthly, and daily natural gas and electricity consumption and cost for homes which use both utilities. "Kilowatts" provides similar analysis for all-electric homes.

When we moved into our present house several years ago, I planned to make a number of energy conservation modifications to the house. I decided that I would like to use my ATARI 800 to determine what impact those modifications had on our energy consumption and costs. That led to the writing of "Kilowatts" which I subsequently rewrote as "Thermowatts," using natural gas data which I still had on hand from my previous house. Both programs make provisions for yearly and monthly temperature fluctuations. Statistics generated by both programs can be displayed on the screen or printed to a line printer.

Monthly and yearly temperature variations are taken into consideration by analyzing kilowatt/therm consumption per cooling/heating degree day, as appropriate. A heating degree day is each degree that the average temperature drops below 65 degrees F on a given day. A cooling degree day is each degree above 65 degrees F. The total number of cooling and heating degree days in each month can be obtained from your local weather bureau (National Oceanic and Atmospheric Administration — NOAA). Our local NOAA office at Baltimore Washington International Airport kindly provided me with several years of monthly degree day information over the telephone.

In a given month, a minimum of 100 cooling degree days is required before the programs will calculate cooling degree day consumption for that month. A minimum of 200 heating degree days is required for heating degree day analysis. This was done because in months when the number of heating or cooling days is below the threshold, energy use for heating or cooling is so low that the data becomes heavily biased by other energy use. This bias makes it seem that consumption per degree day is abnormally high. To change the threshold for cooling degree days, change the value of MINCD in line 100 of Kilowatts and 110 of Thermowatts. To change the threshold of heating degree days, change the value of MINHD on the same line.

In order to further minimize distortion by electricity consumption for uses other than heating and cooling, both programs subtract 400 kilowatts from each month's total electricity use before computing consumption per degree day. (This subtraction is not performed in computing any other statistics.) The variable used in the subtraction is FCTR, also in Line 100/110. It can be changed if you feel your non-heating/cooling electricity use is higher or lower.

All REM statements can be eliminated from both programs without requiring any line number changes. Additionally, if you feel the explanation of DATA statements given in the following paragraph is adequate, you can eliminate the instruction subroutine (Lines 6999-7190 in both programs, 2050 in "Kilowatts," and 2090 in "Thermowatts"). If you do not have a printer, you can eliminate the printer subroutines (Lines 2040 and 5999-6880 in "Kilowatts" and 2040, 2080 and 5999-6860 in "Thermowatts").

One DATA line is required for each month of data. DATA lines must be numbered in increments of 1, beginning with Line 1000, i.e.,

```
1000 DATA JAN,79,1329,29,56.10,30,29.88,984,0
1001 DATA FEB,79,1426,28,60.44,32,31.44,1100,0
1002 DATA MAR,79,520,31,50.98,11,20.33,520,15
```

DATA statements must contain: month (first three letters); year (last two digits); number of kilowatts used; number of days in billing period; cost of electricity (paid on time and including fuel surcharge); number of therms; cost of natural gas; heating degree days; and cooling degree days. The number of

therms and cost of natural gas are not used in "Kilowatts." All of the required information except heating and cooling degree days can be obtained from utility bills. As explained above, the information on heating and cooling degree days can be obtained from your local NOAA office.

If you have been looking for a relatively quick and easy way of neatly aligning columns of figures, particularly those with decimal fractions, you might want to consider using the technique I employed in several subroutines of both programs, for example in Lines 3170-3190. It can be done in four easy steps:

1. Decide the rightmost column for displaying a particular set of figures. Then add 1 to that value. In subroutine 3000, I wanted the last digit of the variable X to be printed in column 11. I then added 1 to that number, for a total of 12. If you are aligning figures with decimal fractions, use the column where the decimal point is to be printed, and do not add 1.

2. Measure the length of the variable by converting it to a string and using the LEN function. In Line 3170, LEN(STR$(INT(X))) means calculate the length (LEN) of the variable X after converting it to an integer (INT) and then to a string (STR$). The variable must be converted to a string because the LEN function can only measure the length of string variables. For this measurement, it is important to convert a numeric variable to an integer when the variable includes a decimal fraction. This is necessary because the ATARI eliminates final zeros after the decimal point. Thus, 3.50 is displayed 3.5. Consequently, if you wished to align the numbers 3.5 and 4.27 and if you measured the whole length of the variable, the columnar alignment of the numbers would be:

```
  3.5
 4.27
```

3. Pick a variable name for the column where printing of the display variable is to begin. (I used CL1 in the example.) Then, use the algorithm in this paragraph to calculate the column where printing is to begin. The algorithm subtracts the length of the integer portion of the string from the value calculated in step 1. In other words, the column where printing is to begin equals the length of the integer portion of the variable subtracted from the column where printing is to end. That is expressed in BASIC as CL1=12-LEN(STR$(INT(X))). This means that the first digit of the variable X will be displayed at screen column 12 minus the length of the integer X.

4. Position the cursor at the column and row where printing is to begin. This is done with the POSITION statement. In Line 3180, the cursor is positioned at column CL1,row PEEK(84). PEEK(84) is the memory location of the current cursor row. Finally, use the PRINT statement to display the variable on the screen. Once you get used to this process, it can be done fairly fast. Of course, it can be further simplified by performing the whole operation at one time:

```
POSITION 12-LEN(STR$(INT(X))),PEEK(84):?X
```

In "Thermowatts," each of the subroutines for the menu options does double duty. Each subroutine computes either gas or electricity statistics, depending on what is requested. The software accomplishes this by setting the variable T to a "0" or a "1" during menu selection. A "0" indicates that electricity data is to be processed, and a "1" indicates natural gas data. Each subroutine has statements which check the value of T and then select the appropriate data or print the proper column headings. For instance, in Lines 3120-3130, if T=0, the variable DD (degree days) = CD (cooling degree days) because electricity powers air conditioning equipment. If T=1, DD=HD (heating degree days) because natural gas provides heat.

During operation of these programs, do not depress the return key at any time when responding to a screen prompt. Simply type the letter(s) or numbers desired for input. The GET statement will determine which key(s) you depressed. In order to access the keyboard, a channel to the keyboard was opened in Line 70.

Variables used in Kilowatts and Thermowatts.

A: Used with GET to determine last key depressed on keyboard.

A1$: Used only in gas and electricity program. Represents variations of the words "therm" or "kilowatts" in column headings on screen or printer. Allows one subroutine to print headings for gas or electricity.

ANET: Used to represent electricty cost (NET) or gas cost (GNET) whenever single subroutine must calculate either gas or electricity statistics.

AVG: Per kilowatt or per therm cost.

B: Use with A when more than one key input from keyboard is required.

C: Used with A & B when three-key input required from keyboard.

CAVG: Average monthly consumption of kilowatts per degree day. Used only in subroutine 6000 of Kilowatts. See explanation under CDAVG.

CD: Cooling degree days in a given month.

CDAVG: Average annual consumption of kilowatts per cooling degree day. Used only in subroutine 6000 of Kilowatts because both cooling and heating degree day information are analyzed and printed at the same time. In Thermowatts, this is not necessary because there is so much data that separate printouts are required for cooling and heating degree day consumption. Consequently, a single variable DDAVG can perform double duty.

CDDIV: Total number of Kilowatts used when computing annual average consumption of Kilowatts

per cooling degree day. Used only in subroutine 6000 of Kilowatts. See explanation under CDAVG.

CDTOT: Total number of cooling days per annum. Used only in subroutine 6000 of Kilowatts. See explanation under CDAVG.

CL1: (Column 1); Column where printing of specified data begins. Used to right-justify screen display.

CL2: (Column 2); Used with CL1 when more than 1 column cannot be right-justified in some other way.

CL3: (Column 3); Used with CL1 and CL2 when more than two columns cannot be right-justified in some other way.

CL4: (Column 4); Used with CL1, CL2 and CL3 when more than three columns cannot be right-justified in some other way.

COST: Total annual cost of gas or electricity.

DAYS: Number of days during billing period.

DD: Used to represent either cooling or heating degree days in subroutines where either can be used.

DDAVG: Average annual use of Kilowatts or therms per cooling or heating degree day.

DDN$: Used in subroutines 3000, 5000, and 6000 to represent words "HEAT" or "COOL" in column headings, depending on whether user has requested cooling or heating degree day information.

DDT: Total number of heating/cooling degree days in a given year.

DIV: Total number of energy units used when computing annual average consumption per degree day. Used in subroutine 5000 of Thermowatts and subroutines 5000 and 6000 of Kilowatts.

FCTR: Estimated minimum amount of electricity used monthly for uses other than heating or cooling. Subtracted from UNITS before computing consumption per degree day. Can be raised or lowered if estimated minimum is different.

GNET: Cost of gas without late charge.

GUNITS: Therms of gas used during billing period.

HAVG: Average monthly consumption of kilowatts per heating degree day. Used only in subroutine 600 of Kilowatts. See explanation under CDAVG.

HDDIV: Total number of kilowatts used when computing average annual consumption of kilowatts per heating degree day. Used only in subroutine 600 of Kilowatts. See explanation under CDAVG.

HDTOT: Total number of heating degree days per annum. Used only in subroutine 6000 of Kilowatts. See explanation of under CDAVG.

HIYR: High year in data base.

HL: No. of lines to be printed on each page.

K$: Month for which data requested in menu options A, B, E, and F.

KPD: Average number of kilowatts or therms per degree day.

KPD$: Used to represent either variable KPD or letters "N/A" when printing out results of kilowatts/therms per degree day computation.

LINE: Last line of DATA.

LOYR: Lowest year of data in data base

M$: Month of data contained in DATA line.

MINCD: Minimum number of cooling degree days necessary for computing electricity consumption per cooling degree day.

MINHD: Minimum number of heating degree days necessary for computing gas/electricty consumption per degree day.

NET: Cost of electricity without late charge.

NR: Used to calculate number of months in data base.

PRNT$: One PRNT$ string is created for each line of data to be printed with the line printer in subroutine 6000. Allows data to be aligned easily in columns without using TAB functions which vary from printer to printer.

R$: Represents month in subroutine 6460/6570 to compare same month of different years.

SET: Sets flag when high line of page print reached during loop.

T: A flag. In Thermowatts, it is set during menu selection. It is used later in subroutines to identify whether gas or electricity data is to be processed. In Kilowatts, it is set at beginning of subroutines 3000 and 5000 to identify whether user has requested information on consumption per cooling or heating degree day. This is unnecessary in Thermowatts because the choice of desired information is implied by menu selection of electricty or natural gas data.

TIME: Last line printed on printer.

UNITS: Kilowatts used during billing period.

UP: Average daily kilowatt or therm consumption.

USE: Total annual consumption of gas or electricity.

Y: Year of data on DATA line.

YR: Year of data being processed.

Z: Index variable for loops, i.e., keeps track of no. of times loop has occurred. □

Thermowatts

```
10 ? "K":POKE 82,0
20 ? "              THERMOWATTS"
30 ? "          GAS & ELECTRICITY"
40 ? "          ANALYSIS PROGRAM":?
50 ? "             BY JOE HARB":?
60 ? :? "DURING OPERATION OF THIS PROG
RAM, DO NOTDEPRESS RETURN KEY AFTER TY
PING ANSWERS TO PROMPTS"
70 OPEN #1,4,0,"K:":REM OPEN KEYBOARD
TO GET INPUTS WHEN GET STATEMENT IS US
ED THROUGHOUT PROGRAM
80 ? :? "DEPRESS ANY KEY TO CONTINUE."
:GET #1,A:? "K"
100 DIM A1$(9),A2$(6),DDN$(4),M$(3),K$
(3),KPD$(6),PRNT$(65),R$(3)
110 MINCD=100:MINHD=200:FCTR=400:REM M
INCD=MINIMUM COOLING DAYS NECESSARY FO
RCOMPUTATION
```

```
120 REM A1$ & A2$ ARE USED TO PRINT VA
RIATIONS OF THE WORDS KWATTS OR THERMS
 SO ONE SUBROUTINE CAN BE USED FOR
130 REM ELECTRICITY OR GAS COMPUTATION
200 REM CALCULATE: TOTAL MONTHS OF DAT
A (NR); LOW YEAR OF DATA (LOYR); AND H
IGH YEAR OF DATA (HIYR)
210 NR=0
220 READ M$,Y,UNITS,DAYS,NET,GUNITS,GN
ET,HD,CD
230 REM M$=MONTH, Y=YEAR, UNITS=KILOWA
TTS USED & GUNITS=GAS THERM USED IN BI
LLING PERIOD
240 REM DAYS=NR. OF DAYS IN BILLING PE
RIOD
250 REM NET=COST OF ELECTRICITY WHEN B
ILL PAID ON TIME, GNET=COST OF GAS PAI
D ON TIME
260 REM CALCULATE NR. OF MONTHS OF INF
O IN DATA BASE(NR), HIGH YEAR OF DATA(
HIYR), & LOW YEAR OF DATA(LOYR)
270 LOYR=Y:RESTORE
280 READ M$,Y,UNITS,DAYS,NET,GUNITS,GN
ET,HD,CD
290 IF M$="END" THEN RESTORE :GOTO 200
0
300 NR=NR+1:HIYR=Y
310 GOTO 280
498 REM SUBROUTINE TO GET INPUT FOR ME
NU OPTIONS A, B, E, F; THEN CLEAR INPU
T QUESTIONS FROM SCREEN TO ALLOW
499 REM DISPLAY OF ADDITIONAL DATA
500 ? "TYPE FIRST THREE LETTERS OF MON
TH YOU   WANT.":GET #1,A:GET #1,B:GET
#1,C
510 REM NEXT LINE CONVERTS ATASCI VALU
ES TYPED ON KEYBOARD TO A STRING
520 K$=CHR$(A):K$(LEN(K$)+1)=CHR$(B):K
$(LEN(K$)+1)=CHR$(C):GOSUB 530:RETURN
530 POKE 84,PEEK(84)-2:FOR Z=0 TO 1:?
"
  ":NEXT Z:REM 39 SPACES
540 POKE 84,PEEK(84)-2:RETURN
1000 DATA JAN,79,624,16,26.20,51,36.18
,984,0
1001 DATA FEB,79,602,31,25.98,60,42.40
,1100,0
1002 DATA MAR,79,536,29,21.65,55,40.61
,520,15
1003 DATA APR,79,454,30,19.80,49,35.20
,354,4
1004 DATA MAY,79,527,32,27.91,40,31.55
,75,72
1005 DATA JUN,79,768,29,38.46,33,22.75
,6,183
1006 DATA JUL,79,1281,30,55.65,10,9.04
,2,348
1007 DATA AUG,79,691,29,36.45,8,8.50,3
,341
1008 DATA SEP,79,1242,32,52.16,12,14.2
6,22,145
1009 DATA OCT,79,571,30,24.43,20,20.11
,311,28
1010 DATA NOV,79,686,32,27.92,32,25.67
,425,1
1011 DATA DEC,79,688,31,26.75,49,40.03
,757,0
1012 DATA JAN,80,619,28,24.45,53,38.88
,962,0
1013 DATA FEB,80,527,32,20.97,57,45.98
,967,0
1014 DATA MAR,80,520,29,21.41,50,41.09
,723,0
1015 DATA APR,80,521,30,24.64,39,23.05
,273,0
1016 DATA MAY,80,591,32,34.82,22,18.97
,74,97
1017 DATA JUN,80,739,29,37.98,16,10.49
,6,203
1018 DATA JUL,80,1603,30,98.70,11,7.43
,0,415
1019 DATA AUG,80,838,29,53.52,8,6.66,0
,431
1020 DATA SEP,80,1530,32,74.26,15,10.7
7,20,245
1021 DATA OCT,80,589,30,30.45,28,22.84
,311,17
1022 DATA NOV,80,690,33,30.21,33,27.64
,620,0
1023 DATA DEC,80,770,31,33.04,41,34.49
,908,0
1024 DATA JAN,81,642,28,28.85,58,53.37
,1145,0
1899 DATA END,999,0,0,0,0,0,0,0
1999 REM MENU SUBROUTINE
2000 ? "THIS PROGRAM ALLOWS THE FOLLOW
ING      SELECTIONS:";?
2010 ? "      A. TOTAL MONTHLY AND AVER
AGE DAILY KILOWATT USE"
2020 ? "      B. TOTAL MONTHLY AND AVER
AGE DAILY KILOWATT COST"
2030 ? "      C. TOTAL ANNUAL KILOWATT
USE AND    COST"
2040 ? "      D. PRINTOUT OF ALL ELECTR
ICAL USE   DATA"
2050 ? "      E. TOTAL MONTHLY AND AVER
AGE DAILY GAS THERM USE"
2060 ? "      F. TOTAL MONTHLY AND AVER
AGE DAILY THERM COST"
2070 ? "      G. TOTAL ANNUAL THERM USE
 AND COST"
2080 ? "      H. PRINTOUT OF ALL GAS US
E DATA"
2090 ? "      I. DATA INPUT INSTRUCTION
S"
2100 ? "      J. EXIT PROGRAM"
2110 REM GET IS USED TO DETERMINE LETT
ER TYPED ON KEY BOARD; A=ATASCI VALUE
OF LETTER TYPED
2120 ? "TYPE LETTER OF OPTION YOU WANT
":GET #1,A:? "K"
2130 T=0:IF A=65 THEN 3000:REM T IS FL
AG TO TELL LATER SUBROUTINES WHETHER G
AS OR ELECTRICITY ANALYSIS REQUESTED
2140 IF A=66 THEN 4000
2150 IF A=67 THEN 5000
2160 IF A=68 THEN OPEN #4,8,0,"P:":GOT
O 6000:REM OPEN CHANNEL TO PRINTER
2170 T=1:IF A=69 THEN 3000
2180 IF A=70 THEN 4000
2190 IF A=71 THEN 5000
2200 IF A=72 THEN OPEN #4,8,0,"P:":GOT
O 6000
2210 IF A=73 THEN 7000
2220 IF A=74 THEN POKE 82,2:END
2999 REM SUBROUTINE FOR MENU OPTIONS A
 & E
3000 GOSUB 500
3010 IF T=0 THEN DDN$="COOL":A1$="KWU
 "
3020 IF T=1 THEN DDN$="HEAT":A1$="THER
M"
3030 ? "                AVG      TOTAL
AVG ";A1$:REM 15 SPACES BEFORE AVG
3040 ? "         TOTAL   DAILY    ";DDN$;
"   USE"
3050 ? "         ";A1$;"  ";A1$;"   DGR
EE  PER ";DDN$
3060 ? "MONTH    USE     USE      DAYS
DGREE DAY"
3070 FOR Z=1 TO NR
3080 READ M$,Y,UNITS,DAYS,NET,GUNITS,G
NET,HD,CD
3090 IF M$<>K$ THEN 3200
3100 X=UNITS:IF T=1 THEN X=GUNITS
3110 UP=INT(1000*X/DAYS)/1000:REM COMP
UTE UNITS PER DAY AND LIMIT DECIMAL PL
ACES DISPLAYED
3120 IF T=0 THEN DD=CD:IF CD>MINCD THE
N 3150
3130 IF T=1 THEN DD=HD:IF HD>MINHD THE
N FCTR=0:GOTO 3150
3140 IF CD<=MINCD OR HD<=MINHD THEN KP
D$="N/A":GOTO 3170
3150 KPD=INT(1000*(X-FCTR)/((DD/30)*DA
YS))/1000:KPD$=STR$(KPD):REM COMPUTE U
NITS PER DEGREE DAY
3160 REM LINES 3170-3190 ALIGN AND PRI
NT   SCREEN DISPLAY
3170 CL1=12-LEN(STR$(INT(X))):CL2=17-L
EN(STR$(INT(UP))):CL3=27-LEN(STR$(DD))
:CL4=32-LEN(STR$(INT(KPD)))
3180 ? M$;" ";Y;"  ";:POSITION CL1,PEE
K(84):? X;:POSITION CL2,PEEK(84):? UP;
:POSITION CL3,PEEK(84):? DD;
```

```
3190 POSITION CL4,PEEK(84):? KPD$
3200 NEXT Z:RESTORE
3210 ? :? "DO YOU WANT TO LOOK AT ANOT
HER MONTH?   TYPE Y OR N.":GET #1,A
3220 IF A=89 THEN GOSUB 530:GOSUB 500:
GOTO 3070
3230 ? "R":GOTO 2000
3999 REM SUBROUTINE FOR MENU OPTIONS B
 & F
4000 GOSUB 500
4010 ? "MONTH    TOTAL   TOTAL    UNIT"
4020 A1$="KWU  ":IF T=1 THEN A1$="THER
M"
4030 ? "         ";A1$;"   COST    COS
T":REM 9 SPACES BEFORE A1$
4040 FOR Z=1 TO NR
4050 READ M$,Y,UNITS,DAYS,NET,GUNITS,G
NET,HD,CD
4060 IF M$<>K$ THEN 4110
4070 X=UNITS:ANET=NET:IF T=1 THEN X=GU
NITS:ANET=GNET
4080 AVG=INT(10000*(ANET/X))/10000:REM
 CALCULATE COST PER UNIT
4090 CL1=13-LEN(STR$(X)):CL2=19-LEN(ST
R$(INT(ANET)))
4100 ? M$;" ";Y;:POSITION CL1,PEEK(84)
:? X;"   ";:POSITION CL2,PEEK(84):? AN
ET;:POSITION 25,PEEK(84):? AVG
4110 NEXT Z:RESTORE :?
4120 ? :? "DO YOU WANT TO LOOK AT ANOT
HER MONTH?   TYPE Y OR N.":GET #1,A
4130 IF A=89 THEN GOSUB 530:GOSUB 500:
GOTO 4040
4140 ? "R":GOTO 2000
4999 REM SUBROUTINE FOR MENU OPTIONS C
 & G
5000 IF T=0 THEN DDN$="COOL":A1$="KWAT
TS"
5010 IF T=1 THEN DDN$="HEAT":A1$="THER
MS":FCTR=0
5020 YR=LOYR
5030 ? "                          ";DDN$;"
   AVG ";A1$:REM 22 SPACES BEFORE DDN$
5040 ? "      ";A1$;"          DGREE
PER DGREE":REM 9 SPACES BEFORE DGREE
5050 ? "YEAR  USED    COST    DAYS   D
AY"
5060 USE=0:COST=0:DDT=0:DIV=0
5070 FOR Z=1 TO NR
5080 READ M$,Y,UNITS,DAYS,NET,GUNITS,G
NET,HD,CD
5090 X=UNITS:ANET=NET:IF T=1 THEN X=GU
NITS:ANET=GNET
5100 IF Y<>YR THEN 5150
5110 IF T=0 THEN DD=CD:IF CD<=MINCD TH
EN DD=0:GOTO 5140
5120 IF T=1 THEN DD=HD:IF HD<=MINHD TH
EN DD=0:GOTO 5140
5130 DDT=DDT+DD:DIV=DIV+X-FCTR
5140 USE=USE+X:COST=COST+ANET
5150 NEXT Z:RESTORE
5160 DDAVG=0:IF DDT>0 THEN DDAVG=INT(1
000*DIV/DDT)/1000
5170 CL1=11-LEN(STR$(USE)):CL2=17-LEN(
STR$(INT(COST))):CL3=26-LEN(STR$(DDT))
:CL4=30-LEN(STR$(INT(DDAVG)))
5180 ? YR+1900;:POSITION CL1,PEEK(84):
? USE;:POSITION CL2,PEEK(84):? COST;
5190 POSITION CL3,PEEK(84):? DDT;:POSI
TION CL4,PEEK(84):? DDAVG
5200 YR=YR+1:IF YR<HIYR+1 THEN 5060
5210 ? "DEPRESS ANY KEY TO RETURN TO M
ENU.":GET #1,A
5220 ? "R":GOTO 2000
5999 REM SUBROUTINE FOR MENU OPTIONS D
 & H FOR (LINE PRINTER)
6000 TIME=0:SET=0:? "TYPE NUMBER OF LI
NES PER PAGE TO BE     PRINTED":GET #1
,A:GET #1,B:HL=((A-48)*10)+(B-48)
6010 LPRINT CHR$(27);CHR$(56):REM DISA
BLE EPSON PRINTER "END OF PAPER" FUNCT
ION
6020 A1$="KWATTS":DDN$="COOL":IF T=1 T
HEN A1$="THERMS":DDN$="HEAT":FCTR=0
6030 ? #4;"                          T
OTAL    AVG":REM 26 SPACES BEFORE TOTA
L
6040 ? #4;"                          "
;DDN$;"     ";A1$:REM 26 SPACES BEFORE
 DDN$
6050 ? #4;"          ";A1$;"  TOTAL
DEGREE   PER":REM 9 SPACES BEFORE A1$
6060 ? #4;"YEAR     USED    COST     D
AYS     DGR DAY":LPRINT
6070 TIME=TIME+5:YR=LOYR
6080 USE=0:COST=0:DDT=0:DIV=0
6090 PRNT$="

 ":REM 65 SPACES
6100 FOR Z=1 TO NR:REM CALCULATE ANNUA
L CONSUMPTION AND COST
6110 READ M$,Y,UNITS,DAYS,NET,GUNITS,G
NET,HD,CD
6120 X=UNITS:ANET=NET:IF T=1 THEN X=GU
NITS:ANET=GNET
6130 IF Y<>YR THEN 6180
6140 IF T=0 THEN DD=CD:IF CD<=MINCD TH
EN DD=0:GOTO 6170
6150 IF T=1 THEN DD=HD:IF HD<=MINHD TH
EN DD=0:GOTO 6170
6160 DDT=DDT+DD:DIV=DIV+X-FCTR
6170 USE=USE+X:COST=COST+ANET
6180 NEXT Z:RESTORE
6190 DDAVG=0:IF DDT>0 THEN DDAVG=INT(1
000*DIV/DDT)/100
6200 PRNT$(11-LEN(STR$(USE)),12)=STR$(
USE)
6210 PRNT$(18-LEN(STR$(INT(COST))),21)
=STR$(COST)
6220 PRNT$(28-LEN(STR$(DDT)),30)=STR$(
DDT)
6230 PRNT$(33-LEN(STR$(INT(DDAVG))),36
)=STR$(DDAVG)
6240 ? #4;YR+1900;PRNT$:TIME=TIME+1
6250 YR=YR+1:IF YR<HIYR+1 THEN 6080
6260 LPRINT :TIME=TIME+1
6399 REM CALCULATE AND PRINT MONTHLY
DATA. SUBROUTINE 6410 PRINTS COLUMN
HEADINS ON EACH SHEET OF PAPER
6400 GOSUB 6410:GOTO 6470
6410 A1$="KWATT":DDN$="COOL":IF T=1 TH
EN A1$="THERM":DDN$="HEAT"
6420 PRINT #4;"
                          ";A1$:REM
51 SPACES BEFORE A1$
6430 PRINT #4;"MONTH     DAILY    MONTHL
Y  TOTAL     COST     ";DDN$;"    PER"
6440 PRINT #4;"          ";A1$;"   ";A1
$;"    MONTHLY  PER      DEGREE  DEGREE
":REM 9 SPACES BEFORE A1$
6450 PRINT #4;"          USE      USE
   COST     ";A1$;"   DAYS    DAY":TIM
E=TIME+4:REM 9 SPACES BEFORE USE
6460 RETURN
6470 R$="JAN":GOSUB 6600
6480 R$="FEB":GOSUB 6600
6490 R$="MAR":GOSUB 6600
6500 R$="APR":GOSUB 6600
6510 R$="MAY":GOSUB 6600
6520 R$="JUN":GOSUB 6600
6530 R$="JUL":GOSUB 6600
6540 R$="AUG":GOSUB 6600
6550 R$="SEP":GOSUB 6600
6560 R$="OCT":GOSUB 6600
6570 R$="NOV":GOSUB 6600
6580 R$="DEC":GOSUB 6600
6590 CLOSE #4:? "R":GOTO 2000
6600 FOR Z=1 TO NR
6610 PRNT$="

"
6620 READ M$,Y,UNITS,DAYS,NET,GUNITS,G
NET,HD,CD
6630 IF M$<>R$ THEN 6800
6640 X=UNITS:ANET=NET:IF T=1 THEN X=GU
NITS:ANET=NET
6650 UP=INT(100*(X/DAYS))/100
6660 AVG=INT(1000*(ANET/X))/1000
6670 IF T=0 THEN DD=CD:IF CD>=MINCD TH
EN 6700
6680 IF T=1 THEN DD=HD:IF HD>=MINHD TH
EN 6700
6690 IF CD<MINCD OR HD<MINHD THEN KPD=
0:GOTO 6710
```

```
6700 KPD=INT(1000*(X-FCTR)/((DD/30)*DA
YS))/1000:KPD$=STR$(KPD)
6710 PRNT$(6-LEN(STR$(INT(UP))),8)=STR
$(UP)
6720 PRNT$(17-LEN(STR$(X)),16)=STR$(X)
6730 PRNT$(25-LEN(STR$(INT(ANET))),27)
=STR$(ANET)
6740 PRNT$(31-LEN(STR$(INT(AVG))),34)=
STR$(AVG)
6750 PRNT$(41-LEN(STR$(DD)),40)=STR$(D
D)
6760 IF KPD=0 THEN PRNT$(49,51)="N/A":
GOTO 6780
6770 PRNT$(48-LEN(STR$(INT(KPD))),51)=
STR$(KPD)
6780 PRINT #4;M$;" ";Y;PRNT$
6790 TIME=TIME+1:IF TIME=HL THEN SET=1
6800 NEXT Z:RESTORE
6810 IF SET=0 THEN 6850
6820 IF R$="DEC" THEN 6860
6830 ? "INSERT ANOTHER SHEET OF PAPER;
 THEN     DEPRESS ANY KEY.":GET #1,A
6840 TIME=0:SET=0:GOSUB 6410
6850 LPRINT :TIME=TIME+1:IF TIME=HL TH
EN 6820
6860 RETURN
6999 REM INSTRUCTIONS FOR PREPARING DA
TA LINES
7000 ? "K":LINE=NR+999
7010 ? "FOR EACH MONTH OF DATA YOU HAV
E, YOU     MUST TYPE ONE DATA LINE.":?
7020 ? "THE FIRST DATA LINE MUST BE NU
MBERED 1000"
7030 ? "AFTER THAT, EACH DATA LINE MUS
T BE      NUMBERED ONE HIGHER THAN THE
 LAST, FOR"
7040 ? "EXAMPLE 1000 MUST BE FOLLOWED
BY 1001,  1002, 1003, 1004, ETC.":?
7050 ? "DEPRESS ANY KEY WHEN READY FOR
 NEXT     INSTRUCTIONS.":GET #1,A
7060 ? "REQUIRED FORMAT FOR DATA LINE:
"
7070 ? "1000 DATA OCT,82,1350,30,79.25
,40,35.20,675,0"
7080 ? "DATA ITEMS ARE:"
7090 ? "1. MONTH; MUST BE 3 LETTERS LO
NG."
7100 ? "2. YEAR; MUST BE 2 NUMBERS LON
G."
7110 ? "3. NUMBER OF KILOWATTS USED DU
RING MONTH"
7120 ? "4. NUMBER OF DAYS IN BILLING P
ERIOD"
7130 ? "5. NET COST OF ELECTRICITY IN
BILLING   PERIOD. DO NOT USE '$' BEFOR
E COST."
7140 ? "6. NUMBER OF THERMS USED DURIN
G BILLING PERIOD."
7150 ? "7. NET GAS COST DURING BILLING
 PERIOD."
7160 ? "8. HEATING DEGREE DAYS IN BILL
ING PERIOD."
7170 ? "9. COOLING DEGREE DAYS IN BILL
ING PERIOD."
7180 ? "THE LAST LINE OF DATA YOU ENTE
RED WAS:  ";LINE
7190 ? "NOW BEGIN TYPING NEW DATA LINE
S."
```

•

CHECKSUM DATA

(See pgs. 7-10)

```
10 DATA 292,441,842,99,579,907,370,636
,836,320,710,391,885,96,502,7906
230 DATA 233,730,591,746,76,520,80,601
,716,931,938,958,840,258,376,8594
540 DATA 999,829,942,42,828,874,900,82
1,722,238,0,882,813,869,834,10593
1014 DATA 790,789,901,886,790,706,235,
40,817,754,52,716,513,196,246,8431
2020 DATA 108,384,413,926,623,957,582,
346,410,660,150,289,852,856,62,7618
2170 DATA 43,853,857,152,858,829,785,7
25,904,34,834,67,296,576,496,8309
3080 DATA 529,843,657,119,289,954,155,
6,288,885,643,249,778,316,846,7557
3230 DATA 308,791,727,470,859,221,495,
528,844,558,976,709,947,58,316,8807
4130 DATA 843,308,797,965,960,821,903,
227,574,777,500,533,562,736,79,9585
5120 DATA 103,217,179,784,160,155,583,
146,442,40,311,374,459,583,449,4985
6030 DATA 909,968,263,948,489,781,562,
324,531,560,750,89,113,222,184,7693
6180 DATA 789,719,261,379,272,537,363,
456,452,183,373,949,793,589,58,7173
6450 DATA 209,807,0,993,12,27,14,58,57
,35,33,24,61,993,415,3738
6600 DATA 513,715,547,890,256,544,862,
628,652,432,421,622,602,336,833,8853
6750 DATA 94,607,871,681,101,802,752,9
74,248,891,158,819,716,946,671,9331
7020 DATA 376,829,188,999,227,833,62,3
68,898,846,531,489,402,50,291,7389
7170 DATA 342,975,364,1681
```

•

Kilowatts

```
10 POKE 82,0
20 ? "K↓                KILOWATTS"
30 ? "                ELECTRICITY"
40 ? "              ANALYSIS PROGRAM"
50 ? "                BY JOE HARB"
60 ? "↓↓DURING OPERATION OF THIS PROGR
AM,     DO NOT DEPRESS RETURN KEY AF
TER TYPING  ANSWERS TO PROMPTS."
70 OPEN #1,4,0,"K:":REM OPEN KEYBOARD
TO GET INPUTS LATER IN PROGRAM  WHEN G
ET STATEMENT IS USED
80 ? :? "DEPRESS ANY KEY TO CONTINUE."
:GET #1,A
90 DIM DDN$(4),M$(3),K$(3),KPD$(6),PRN
T$(65),R$(3)
100 MINCD=100:MINHD=200:FCTR=400:REM M
INCD=MINIMUM COOLING DAYS NECESSARY FO
RCOMPUTATION
110 REM MINHD=MINIMUM HEATING DEGREE D
AYS NECESSARY
120 REM FCTR=NR. OF KILOWATTS TO BE SU
BTRACTED FROM MONTHLY KILOWATT USE WHE
N COMPUTING DEGREE DAYS.
130 REM SUBTRACTING FCTR REDUCES EXTEN
T TO WHICH OTHER HOUSEHOLD ELECTRICITY
 USE BIASES HEATING AND COOLING STATS
200 REM CALCULATE: TOTAL MONTHS OF DAT
A (NR); LOW YEAR OF DATA (LOYR); AND H
IGH YEAR OF DATA (HIYR)
210 NR=0
220 READ M$,Y,UNITS,DAYS,NET,HD,CD
230 REM M$=MONTH, Y=YEAR, UNITS=KILOWA
TTS USED USED IN BILLING PERIOD
240 REM DAYS=NR. OF DAYS IN BILLING PE
RIOD
250 REM NET=COST OF ELECTRICITY WHEN B
ILL PAID ON TIME,DD=DEGREE DAYS DURING
 BILLING MONTH
260 REM HD=HEATING DEGREE DAYS
270 REM CD=COOLING DEGREE DAYS
280 LOYR=Y:RESTORE
290 READ M$,Y,UNITS,DAYS,NET,HD,CD
300 IF M$="END" THEN RESTORE :GOTO 200
0
310 NR=NR+1:HIYR=Y
320 GOTO 290
498 REM SUBROUTINE TO GET INPUT FOR ME
NU OPTIONS A & B; THEN CLEAR INPUT QUE
STIONS FROM SCREEN TO ALLOW DISPLAY
499 REM OF ADDITIONAL DATA
500 ? "KTYPE FIRST THREE LETTERS OF MO
NTH YOU    WANT.":GET #1,A:GET #1,B:GET
 #1,C
510 REM NEXT LINE CONVERTS ATASCI VALU
ES TYPED ON KEYBOARD TO A STRING
520 K$=CHR$(A):K$(LEN(K$)+1)=CHR$(B):K
$(LEN(K$)+1)=CHR$(C):GOSUB 530:RETURN
```

```
530 POKE 84,PEEK(84)-2:FOR Z=0 TO 1:?
"
  ":NEXT Z:REM 39 SPACES
540 POKE 84,PEEK(84)-2:RETURN
1000 REM YOUR DATA STATEMENTS GO HERE
1899 DATA END,999,0,0,0,0,0
1999 REM MENU OPTIONS
2000 ? "{CLR}THIS PROGRAM ALLOWS THE FOLLO
WING       SELECTIONS:":?
2010 ? "     A. TOTAL MONTHLY AND AVER
AGE DAILY          KILOWATT USE"
2020 ? "     B. TOTAL MONTHLY AND AVER
AGE DAILY          KILOWATT COST"
2030 ? "     C. TOTAL ANNUAL KILOWATT
USE AND            COST"
2040 ? "     D. PRINTOUT OF ALL ELECTR
ICAL USE           DATA"
2050 ? "     E. DATA INPUT INSTRUCTION
S"
2060 ? "     F. EXIT PROGRAM":?
2070 ? "TYPE LETTER OF OPTION YOU WANT
.":GET #1,A
2080 REM GET IS USED TO DETERMINE LETT
ER TYPED ON KEYBOARD; A=ATASCI VALUE O
F LETTER TYPED
2090 IF A=65 THEN 3000
2100 IF A=66 THEN 4000
2110 IF A=67 THEN 5000
2120 IF A=68 THEN TRAP 2160:OPEN #4,8,
0,"P:":TRAP 10000:GOTO 6000
2130 IF A=69 THEN 7000
2140 IF A=70 THEN POKE 82,2:END
2150 GOTO 2070
2160 REM PRINTER ERROR MESSAGE
2170 CLOSE #4:? "PRINTER IS NOT ON-LIN
E!{BELL}":TRAP 10000:GOTO 2070
2999 REM SUBROUTINE FOR MENU OPTION A
3000 GOSUB 500
3010 ? "DO YOU WANT TO INCLUDE INFORMA
TION ON   HEATING(H), COOLING(C) OR NE
ITHER(N)?":GET #1,A:GOSUB 530
3020 IF A=67 THEN DDN$="COOL":T=0
3030 IF A=72 THEN DDN$="HEAT":T=1
3040 IF A=78 THEN DDN$="    ":T=2:DD=0
3050 ? "MONTH   TOTAL AVG     TOTAL  A
VG KWU"
3060 ? "        KWU   DAILY   DGREE  P
ER ";DDN$:REM 8 SPACES BEFORE KWU
3070 ? "              KWU USE DAYS   D
GREE DAY";:REM 14 SPACES BEFORE KWU
3080 FOR Z=1 TO NR
3090 READ M$,Y,UNITS,DAYS,NET,HD,CD
3100 IF M$<>K$ THEN 3200
3110 UP=INT(100*UNITS/DAYS)/100:REM CO
MPUTE UNITS PER DAY AND LIMIT DECIMAL
PLACES DISPLAYED
3120 IF T=0 THEN DD=CD:IF CD>MINCD THE
N GOTO 3150
3130 IF T=1 THEN DD=HD:IF HD>MINHD THE
N GOTO 3150
3140 IF T=2 OR CD<=MINCD OR HD<=MINHD
THEN KPD$="N/A":GOTO 3170
3150 KPD=INT(100*(UNITS-FCTR)/((DD/30)
*DAYS))/100:KPD$=STR$(KPD):REM COMPUTE
 UNITS PER DEGREE DAY
3160 REM LINES 3170-3190 USED TO ALIGN
   AND PRINT SCREEN DISPLAY
3170 CL1=12-LEN(STR$(INT(UNITS))):CL2=
17-LEN(STR$(INT(UP))):CL3=26-LEN(STR$(
DD)):CL4=32-LEN(STR$(INT(KPD)))
3180 ? M$;" ";Y;:POSITION CL1,PEEK(84)
:? UNITS;:POSITION CL2,PEEK(84):? UP;:
POSITION CL3,PEEK(84):? DD;
3190 POSITION CL4,PEEK(84):? KPD$
3200 NEXT Z:RESTORE
3210 ? :? "DO YOU WANT TO LOOK AT ANOT
HER MONTH?   TYPE Y OR N.":GET #1,A
3220 IF A=89 THEN GOSUB 530:GOSUB 500:
GOTO 3080
3230 ? "{CLR}":GOTO 2000
3999 REM SUBROUTINE FOR MENU OPTION B
4000 GOSUB 500
4010 ? "MONTH     TOTAL    TOTAL    UNIT"
4020 ? "          KWU      COST     COST"
:REM 9 SPACES BEFORE KWU
4030 FOR Z=1 TO NR
4040 READ M$,Y,UNITS,DAYS,NET,HD,CD
4050 IF M$<>K$ THEN 4090
4060 AVG=INT(10000*(NET/UNITS))/10000:
REM CALCULATE AVERAGE DAILY USE
4070 CL1=13-LEN(STR$(INT(UNITS))):CL2=
19-LEN(STR$(INT(NET)))
4080 ? M$;" ";Y;:POSITION CL1,PEEK(84)
:? UNITS;"   ";:POSITION CL2,PEEK(84):
? NET;:POSITION 25,PEEK(84):? AVG
4090 NEXT Z:RESTORE
4100 ? :? "DO YOU WANT TO LOOK AT ANOT
HER MONTH?   TYPE Y OR N.":GET #1,A
4110 IF A=89 THEN GOSUB 530:GOSUB 500:
GOTO 4030
4120 ? "{CLR}":GOTO 2000
4999 REM SUBROUTINE FOR MENU OPTION C
5000 ? "{CLR}DO YOU WANT TO INCLUDE INFORM
ATION ON   HEATING (H) OR COOLING (C)?
":GET #1,A:YR=LOYR
5010 IF A=67 THEN DDN$="COOL":T=0
5020 IF A=72 THEN DDN$="HEAT":T=1
5030 ? "                    ";DDN$;"
   AVG KWATT":REM 22 SPACES BEFORE DDN
$
5040 ? "YEAR  KWATTS          DGREE  P
ER DGREE":REM 10 SPACES BEFORE DGREE
5050 ? "      USED    COST    DAYS   D
AY"
5060 USE=0:COST=0:DDT=0:DIV=0
5070 FOR Z=1 TO NR
5080 READ M$,Y,UNITS,DAYS,NET,HD,CD
5090 IF Y<>YR THEN 5140
5100 IF T=0 THEN DD=CD:IF CD<=MINCD TH
EN DD=0:GOTO 5130
5110 IF T=1 THEN DD=HD:IF HD<=MINHD TH
EN DD=0:GOTO 5130
5120 DDT=DDT+DD:DIV=DIV+UNITS-FCTR
5130 USE=USE+UNITS:COST=COST+NET
5140 NEXT Z:RESTORE
5150 DDAVG=0:IF DDT>0 THEN DDAVG=INT(1
00*DIV/DDT)/100
5160 CL1=17-LEN(STR$(INT(COST))):CL2=2
6-LEN(STR$(INT(DDT))):CL3=30-LEN(STR$(
INT(DDAVG)))
5170 ? YR+1900;"  ";USE;"   ";:POSITIO
N CL1,PEEK(84):? COST;:POSITION CL2,PE
EK(84):? DDT;:POSITION CL3,PEEK(84)
5180 ? DDAVG
5190 YR=YR+1:IF YR<HIYR+1 THEN 5060
5200 RESTORE
5210 ? :? "DEPRESS ANY KEY TO RETURN T
O MENU.":GET #1,A
5220 GOTO 2000
5999 REM SUBROUTINE FOR MENU OPTION D
6000 TIME=0:SET=0:? "{CLR}TYPE NUMBER OF L
INES PER PAGE TO BE     PRINTED.":GET
#1,A:GET #1,B:HL=((A-48)*10)+(B-48)
6010 LPRINT CHR$(27);CHR$(56):REM DISA
BLE EPSON "END OF PAPER" FUNCTION
6020 ? #4;"                              TOT
AL   AVG     TOTAL  AVG":REM 24 SPACES
 BEFORE TOTAL
6030 ? #4;"                             HEA
T    KWATT   COOL   KWATT":REM 24 SPAC
ES BEFORE HEAT
6040 ? #4;"         KWATTS  TOTAL   DGR
     PER     DGR    PER":REM 8 SPACES
BEFORE KWATT
6050 ? #4;"YEAR    USED    COST    DAY
S    DGR DAY DAYS   DGR DAY":LPRINT
6060 TIME=TIME+5:YR=LOYR
6070 USE=0:COST=0:CDDIV=0:CDTOT=0:HDDI
V=0:HDTOT=0:CDAVG=0:HDAVG=0
6080 REM CDDIV & HDDIV ARE NUMBER OF A
NNUAL KILOWATTS FOR HEATING & COOLING.
 ONLY MONTHS WITH MORE THAN 100
6090 REM COOLING OR 200 HEATING DEGREE
DAYS ARE INCLUDED. 500 KWATTS PER MONT
H SUBTRACTED BY FCTR FOR OTHER ELECT.
6100 REM CDTOT & HDTOT ARE TOTAL HEATI
NG/COOLING DEGREES PER ANNUM FROM MONT
HS WITH SUFFICIENT DEGREE DAYS
6110 PRNT$="

 ":REM 65 SPACES
6120 FOR Z=1 TO NR:REM CALCULATE ANNUA
L CONSUMPTION AND COST
6130 READ M$,Y,UNITS,DAYS,NET,HD,CD
```

```
6140 IF Y<>YR THEN 6180
6150 IF CD>MINCD THEN CDTOT=CDTOT+CD:C
DDIV=CDDIV+UNITS-FCTR
6160 IF HD>MINHD THEN HDTOT=HDTOT+HD:H
DDIV=HDDIV+UNITS-FCTR
6170 USE=USE+UNITS:COST=COST+NET
6180 NEXT Z:RESTORE
6190 IF CDTOT>0 THEN CDAVG=INT(100*CDD
IV/CDTOT)/100
6200 IF HDTOT>0 THEN HDAVG=INT(100*HDD
IV/HDTOT)/100
6210 ? #4;YR+1900;:PRNT$(11-LEN(STR$(U
SE)),10)=STR$(USE)
6220 PRNT$(16-LEN(STR$(INT(COST))),18)
=STR$(COST)
6230 PRNT$(25-LEN(STR$(HDTOT)),24)=STR
$(HDTOT)
6240 PRNT$(30-LEN(STR$(INT(HDAVG))),32
)=STR$(HDAVG)
6250 PRNT$(41-LEN(STR$(CDTOT)),40)=STR
$(CDTOT)
6260 PRNT$(45-LEN(STR$(INT(CDAVG))),47
)=STR$(CDAVG)
6270 ? #4;PRNT$:TIME=TIME+1
6280 YR=YR+1:IF YR<HIYR+1 THEN 6070
6290 RESTORE :LPRINT :TIME=TIME+1
6399 REM CALCULATE AND PRINT MONTHLY
DATA. SUBROUTINE 6410 PRINTS COLUMN
HEADINGS ON EACH SHEET OF PAPER
6400 GOSUB 6410:GOTO 6460
6410 ? #4;"
                    KWAT            KWAT
":TIME=TIME+1:REM 47 & 10 SPACES
6420 ? #4;"MONTH    DAILY    MNTHLY   MNT
HLY   COST     HEAT    PER     COOL    PER"
:TIME=TIME+1
6430 ? #4;"             KWATT    KWATT    COS
T     PER      DGRE    DGRE    DGRE
":TIME=TIME+1:REM 8 SPACES BE4 KWATT
6440 ? #4;"          USE      USE
     KWU      DAYS    DAY     DAYS    DAY"
:LPRINT :TIME=TIME+1:REM 8 & 13 SPCS
6450 RETURN
6460 R$="JAN":GOSUB 6600
6470 R$="FEB":GOSUB 6600
6480 R$="MAR":GOSUB 6600
6490 R$="APR":GOSUB 6600
6500 R$="MAY":GOSUB 6600
6510 R$="JUN":GOSUB 6600
6520 R$="JUL":GOSUB 6600
6530 R$="AUG":GOSUB 6600
6540 R$="SEP":GOSUB 6600
6550 R$="OCT":GOSUB 6600
6560 R$="NOV":GOSUB 6600
6570 R$="DEC":GOSUB 6600
6580 CLOSE #4:? "↖":GOTO 2000
6600 FOR Z=1 TO NR:REM CALCULATE MONTH
LY CONSUMPTION AND COST
6610 READ M$,Y,UNITS,DAYS,NET,HD,CD
6620 HAVG=0:CAVG=0
6630 IF M$<>R$ THEN 6820
6640 ? #4;M$;" ";Y;
6650 PRNT$="

 ":REM 65 SPACES
6660 UP=INT(100*(UNITS/DAYS))/100
6670 PRNT$(6-LEN(STR$(INT(UP))),8)=STR
$(UP)
6680 PRNT$(15-LEN(STR$(UNITS)),14)=STR
$(UNITS)
6690 PRNT$(22-LEN(STR$(INT(NET))),24)=
STR$(NET)
6700 AVG=INT(1000*(NET/UNITS))/1000
6710 PRNT$(28-LEN(STR$(INT(AVG))),31)=
STR$(AVG)
6720 PRNT$(39-LEN(STR$(HD)),38)=STR$(H
D)
6730 IF HD>MINHD THEN HAVG=INT(100*((U
NITS-FCTR)/HD))/1000
6740 IF HAVG=0 THEN PRNT$(42,44)="N/A"
:GOTO 6760
6750 PRNT$(43-LEN(STR$(INT(HAVG))),46)
=STR$(HAVG)
6760 PRNT$(53-LEN(STR$(CD)),52)=STR$(C
D)
6770 IF CD>MINCD THEN CAVG=INT(100*((U
NITS-FCTR)/CD))/1000
6780 IF CAVG=0 THEN PRNT$(56,58)="N/A"
:GOTO 6800
6790 PRNT$(57-LEN(STR$(INT(CAVG))),60)
=STR$(CAVG)
6800 TIME=TIME+1:IF TIME=HL THEN SET=1
6810 ? #4;PRNT$
6820 NEXT Z:RESTORE
6830 IF SET=0 THEN 6870
6840 IF R$="DEC" THEN 6880
6850 ? "INSERT ANOTHER SHEET OF PAPER;
 THEN   DEPRESS ANY KEY":GET #1,A
6860 TIME=0:SET=0:GOSUB 6410
6870 LPRINT :TIME=TIME+1:IF TIME=HL TH
EN 6840
6880 RETURN
6999 REM INSTRUCTIONS FOR PREPARING DA
TA LINES
7000 LINE=NR+999
7010 ? "↖FOR EACH MONTH OF DATA YOU HA
VE, YOU    MUST TYPE ONE DATA LINE."
7020 ? "↓THE FIRST DATA LINE MUST BE N
UMBERED    1000."
7030 ? "↓AFTER THAT, EACH DATA LINE MU
ST BE      NUMBERED ONE HIGHER THAN TH
E LAST."
7040 ? "FOR EXAMPLE, 1000 MUST BE FOLL
OWED BY   1001, 1002, 1003, 1004, ETC.
"
7050 ? "↓DEPRESS ANY KEY WHEN READY FO
R NEXT     INSTRUCTIONS.":GET #1,A
7060 ? "↖THE FOLLOWING IS THE FORMAT F
OR A DATA  LINE:"
7070 ? "↓1000 DATA OCT,82,1350,30,79.2
5,495,0"
7080 ? "↓REQUIRED DATA SEQUENCE AND FO
RMAT:"
7090 ? "1. MONTH; MUST BE 3 LETTERS LO
NG"
7100 ? "2. YEAR; MUST BE 2 NUMBERS LON
G"
7110 ? "3. NUMBER OF KILOWATTS USED IN
 MONTH"
7120 ? "4. NUMBER OF DAYS IN BILLING P
ERIOD"
7130 ? "5. NET COST OF ELECTRICITY IN
BILLING      PERIOD"
7140 ? "6. HEATING DEGREE DAYS IN BILL
ING          PERIOD"
7150 ? "7. COOLING DEGREE DAYS IN BILL
ING          PERIOD"
7160 IF LINE<>999 THEN ? "↓LAST LINE O
F DATA YOU ENTERED WAS: ";LINE
7170 ? "↓NOW BEGIN TYPING NEW DATA LIN
ES."
7180 END
```

•

CHECKSUM DATA

(See pgs. 7-10)

```
10 DATA 626,571,316,11,87,650,765,970,
462,317,312,349,844,885,96,7261
220 DATA 748,501,730,152,156,203,79,76
9,55,604,722,11,498,617,840,6685
520 DATA 258,376,999,591,189,73,420,43
5,728,140,203,330,681,639,405,6467
2090 DATA 851,848,852,604,860,824,723,
279,394,487,725,125,171,153,332,8228
3050 DATA 496,276,286,497,761,837,185,
274,293,885,611,584,357,351,249,6942
3200 DATA 778,316,848,308,491,727,470,
84,494,758,865,148,662,750,783,8482
4100 DATA 314,839,306,495,552,174,156,
129,793,294,777,500,764,740,77,6910
5110 DATA 101,50,652,783,713,236,823,6
69,448,51,341,715,499,355,108,6544
6020 DATA 373,553,240,527,488,719,149,
800,872,557,326,764,751,18,104,7241
6170 DATA 658,789,616,635,828,389,85,5
36,63,546,626,457,719,248,372,7567
6410 DATA 109,812,106,297,806,999,992,
11,33,13,57,56,34,32,23,4380
6560 DATA 60,992,414,734,777,465,896,3
52,576,210,625,142,869,295,840,8247
```

```
6720 DATA 127,482,778,321,101,446,779,
309,95,995,804,758,980,109,893,7977
6870 DATA 166,821,716,350,473,886,594,
695,429,247,84,642,143,82,144,6472
7120 DATA 531,305,47,98,613,435,276,23
05
```

•

Snowflake Demo

```
10 REM *** SNOWFLAKE GENERATOR ***
20 REM
30 REM BY TOM HUDSON
40 REM
50 REM SET UP GRAPHICS MODE, COLORS
60 REM
70 GRAPHICS 8+16:SETCOLOR 2,0,0:COLOR
1
80 REM
90 REM SET UP DEGREES, X AND Y TABLES
100 REM
110 DEG :DIM D(10),X(10),Y(10)
120 REM
130 REM RANDOMIZE SHAPE
140 REM
150 FOR I=1 TO 10:D(I)=0:X(I)=RND(0)*8
0:Y(I)=RND(0)*I*4:NEXT I:POKE 77,0
160 REM
170 REM ECHO AND ROTATE SHAPE
180 REM
190 PLOT 160,96:FOR I=1 TO 10:DRAWTO 1
60+(X(I)*COS(D(I))+Y(I)*SIN(D(I))),96+
(-X(I)*SIN(D(I))+Y(I)*COS(D(I)))
200 D(I)=D(I)+60:NEXT I:IF D(1)<360 TH
EN 190
210 FOR I=1 TO 10:D(I)=0:NEXT I
220 PLOT 160,96:FOR I=1 TO 10:DRAWTO 1
60+(X(I)*COS(D(I))-Y(I)*SIN(D(I))),96+
(-X(I)*SIN(D(I))-Y(I)*COS(D(I)))
230 D(I)=D(I)+60:NEXT I:IF D(1)<360 TH
EN 220
240 REM
250 REM LEAVE IT ON SCREEN A WHILE
260 REM
270 FOR DELAY=1 TO 5000:NEXT DELAY:RUN
```

TYPING TRAINER

16K Cassette 24K Disk

by Regena

Typing Trainer utilizes color, graphics and sound to help a student practice typing sentences for accuracy. There are 40 different 30-stroke sentences that are chosen randomly for the drills. Each drill consists of ten different sentences.

A sentence is shown on the screen. The student types and enters it. If it is incorrect, an "uh-oh" sounds and a "wrong" score is posted. The student has time to review the sentence before continuing. If the typed sentence is correct, a "right" score is posted, a train whistle sounds, and there are two blasts of steam from the engine's smokestack.

The running total score is displayed on the screen after each sentence. After ten sentences, the final score is displayed and a tune is played.

Following each drill of ten sentences, the student may choose whether to try again or not. If "N" for "no" is entered, the program ends. If "Y" for "yes" is entered, the drill is repeated with ten different sentences. Each drill chooses the sentences randomly, and the drill may be performed four times without sentences being repeated. After that, the sentences are all available for four more drills. The drills will be different each time because the sentences are chosen randomly. This process continues as long as the student wishes to continue.

Programming techniques.

ATARI does not allow arrays of string variables, so an array of sentence numbers is used. The sentences are numbered 1 through 40, where J is the number. Initially, all A(J)s are set to zero. After a sentence is used, A(J)=1.

To print a sentence, first a number J is chosen as a random integer from 1 through 40 (Line 220). If A(J)=1 the sentence has been used before and may not be chosen again, so another J is chosen (Line 230). If A(J)=0, SEN$ is set equal to the Jth sentence and the program branches to the drill (Lines 232-250, 4000-4390).

After the drill has been performed four times (using FLAG as a counter), all A(J)s are reset to zero so the sentences are all available for use in the next drill (Line 180).

To avoid the possibility of the student "crashing" the program during responses, an INPUT procedure is avoided. Instead, the program looks at what key is pressed by using B=PEEK(764). Yes or no responses are received by the student pressing "Y" or "N". Any other key pressed is ignored.

When sentences are typed, the characters are printed as each key is pressed until "RETURN" is pressed (which indicates the student is finished typing the sentence). The control keys or SHIFTing are not allowed, since a typist practicing sentences should not backspace and type over letters, nor type capital letters in the middle of the sentence (actually, the student types all capital letters in the standard computer mode but does not SHIFT). If a control key or SHIFT is pressed, an asterisk is printed in that character position of the student's sentence.

To avoid scrolling, the student is permitted to type only 34 characters in the sentence (Line 2005). The student's sentence is compared with the given sentence either after "RETURN" is pressed or after 34 characters have been typed. □

Explanation of the program.

Variables Used

J	Sentence number.
A(J)	=0 for available sentence, =1 if sentence has been used.
FLAG	Counter for number of times drill is performed.
WS	Wrong score.
RS	Right score.
PROB	Counter for number of sentences.
R	=1 if sentence is typed correctly, =0 if sentence is typed incorrectly.
D	Counter in delay loop for SOUND.

B	Value in PEEK(764) for key pressed.
BB	=1 for "yes" response, =0 for "no" response.
I	Counter in loop.
C, L	ASCII value.
SEN$	Typing sentence.
OLDB	Holding variable for B value.
K	Counter for number of characters printed in student's sentence.
C$	Character for key pressed.
T$	Student's typed sentence.
X,X1,Y,Y1,II, X2,Y2,X3,Y3	Coordinates for graphics.

Line Numbers	Procedure
10	Prints title screen and plays music.
30	Prints instruction screen.
100	DIMensions variables.
120	Reads in data for ASCII codes related to key pressed.
180-200	Initializes variables.
202	Draws train.
205	Initializes score to be zero.
210	Performs the drill for 10 sentences.
220-230	Randomly chooses a sentence; if the sentence has been used previously, chooses another one.
232-250	Depending on the J chosen, prints the corresponding sentence and prints the student's sentence; compares sentences.
255-280	If sentence is incorrect, sounds "uh-oh" and increments wrong score.
300-310	If sentence is correct, train toots whistle and blows steam; increments right score.
320	Prints running score.
330-345	Short delay for correct sentence, longer delay for incorrect sentence.
350	A(J)=1 indicates sentence J has been used and will not be available to use again.
355-360	Clears text screen and goes to next sentence.
370	After ten sentences, prints total score on full screen and plays music.
400-495	Asks the student to "try again?" and waits for the student to press "Y" or "N."
496	If the student pressed "N," ends program.
510-530	If the student pressed "Y," increments the number of times the drill was performed, If the drill has been performed 4 times, resets all sentences to be available; branches to beginning of drill.
999	End.

Subroutines

1000-1060	Subroutine reads DATA for assigning ASCII code to key pressed for use in printing.
1900-2500	Subroutine prints the sentence and accepts student's sentence.
1905	Prints the sentence.
1910-1930	Sounds a "beep" to indicate the student's turn to type.
2000	Initializes variables.
2005	Allows student to input up to 34 characters.
2010-2400	Prints each character as the student types it. If the student tries to press a control or SHIFTed character, "*" is printed. The student presses "RETURN" to end the sentence.
2410-2500	Sets R=1 if the sentence typed matches the given sentence, otherwise R=0, then returns.
4000-4390	The given 30-stroke typing sentences.
5000-6840	Subroutine draws the train and coal car.
7000-7490	Subroutine prints title screen and plays music.
8000-8160	Subroutine prints instructions and waits for student to press "RETURN" to continue.
9000-9080	Subroutine prints score and plays music.

```
10 GRAPHICS 18:GOSUB 7000
30 GOSUB 8000
100 DIM A(40),L(63),SEN$(30),T$(35),C$
(1),N$(1)
120 GOSUB 1000
180 FOR J=1 TO 40:A(J)=0:NEXT J
200 FLAG=0
202 GOSUB 5000
205 WS=0:RS=0
210 FOR PROB=1 TO 10
220 J=INT(40*RND(1))+1
230 IF A(J)=1 THEN 220
232 IF J>30 THEN 248
234 IF J>20 THEN 244
236 IF J>10 THEN 240
238 ON J GOSUB 4000,4010,4020,4030,404
0,4050,4060,4070,4080,4090
239 GOTO 255
240 JJ=J-10
242 ON JJ GOSUB 4100,4110,4120,4130,41
40,4150,4160,4170,4180,4190
243 GOTO 255
244 JJ=J-20
245 ON JJ GOSUB 4200,4210,4220,4230,42
40,4250,4260,4270,4280,4290
246 GOTO 255
248 JJ=J-30
250 ON JJ GOSUB 4300,4310,4320,4330,43
40,4350,4360,4370,4380,4390
255 IF R=1 THEN 300
260 SOUND 0,84,10,14
264 FOR D=1 TO 40:NEXT D
268 SOUND 0,101,10,14
270 FOR D=1 TO 40:NEXT D
275 SOUND 0,0,10,0
280 WS=WS+1:GOTO 320
300 GOSUB 3000
310 RS=RS+1
```

```
320 PRINT :PRINT R5;" RIGHT",W5;" WRON
G"
330 IF R=1 THEN 345
340 FOR D=1 TO 500:NEXT D
345 FOR D=1 TO 500:NEXT D
350 A(J)=1
355 PRINT :PRINT :PRINT
360 NEXT PROB
370 GOSUB 9000
400 GRAPHICS 0
410 PRINT :PRINT :PRINT
420 PRINT "DO YOU WANT TO TRY AGAIN?"
430 PRINT :PRINT "PRESS 'Y' FOR YES"
440 PRINT "       'N' FOR NO"
450 B=PEEK(764)
460 IF B=43 THEN BB=1:GOTO 490
470 IF B=35 THEN BB=0:GOTO 490
480 GOTO 450
490 SOUND 0,23,10,8
492 FOR D=1 TO 10:NEXT D
494 SOUND 0,0,10,8
495 POKE 764,255:B=255
496 IF BB=0 THEN 999
500 PRINT "R"
510 FLAG=FLAG+1:IF FLAG=3 THEN 180
530 GOTO 202
999 END
1000 FOR I=0 TO 63
1010 READ C:L(I)=C:NEXT I
1040 DATA 76,74,59,0,0,75,43,42,79,0,8
0,85,0,73,45,61,86,0,67,0,0,66,88,90,5
2,0,51,54,0,53,50
1050 DATA 49,44,32,46,78,0,77,47,0,82,
0,69,89,0,84,87,81,57,0,48,55,0,56,60,
62,70,72,68,0,0,71,83,65
1060 RETURN
1900 POKE 764,255:B=255
1905 PRINT SEN$
1910 SOUND 0,47,10,14
1920 FOR D=1 TO 60:NEXT D
1930 SOUND 0,0,10,0
2000 OLDB=-1:T$="":OPEN #1,4,0,"K:"
2005 FOR K=1 TO 34
2010 GET #1,B:IF B=155 THEN 2400
2020 IF B>96 THEN C$="*":GOTO 2065
2060 C$=CHR$(B)
2065 PRINT C$;:T$(LEN(T$)+1)=C$
2080 NEXT K
2090 GOTO 2400
2100 I=INT(PEEK(53775)/4):IF (I/2)=INT
(I/2) THEN 2010
2110 POKE 764,255:OLDB=-1
2120 GOTO 2010
2400 CLOSE #1
2410 IF T$=SEN$ THEN R=1:GOTO 2500
2420 R=0
2500 RETURN
3000 FOR II=1 TO 2
3010 SOUND 0,50,10,14:SOUND 1,63,10,14
3025 COLOR 2
3030 GOSUB 3500
3040 FOR D=1 TO 100:NEXT D
3050 SOUND 0,0,10,0:SOUND 1,0,10,0
3070 COLOR 0:GOSUB 3500
3090 NEXT II:RETURN
3500 PLOT 121,3
3510 PLOT 125,14:DRAWTO 126,10
3530 PLOT 124,14:DRAWTO 125,0
3550 PLOT 123,14:DRAWTO 123,0
3570 PLOT 122,11:DRAWTO 121,4
3590 RETURN
4000 SEN$="HE FEELS SHE HAS A SAFE LEA
SE.":GOTO 1900
4010 SEN$="ANDY MUST GIVE MY BAND A HA
ND.":GOTO 1900
4020 SEN$="SHE IS STILL AT THE LAKE SI
TE.":GOTO 1900
4030 SEN$="THERE IS A QUICK QUIZ FOR H
IM.":GOTO 1900
4040 SEN$="JUST SOME OF US HAVE TO DO
IT.":GOTO 1900
4050 SEN$="TWO OF THE GIRLS ARE HERE N
OW.":GOTO 1900
4060 SEN$="JANE STARTS HER TALK AT THR
EE.":GOTO 1900
4070 SEN$="TRY NOT TO LOOK AT YOUR HAN
DS.":GOTO 1900
4080 SEN$="HE DID SEEK AID FOR THE TRU
CK.":GOTO 1900
4090 SEN$="CHECK THE PAPER FOR ANY MAR
KS.":GOTO 1900
4100 SEN$="IT IS THIS DESK FILE HE SEE
KS.":GOTO 1900
4110 SEN$="HE KNOWS HE MUST KEEP WORKI
NG.":GOTO 1900
4120 SEN$="WE WOULD GIVE HIM A GOOD WA
GE.":GOTO 1900
4130 SEN$="BRING ALL BOOKS TO THE TABL
ES.":GOTO 1900
4140 SEN$="I HOPE THAT TAX DOES NOT PA
SS.":GOTO 1900
4150 SEN$="GREG BROUGHT IN A LARGE CHE
CK.":GOTO 1900
4160 SEN$="IT IS UP TO THEM TO WORK HA
RD.":GOTO 1900
4170 SEN$="PUT A LITTLE MORE EFFORT HE
RE.":GOTO 1900
4180 SEN$="HAVE A GOAL; WORK TO REACH
IT.":GOTO 1900
4190 SEN$="ALL GLAD DADS HAD A GLASS J
AR.":GOTO 1900
4200 SEN$="IT IS HOW WE WORK THAT COUN
TS.":GOTO 1900
4210 SEN$="TOM WAS QUICK TO SEND THE B
OX.":GOTO 1900
4220 SEN$="REX WILL HAVE MUCH MORE TO
DO.":GOTO 1900
4230 SEN$="I WILL GO TO TOWN TO GET TH
EM.":GOTO 1900
4240 SEN$="HE CAN LEND A HAND TO THE B
OY.":GOTO 1900
4250 SEN$="I PAID THE MEN FOR THEIR WO
RK.":GOTO 1900
4260 SEN$="THE WORKER SAID HE STRUCK O
IL.":GOTO 1900
4270 SEN$="SHE SAID WE NEED A NEW CAMP
ER.":GOTO 1900
4280 SEN$="I BOUGHT THE BIG BOX OF BOO
KS.":GOTO 1900
4290 SEN$="WE SHOULD SET A GOAL FOR TH
EM.":GOTO 1900
4300 SEN$="TRY TO TYPE ALL THE BIG WOR
DS.":GOTO 1900
4310 SEN$="WE MAY QUIT THIS WORK AT FI
VE.":GOTO 1900
4320 SEN$="YOU HAVE TO WORK FOR TWO DA
YS.":GOTO 1900
4330 SEN$="TRY TO GET ONE OR TWO OF TH
EM.":GOTO 1900
4340 SEN$="YOUR BEST MEN WILL HELP DO
IT.":GOTO 1900
4350 SEN$="HAVE THE BOYS DO THE WORK N
OW.":GOTO 1900
4360 SEN$="LET HIM PROVE THE RIGHT THI
NG.":GOTO 1900
4370 SEN$="THEY SHOULD READ MY GOOD BO
OK.":GOTO 1900
4380 SEN$="SHE CAN DO A BIG JOB THE BE
ST.":GOTO 1900
4390 SEN$="DAVE MADE A CAGE FOR HIS PE
TS.":GOTO 1900
5000 GRAPHICS 7:COLOR 1
5005 COLOR 1
5010 FOR Y=20 TO 25
5020 PLOT 55,Y:DRAWTO 88,Y
5040 NEXT Y
5050 FOR Y=26 TO 37
5060 PLOT 60,Y:DRAWTO 65,Y
5080 PLOT 83,Y:DRAWTO 88,Y
5100 NEXT Y
5110 FOR Y=38 TO 58
5120 PLOT 60,Y:DRAWTO 130,Y
5140 NEXT Y
5150 FOR Y=34 TO 37
5160 PLOT 97,Y:DRAWTO 103,Y
5180 NEXT Y
5190 PLOT 98,33:DRAWTO 102,33
5210 PLOT 100,32:PLOT 122,38
5230 DRAWTO 118,18
5240 DRAWTO 122,15
5250 DRAWTO 126,15
5260 DRAWTO 130,18
5270 DRAWTO 126,38
5280 COLOR 2
```

```
5290 PLOT 59,58:DRAWTO 50,58
5310 FOR X=49 TO 19 STEP -1
5320 PLOT X,40:DRAWTO X,58
5340 NEXT X
5350 COLOR 3
5360 X1=120:Y1=56
5370 GOSUB 6000
5380 X2=80:Y2=48
5390 GOSUB 6500
5392 X3=37:Y3=59:GOSUB 6200
5395 X3=27:Y3=59:GOSUB 6200
5400 FOR II=2 TO 4
5410 PLOT II*10,39
5420 DRAWTO II*10+8,39
5430 PLOT II*10+2,38
5440 DRAWTO II*10+7,38
5450 PLOT II*10+3,37
5460 DRAWTO II*10+7,37
5470 PLOT II*10+5,36
5480 NEXT II
5490 RETURN
6000 PLOT X1,Y1
6010 DRAWTO X1+4,Y1
6020 DRAWTO X1+7,Y1+3
6030 DRAWTO X1+7,Y1+7
6040 DRAWTO X1+4,Y1+10
6050 DRAWTO X1,Y1+10
6060 DRAWTO X1-3,Y1+7
6070 DRAWTO X1-3,Y1+3
6080 DRAWTO X1,Y1
6090 RETURN
6200 COLOR 3
6205 PLOT X3,Y3
6210 DRAWTO X3+4,Y3
6220 DRAWTO X3+6,Y3+2
6230 DRAWTO X3+6,Y3+6
6240 DRAWTO X3+4,Y3+8
6250 DRAWTO X3,Y3+8
6260 DRAWTO X3-2,Y3+6
6270 DRAWTO X3-2,Y3+2
6280 DRAWTO X3,Y3
6290 PLOT X3+2,Y3+4
6300 RETURN
6500 PLOT X2,Y2
6510 DRAWTO X2+6,Y2
6520 PLOT X2+7,Y2+1
6530 PLOT X2+8,Y2+1
6540 PLOT X2+9,Y2+2
6550 PLOT X2+10,Y2+3
6560 PLOT X2+11,Y2+4
6570 PLOT X2+11,Y2+5
6580 PLOT X2+12,Y2+6
6590 DRAWTO X2+12,Y2+12
6600 PLOT X2+11,Y2+13
6610 PLOT X2+11,Y2+14
6620 PLOT X2+10,Y2+15
6630 PLOT X2+9,Y2+16
6640 PLOT X2+8,Y2+17
6650 PLOT X2+7,Y2+17
6660 PLOT X2+6,Y2+18
6670 DRAWTO X2,Y2+18
6680 PLOT X2-1,Y2+17
6690 PLOT X2-2,Y2+17
6700 PLOT X2-3,Y2+16
6710 PLOT X2-4,Y2+15
6720 PLOT X2-5,Y2+14
6730 PLOT X2-5,Y2+13
6740 PLOT X2-6,Y2+12
6750 DRAWTO X2-6,Y2+6
6760 PLOT X2-5,Y2+5
6770 PLOT X2-5,Y2+4
6780 PLOT X2-4,Y2+3
6790 PLOT X2-3,Y2+2
6800 PLOT X2-2,Y2+1
6810 PLOT X2-1,Y2+1
6820 PLOT X2+3,Y2+9
6830 DRAWTO X1+2,Y1+5
6840 RETURN
7000 POSITION 3,3:PRINT #6;"TYPING"
7020 POSITION 3,5:PRINT #6;"TRAINER"
7040 SOUND 0,50,10,8
7060 FOR D=1 TO 50:NEXT D
7090 SOUND 0,0,10,8
7100 SOUND 0,50,10,8
7120 FOR D=1 TO 25:NEXT D
7130 SOUND 0,0,10,8:SOUND 0,50,10,8
7140 FOR D=1 TO 25:NEXT D
7150 SOUND 0,42,10,8
7170 FOR D=1 TO 50:NEXT D
7200 SOUND 0,0,10,0
7210 SOUND 0,42,10,8
7230 FOR D=1 TO 25:NEXT D
7240 SOUND 0,50,10,8
7260 FOR D=1 TO 25:NEXT D
7270 SOUND 0,63,10,8
7290 FOR D=1 TO 25:NEXT D
7300 SOUND 0,0,10,8:SOUND 0,63,10,8
7310 FOR D=1 TO 25:NEXT D
7320 SOUND 0,56,10,8
7340 FOR D=1 TO 25:NEXT D
7350 SOUND 0,0,10,8:SOUND 0,56,10,8
7360 FOR D=1 TO 25:NEXT D
7370 SOUND 0,50,10,8
7410 FOR D=1 TO 50:NEXT D
7420 SOUND 0,63,10,8
7430 SOUND 1,127,10,2
7440 SOUND 2,101,10,2
7450 FOR D=1 TO 100:NEXT D
7460 SOUND 0,0,10,0
7470 SOUND 1,0,10,0
7480 SOUND 2,0,10,0
7490 RETURN
8000 GRAPHICS 0
8010 PRINT :PRINT
8020 PRINT "YOU WILL SEE A SENTENCE"
8030 PRINT "ON THE SCREEN."
8040 PRINT :PRINT "TYPE AND ENTER IT."
8050 PRINT :PRINT "IF IT IS CORRECT,"
8060 PRINT "THE TRAIN WHISTLE WILL BLO
W."
8065 PRINT :PRINT "IF IT IS INCORRECT,
 YOU WILL"
8066 PRINT "HAVE TIME TO CHECK YOUR TY
PING."
8070 PRINT :PRINT "YOU WILL BE SHOWN Y
OUR SCORE"
8080 PRINT "AFTER EACH SENTENCE."
8090 PRINT :PRINT "AFTER TEN SENTENCES
"
8100 PRINT "YOUR FINAL SCORE IS SHOWN.
"
8120 PRINT :PRINT
8130 PRINT "PRESS 'RETURN' TO CONTINUE
."
8140 B=PEEK(764):IF B<>12 THEN 8140
8145 SOUND 0,23,10,8
8146 FOR D=1 TO 10:NEXT D
8147 SOUND 0,0,10,0
8150 POKE 764,255:B=255
8160 RETURN
9000 GRAPHICS 18
9010 POSITION 2,3
9020 PRINT #6;"RIGHT",R5
9030 POSITION 2,5
9040 PRINT #6;"WRONG",W5
9070 GOSUB 7040
9080 RETURN
```

●

CHECKSUM DATA

(See pgs. 7-10)

```
10 DATA 640,5,840,793,174,607,803,64,5
90,917,817,464,452,440,129,7735
239 DATA 725,450,983,722,456,5,725,462
,22,569,433,132,605,131,297,6717
280 DATA 648,797,529,544,574,503,508,5
03,494,221,836,890,479,154,971,8651
440 DATA 421,10,892,898,734,599,137,31
4,957,510,972,506,704,78,347,8079
1010 DATA 456,568,468,785,718,125,543,
534,489,973,356,618,247,934,929,8743
2080 DATA 500,714,75,289,708,867,537,1
,796,310,377,650,939,275,155,7193
3070 DATA 893,758,906,727,520,518,522,
807,631,817,116,872,875,959,913,10834
4070 DATA 154,798,920,707,961,772,91,9
87,905,747,41,774,711,956,15,9539
4220 DATA 901,20,757,28,124,753,764,92
4,68,41,932,13,971,923,988,8207
```

```
4370 DATA 959,698,810,980,651,559,484,
544,580,477,497,543,588,576,547,9493
5150 DATA 580,599,551,518,536,352,347,
352,353,363,656,683,320,485,550,7245
5350 DATA 657,34,949,205,961,382,387,3
30,299,21,398,19,400,19,405,5466
5480 DATA 740,808,136,698,730,735,975,
580,732,729,298,798,651,156,714,9480
6220 DATA 743,748,747,778,745,742,316,
662,798,153,722,669,673,679,523,9698
6560 DATA 528,532,537,902,722,724,725,
531,532,530,531,630,519,523,517,8983
6710 DATA 518,519,517,518,762,685,684,
680,676,665,663,682,747,817,597,9730
7020 DATA 897,341,520,496,340,515,45,5
17,344,524,477,343,519,347,522,6747
7270 DATA 356,525,54,520,354,523,59,52
5,353,527,357,528,516,296,489,5982
7470 DATA 491,493,812,56,589,969,673,4
4,159,494,590,861,734,421,423,7809
8100 DATA 155,593,773,106,350,523,494,
713,802,326,223,270,231,305,961,6825
9080 DATA 803,803
```

•

Graphics 8 Color Demo

```
10 GRAPHICS 8:SETCOLOR 2,0,15:SETCOLOR
 1,0,0:COLOR 1
20 FOR X=0 TO 200 STEP 2
30 PLOT X,0:DRAWTO X,10
40 NEXT X
50 FOR X=1 TO 201 STEP 2
60 PLOT X,20:DRAWTO X,30
70 NEXT X
80 FOR X=0 TO 200
90 PLOT X,40:DRAWTO X,50
100 NEXT X
```

CHECKSUM DATA
(See pgs. 7-10)

```
10 DATA 137,79,108,393,90,207,399,111,
225,758,2507
```

ENTERTAINMENT

MOTORCYCLE MAZE RIDER

16K Cassette 24K Disk

by Charles Bachand

Maze Rider is a game in which you roar through a twisting maze of tunnels on a motorcycle. You are hindered in this feat by the fact that your viewpoint is from inside the maze. The display is your window into the maze.

In order to play **Maze Rider**, a joystick must be inserted into port #1. After typing RUN the program will initialize and generate an introduction screen. The program will ask you to respond to questions about game options. The first question is "Do you want to leave a trail?" If the answer is yes, the game will display a line on the ground where you have previously traveled. The "Extra Passages" option will add more interconnecting passages to the maze. The map option allows you to see a map of the maze displaying an overhead view of the game area. Motorcycle noise can be eliminated in the last option if desired.

Pushing the joystick forward will move you forward within the maze. Pulling back on the joystick will make you move backward within the maze. Pushing the joystick to the left or right will change the direction that you are facing. Pushing the joystick to the left will make you turn in a counter clockwise direction and pushing to the right will make you turn in a clockwise direction. If the map option has been enabled, pushing the joystick trigger button will display an overhead view of the maze for about ten seconds.

For the technical types out there who are interested in how things work, the maze in this game is generated using a modified random walk routine that stores the X and Y locations it has traveled to into two tables, which are stored on page six of the computer's memory. As the cursor walks along, generating the maze, the X axis is stored at XPNT+PNTR. The index variable PNTR is then incremented by one. This operation continues until it runs into a dead end. At this point the program starts backtracking back to its origin. The index variable PNTR is decremented by one and the last X and Y coordinates are pulled from their locations in page six. The program then does LOCATEs up, down, left and right, looking for an unused space. If the program detects such a space around the cursor, the maze drawing process is turned back on. The cursor continues to advance and retreat until it bumps into its origin. □

Line	Explanation
100-640	Generates maze
640-830	Draws maze interior
840-1060	Main program routine
1070-1110	End of game.
1120-1130	Draws map of maze
1140	Draws outline of maze
1150-1230	Title and options select
1240	Perspective view data

```
100 REM *** MOTORCYCLE MAZE RIDER ***
110 REM * COPYRIGHT 1980  C.BACHAND *
120 REM
130 REM ***  FOR ANALOG MAGAZINE  ***
140 REM
150 TOP=PEEK(106):SWITCH=0
160 GOSUB 1150:GOSUB 1140
170 PRINT "↓)** GENERATING MAZE GRID *
*"
180 XC=INT(RND(Z)*((WIDTH-3)/2))*2+3
190 YC=INT(RND(0)*((LENGTH-3)/2))*2+3
200 EX=XC:EY=YC:XPNT=1536:YPNT=1632
210 SETCOLOR 1,0,14
220 COLOR 2:PLOT XC,YC:COLOR 1
230 LNG=INT(RND(0)*3)*2+2
240 DIR=INT(RND(0)*4)
250 S=(DIR=0)-(DIR=1)
260 T=(DIR=2)-(DIR=3)
270 FOR I=2 TO LNG STEP 2
280 LOCATE XC+S*I,YC+T*I,P
290 IF P AND I=2 THEN POP :GOTO 230
300 IF P THEN POP :LNG=2:GOTO 250
310 NEXT I:XC=XC+S*LNG:YC=YC+T*LNG
320 IF PNTR>PMAX THEN PMAX=PNTR:MX=XC:
MY=YC:MS=S:MT=T
330 DRAWTO XC,YC:PNTR=PNTR+1
340 SOUND 0,D2-PNTR*8,10,8
350 POKE XPNT+PNTR,XC
360 POKE YPNT+PNTR,YC
370 GOSUB 600:IF P THEN 390
380 SOUND 0,D2-PNTR*8,10,2:GOTO 230
390 XC=PEEK(XPNT+PNTR)
400 YC=PEEK(YPNT+PNTR)
410 PNTR=PNTR-1:GOSUB 600
420 SOUND 0,D2-PNTR*8,10,8
430 IF P AND PNTR THEN 390
440 POKE 77,Z:SOUND 0,D2-PNTR*8,10,2
```

```
450 PLOT XC,YC:IF PNTR THEN 230
460 COLOR 3:PLOT MX,MY:COLOR 1
470 MAP=ADR(MAP$):IF 1-EXTRA THEN 530
480 FOR I=1 TO 25
490 XC=INT(RND(0)*(WIDTH-4))+3
500 YC=INT(RND(0)*(LENGTH-4))+3
510 Y=(XC+YC)/2:IF INT(Y)=Y THEN 490
520 PLOT XC,YC:NEXT I
530 SOUND 0,0,0,0:FOR Y=1 TO LENGTH
540 FOR X=1 TO WIDTH:LOCATE X,Y,P
550 POKE MAP+Y*40+X,P:NEXT X:NEXT Y
560 S=-MS:T=-MT:MS=0:FOR I=0 TO 6
570 READ X:POKE XPNT+I,X:NEXT I:P3=0
580 YPNT=XPNT+8:POKE YPNT-1,79
590 FOR I=0 TO 6:POKE YPNT+I,79-(PEEK(
XPNT+I)+PEEK(XPNT+I-1))/4:NEXT I:GOTO
840
600 LOCATE XC+2,YC,P1
610 LOCATE XC-2,YC,P2
620 LOCATE XC,YC+2,P3
630 LOCATE XC,YC-2,P4
640 P=P1 AND P2 AND P3 AND P4:RETURN
650 P1=0:GRAPHICS 6:SETCOLOR 1,0,14:PO
KE 752,1:PRINT :PRINT "↓LOOKING ";A$,"
MOVES ";MOVE:MOVE=MOVE+1
660 FOR YC=0 TO 6:P2=NR(1,YC)
670 IF P2=2 THEN GOSUB 830
680 IF  NOT P2 THEN POP :GOTO 830
690 X1=P1:X2=PEEK(XPNT+YC):P1=X2
700 IF FEET AND YC THEN IF P2=3 AND NR
(1,YC-1)=3 THEN PLOT 79,PEEK(YPNT+YC-1
):DRAWTO 79,PEEK(YPNT+YC)
710 FOR XC=0 TO 2 STEP 2
720 IF XC THEN X1=158-X1:X2=158-X2
730 XD1=X1/2:XD2=X2/2
740 IF NR(XC,YC) THEN 760
750 PLOT X1,XD1:DRAWTO X2,XD2:PLOT X1,
79-XD1:DRAWTO X2,79-XD2:GOTO 790
760 PLOT X1,XD1:DRAWTO X1,79-XD1:PLOT
X1,XD2:DRAWTO X2,XD2:PLOT X1,79-XD2:DR
AWTO X2,79-XD2
770 IF NR(1,YC+1) THEN DRAWTO X2,XD2
780 GOTO 800
790 P2=NR(1,YC+1):IF P2=0 OR P2=2 THEN
 DRAWTO X2,XD2
800 IF FEET THEN IF YC AND NR(XC,YC)=3
 THEN PLOT 79,PEEK(YPNT+YC):DRAWTO X1,
PEEK(YPNT+YC)
810 NEXT XC:NEXT YC:IF  NOT NR(1,7) TH
EN IF NR(0,6) OR NR(2,6) THEN PLOT 79,
39:PLOT 79,40
820 RETURN
830 PLOT X2,XD2:DRAWTO 159-X2,XD2:PLOT
 X2,79-XD2:DRAWTO 159-X2,79-XD2:RETURN
840 SOUND 1,250,2,SND*4:COLOR 1:MAP=AD
R(MAP$):IF T<>1 THEN 860
850 FOR XC=-1 TO 1:FOR YC=0 TO 7:NR(XC
+1,YC)=PEEK(MAP+(MY+YC)*40+MX-XC):NEXT
 YC:NEXT XC:A$="SOUTH":GOTO 920
860 IF T<>-1 THEN 880
870 FOR XC=-1 TO 1:FOR YC=0 TO 7:NR(XC
+1,YC)=PEEK(MAP+(MY-YC)*40+MX+XC):NEXT
 YC:NEXT XC:A$="NORTH":GOTO 920
880 IF S<>-1 THEN 900
890 FOR XC=-1 TO 1:FOR YC=0 TO 7:NR(XC
+1,YC)=PEEK(MAP+(MY-XC)*40+MX-YC):NEXT
 YC:NEXT XC:A$="WEST":GOTO 920
900 IF S<>1 THEN 920
910 FOR XC=-1 TO 1:FOR YC=0 TO 7:NR(XC
+1,YC)=PEEK(MAP+(MY+XC)*40+MX+YC):NEXT
 YC:NEXT XC:A$="EAST"
920 POKE 54286,0:SWITCH=16-SWITCH:POKE
 106,TOP-SWITCH:GOSUB 650:POKE 54286,6
4:POKE 77,0
930 IF STICK(0)<13 THEN 930
940 IF STRIG(0) OR MAPSW=0 THEN 980
950 IF P3>2 THEN SOUND 0,50,12,6:PRINT
 "↓↓↓THREE LOOKS IS YOUR LIMIT":FOR I=
1 TO 100:NEXT I:GOTO 980
960 GOSUB 1140:P3=P3+1:SETCOLOR 1,0,14
:PRINT "↓↓CHECK MOTORCYCLE MAZE MAP #"
;P3:GOSUB 1120
970 FOR X=1 TO 10:FOR P=1 TO 4:FOR I=1
 TO 10:NEXT I:COLOR P:PLOT MX,MY:NEXT
P:NEXT X:SOUND 0,0,0,0:GOTO 840
980 SOUND 0,0,0,0:P=STICK(0):IF P=15 O
R P=5 OR P=6 OR P=9 OR P=10 THEN 940
990 IF P=14 THEN MX=MX+S:MY=MY+T:SOUND
 0,120,6,SND*6:IF  NOT PEEK(MAP+MY*40+
MX) THEN MX=MX-S:MY=MY-T:P=0
1000 IF P=13 THEN MX=MX-S:MY=MY-T:SOUN
D 0,120,6,SND*6:IF  NOT PEEK(MAP+MY*40
+MX) THEN MX=MX+S:MY=MY+T:P=0
1010 IF P=7 OR P=11 THEN P1=S:S=-T:T=P
1
1020 IF P=11 THEN S=-S:T=-T
1030 IF P=0 THEN PRINT "↓↓↓↓CRASH!!":F
OR P=15 TO 0 STEP -1:SOUND 0,120,12,P:
FOR I=1 TO 5:NEXT I:NEXT P:P=0:MS=0
1040 I=MAP+MY*40+MX:IF PEEK(I)=2 THEN
1070
1050 POKE I,3:IF P THEN 840
1060 GOTO 940
1070 PRINT "↓↓↓***** YOU ARE FREE ****
*":FOR X=1 TO 5:FOR Y=200 TO 0 STEP -4
1080 SOUND Z,Y,10,X*3:NEXT Y:FOR I=1 T
O 4:PLOT RND(0)*159,0:DRAWTO RND(0)*15
9,79:NEXT I:NEXT X:SOUND Z,Z,Z,Z
1090 POKE 106,TOP
1100 FOR I=1 TO 100:NEXT I:GOSUB 1140:
SETCOLOR 1,0,14:PRINT "↓↓↓ *** YOU'RE
FINAL MAP ***":GOSUB 1120
1110 POKE 752,0:END
1120 MAP=ADR(MAP$):FOR Y=3 TO LENGTH-2
:FOR X=3 TO WIDTH-2:COLOR PEEK(MAP+Y*4
0+X)
1130 SOUND 0,290-Y*14-X,10,6:PLOT X,Y:
NEXT X:NEXT Y:RETURN
1140 GRAPHICS 3:COLOR 1:PLOT 1,1:DRAWT
O WIDTH,1:DRAWTO WIDTH,LENGTH:DRAWTO 1
,LENGTH:DRAWTO 1,1:POKE 752,1:RETURN
1150 GRAPHICS 2:SETCOLOR 1,0,14:PRINT
#6;"   / motorcycle \":PRINT #6;"  /
maze rider  \":PRINT #6:OPEN #1,4,0,"K
:"
1160 WIDTH=39:LENGTH=19:DP=96:D2=DP*8
1170 DIM MAP$(800),A$(5),C$(1),NR(2,7)
1180 PRINT #6:PRINT #6:PRINT #6;" ANALOG 400/800 ":PRINT #6;" Magazine ":PRINT #6
1190 PRINT "↓↓↓   DO YOU WANT TO LEAVE
A TRAIL";:GET #1,A:IF CHR$(A)="Y" THEN
 FEET=1
1200 PRINT "↓↓↓   DO YOU WANT EXTRA PAS
SAGES";:GET #1,A:IF CHR$(A)="Y" THEN E
XTRA=1:GOTO 1210
1210 PRINT "↓↓↓   DO YOU WANT TO USE TH
E MAP";:GET #1,A:IF CHR$(A)="Y" THEN M
APSW=1:GOTO 1220
1220 PRINT "↓↓↓   DO YOU WANT MOTORCYCL
E SOUND";:GET #1,A:IF CHR$(A)="Y" THEN
 SND=1
1230 RETURN
1240 DATA 0,28,46,60,68,74,78
```

•

CHECKSUM DATA

(See pgs. 7-10)

```
100 DATA 973,829,80,789,86,469,439,385
,404,693,788,697,18,368,835,7853
250 DATA 747,761,569,387,358,457,828,5
40,163,271,171,179,514,336,458,6739
400 DATA 438,712,267,423,694,456,100,9
63,158,901,122,719,82,430,631,7096
550 DATA 867,379,93,341,481,828,838,76
9,779,958,394,608,285,382,423,8425
700 DATA 88,115,600,785,415,343,948,76
8,733,903,766,874,601,208,806,8953
850 DATA 254,789,244,770,417,496,244,3
17,487,497,509,527,451,885,494,7381
1000 DATA 548,49,452,784,177,429,896,3
22,878,159,677,368,241,580,79,6639
1150 DATA 298,82,185,147,995,426,316,4
95,788,240,3972
```

•

DINO BATTLE

24K Cassette 32K Disk

by Art V. Cestaro III

Dino Battle is a game of primordial confrontation, a fierce battle between two players. See if you can defeat a dinosaur!

Your goal is to bite your opponent's dinosaur on the back of the neck. By moving your joystick and pressing the firing button, you can move your dinosaur and open and close his mouth. You may make a number of attempts before you succeed. Try to bite your opponent as many times as you can before the time is up.

Your score is displayed on the side of each dinosaur at the start of the game. You receive one point each time you bite the other dinosaur. □

Line	Explanation
Y, Y1	Vertical position of dinosaur 1
3	Sets GRAPHIC mode and colors
12	Sets time and score
13-16	Draws landscape
80-81	Prints text
100-200	Main loop: checks joystick and triggers and increments time
300-315	Moves dinosaur figures on screen
1000-1015	Turns dinosaur number 1 around
1100-1115	Turns dinosaur number 2 around
3500-3595	Makes dinosaur 1 open his mouth and try to bite the other one
3600-3710	Makes dinosaur 2 do the same thing
3800	Prints both players' scores
3900-3905	Plots cacti
3910-3930	Plots rocks
4000-4021	Plots dinosaur 1, fall routine
4500-4531	Plots dinosaur 2, fall routine
4600	Erases the dinosaur
4800-4810	Moves dinosaur away from defeated opponent
5000-5990	Plots title
7000-7110	Opening display
8000-8220	End of game
10000-10035	Sets up player/missile graphics
10040-11000	Reads shape data and stores it in the proper arrays
12000-12900	Data for shapes

Name	Variable
Time	Time in seconds of the game
Score 1	Players' scores
Score 2	
TT	Timing variable
X	Horizontal position of dinosaur 1
X2	Horizontal position of dinosaur 2
DR1	Direction dinosaur 1 is facing
DR2	Direction dinosaur 2 is facing
DF1	Area in memory where player data is poked
DB1	
DF2	
DB2	
Y, Y1	Vertical position of dinosaur 1
Y2, Y3	Vertical position of dinosaur 2
RT, RET, RT1	Return Flags
G, H, DD	Dummy variables
C, Z,	
I	Top of RAM: used for setting up player/missile area

Arrays	
TF1, TF2	Flying dinosaur's front
TB1, TB2	Flying dinosaur's back
D1NF1	
D1NF2	Dinosaur front and back views
D1NB1	
D1NB2	
DHR	Dinosaur's head and mouth open

Each dinosaur is made up of two players, positioned next to each other so they make up one dinosaur shape. □

```
0 REM  DINO BATTLE  REV 1.0
1 REM By Art V Cestaro III 10/13/81
3 GRAPHICS 7:CLR :POKE 752,1:POKE 712,
197:POKE 710,24:POKE 708,99:POKE 709,1
95
6 GOSUB 3930
12 TIME=59:TIM=0:SCORE1=0:SCORE2=0:COL
OR 1
13 Y=INT(RND(0)*35+10):D=1:FOR X=0 TO
158 STEP 2:Y1=INT(15*RND(0)+Y-5*D):PLO
T X,47:DRAWTO X,Y:PLOT X+1,47
14 DRAWTO X+1,(Y+Y1)/2:Y=Y1:IF Y>40 TH
EN Y=Y-10:D=2
15 IF Y<20 THEN Y=Y+10:D=1
16 NEXT X
17 GOSUB 3900:GOSUB 3910
30 GOSUB 7000
75 RET=0:GOSUB 10000:GOSUB 1000:GOSUB
1100
80 POKE 752,1:POKE 656,0:POKE 657,3:?
"  SCORE  ":POKE 656,0:POKE 657,28:? "
 SCORE  "
81 POKE 656,0:POKE 657,12:? "|↓←|↓←|":
POKE 656,0:POKE 657,27:? "|↓←|↓←|":POK
E 656,0:POKE 657,16:? "  TIME  "
82 GOSUB 3800
100 TT=TT+0.2:IF TT>1 THEN TT=0:TIME=T
IME-1:IF TIME<1 THEN TIME=59:TIM=TIM-1
104 IF STICK(0)=7 THEN X=X+2:IF DR1=1
THEN GOSUB 1000
105 IF STRIG(0)=0 THEN RT=0:GOSUB 3500
110 IF STICK(1)=7 THEN X2=X2+2:IF DR2=
2 THEN GOSUB 1110
111 IF X<55 THEN X=55
112 IF X>195 THEN X=195
115 ON DR1 GOSUB 300,305
120 IF STICK(1)=11 THEN X2=X2-2:IF DR2
=1 THEN GOSUB 1100
130 IF STICK(0)=11 THEN X=X-2:IF DR1=2
 THEN GOSUB 1010
132 IF STRIG(1)=0 THEN RT1=0:GOSUB 360
0
133 IF X2<55 THEN X2=55
134 IF X2>195 THEN X2=195
135 ON DR2 GOSUB 310,315
169 IF TIM<1 AND TIME<2 THEN POKE 656,
2:POKE 657,18:? "0:00":GOTO 8000
172 IF TIME<10 THEN POKE 656,2:POKE 65
7,18:? TIM;":0";TIME:GOTO 180
175 POKE 656,2:POKE 657,18:? TIM;":";T
IME;" "
180 POKE 77,0
200 GOTO 100
300 POKE 53248,X:POKE 53249,X-8:RETURN
305 POKE 53249,X-8:POKE 53248,X:RETURN
310 POKE 53250,X2-8:POKE 53251,X2:RETU
RN
315 POKE 53251,X2:POKE 53250,X2-8:RETU
RN
1000 DR1=2:FOR G=1 TO 4:POKE DB1+G,0:N
EXT G:Y=65:Y1=69:DF1=Y+J:DB1=Y1+J1:FOR
 G=1 TO 18:POKE DB1+G,DINB1(G)
1005 POKE DF1+G,DINF1(G):NEXT G:FOR G=
19 TO 22:POKE DF1+G,DINF1(G):NEXT G:RE
TURN
1010 DR1=1:FOR G=1 TO 4:POKE DF1+G,0:N
EXT G:Y=69:Y1=65:DF1=Y+J:DB1=Y1+J1:FOR
 G=1 TO 18:POKE DF1+G,DINB2(G)
1015 POKE DB1+G,DINF2(G):NEXT G:FOR G=
19 TO 22:POKE DB1+G,DINF2(G):NEXT G:RE
TURN
1100 DR2=2:FOR G=1 TO 4:POKE DB2+G,0:N
EXT G:Y2=65:Y3=69:DF2=Y2+J2:DB2=Y3+J3:
FOR G=1 TO 18:POKE DF2+G,DINF2(G)
1105 POKE DB2+G,DINB2(G):NEXT G:FOR G=
19 TO 22:POKE DF2+G,DINF2(G):NEXT G:RE
TURN
1110 DR2=1:FOR G=1 TO 4:POKE DF2+G,0:N
EXT G:Y2=69:Y3=65:DF2=Y2+J2:DB2=Y3+J3:
FOR G=1 TO 18:POKE DF2+G,DINB1(G)
1115 POKE DB2+G,DINF1(G):NEXT G:FOR G=
19 TO 22:POKE DB2+G,DINF1(G):NEXT G:RE
TURN
3500 ON DR1 GOTO 3510,3520
3510 BB=DB1:GG=3590:GOTO 3550
3520 BB=DF1:GG=3580
3550 GOSUB GG
3555 FOR G=50 TO 100:SOUND 0,G,10,15:S
OUND 0,100-(G-50),10,15:NEXT G:SOUND 0
,0,0,0
3560 ON DR1 GOTO 3563,3565
3563 POKE BB,0:FOR G=1 TO 6:POKE BB+G,
DINF2(G):NEXT G:GOTO 3591
3565 POKE BB,0:FOR G=1 TO 6:POKE BB+G,
DINF1(G):NEXT G:GOTO 3591
3570 RETURN
3580 POKE BB+6,224:FOR G=0 TO 5:POKE B
B+G,DHR(G+1):NEXT G:RETURN
3590 POKE BB+6,7:FOR G=0 TO 5:POKE BB+
G,DHL(G+1):NEXT G:RETURN
3591 IF RT=1 THEN RETURN
3592 IF DR1=2 AND DR2=1 AND PEEK(53260
)=12 THEN GOSUB 4500
3593 IF DR1=1 AND DR2=2 AND PEEK(53261
)=12 THEN GOSUB 4500
3595 POKE 53278,0:RETURN
3600 ON DR2 GOTO 3610,3620
3610 BB=DB2:GG=3580:GOTO 3650
3620 BB=DF2:GG=3590
3650 GOSUB GG
3655 FOR G=50 TO 100:SOUND 0,G,10,15:S
OUND 0,100-(G-50),12,10:NEXT G:SOUND 0
,0,0,0
3660 ON DR2 GOTO 3663,3665
3663 POKE BB,0:FOR G=1 TO 6:POKE BB+G,
DINF1(G):NEXT G:GOTO 3700
3665 POKE BB,0:FOR G=1 TO 6:POKE BB+G,
DINF2(G):NEXT G
3700 IF RT1=1 THEN RETURN
3701 IF DR2=2 AND DR1=1 AND PEEK(53262
)=3 THEN GOSUB 4000
3705 IF DR2=1 AND DR1=2 AND PEEK(53263
)=3 THEN GOSUB 4000
3710 POKE 53278,0:RETURN
3800 POKE 656,2:POKE 657,6:? SCORE1;"
 ":POKE 656,2:POKE 657,31:? SCORE2;"
":RETURN
3900 COLOR 2:FOR J=1 TO 4:H=INT(45+RND
(0)*10):G=RND(0)*145+10:GOSUB 3903:NEX
T J:RETURN
3901 DRAWTO G+2,H+5:DRAWTO G+2,H+3:RET
URN
3903 PLOT G,H:DRAWTO G,H+9:PLOT G,H+4:
DRAWTO G-2,H+4:DRAWTO G-2,H+1:PLOT G,H
+5
3905 DRAWTO G+2,H+5:DRAWTO G+2,H+3:RET
URN
3910 COLOR 1:FOR J=1 TO 3:H=48+RND(0)*
10:G=RND(0)*145+10:GOSUB 3913:NEXT J:R
ETURN
3911 DRAWTO G+5,H+5:DRAWTO G+3,H+9:RET
URN
3913 PLOT G,H:DRAWTO G-5,H+5:DRAWTO G+
3,H+9:DRAWTO G,H:DRAWTO G+4,H+1
3915 DRAWTO G+5,H+5:DRAWTO G+3,H+9:RET
URN
3930 COLOR 3:FOR G=79 TO 47 STEP -1:PL
OT 0,G:DRAWTO 159,G:NEXT G:RETURN
4000 BB1=DF1:BB2=DB1:GOSUB 4600
4003 Y=75:Y1=74:DF1=Y+J:DB1=Y1+J1
4005 ON DR1 GOSUB 4010,4020
4006 RT1=1:GOSUB 3600:GOTO 4810
4010 FOR G=1 TO 9:POKE DB1+G,DLF(G):PO
KE DF1+G,DLB(G):SOUND 0,120,8,15-G:NEX
T G
4011 POKE DF1+10,DLB(10):POKE DF1+11,D
LB(11):FOR G=1 TO 6:SOUND 0,120,8,15-G
:FOR HH=1 TO 10:NEXT HH:NEXT G:RETURN
4020 FOR G=1 TO 9:POKE DB1+G,DRB(G):PO
KE DF1+G,DRF(G):SOUND 0,120,8,15-G:NEX
T G
4021 POKE DB1+10,DRB(10):POKE DB1+11,D
RB(11):FOR G=1 TO 6:SOUND 0,120,8,15-G
:FOR HH=1 TO 10:NEXT HH:NEXT G:RETURN
4500 BB1=DF2:BB2=DB2:GOSUB 4600
4503 Y2=74:Y3=75:DF2=Y2+J2:DB2=Y3+J3
4505 ON DR2 GOSUB 4520,4530
4510 RT=1:GOSUB 3500:GOTO 4800
4520 FOR G=1 TO 9:POKE DF2+G,DRB(G):PO
KE DB2+G,DRF(G):SOUND 0,110,8,15-G:NEX
T G
4521 POKE DF2+10,DRB(10):POKE DF2+11,D
RB(11):FOR G=1 TO 6:SOUND 0,110,8,15-G
:FOR HH=1 TO 10:NEXT HH:NEXT G:RETURN
```

```
4530 FOR G=1 TO 9:POKE DB2+G,DLB(G):PO
KE DF2+G,DLF(G):SOUND 0,110,8,15-G:NEX
T G
4531 POKE DB2+10,DLB(10):POKE DB2+11,D
LB(11):FOR G=1 TO 6:SOUND 0,110,8,15-G
:FOR HH=1 TO 10:NEXT HH:NEXT G:RETURN
4600 FOR G=1 TO 22:POKE BB1+G,0:POKE B
B2+G,0:NEXT G:RETURN
4800 X=INT(RND(0)*145+50):ON DR1 GOSUB
 300,305:GOSUB 1100:SCORE1=SCORE1+10:G
OSUB 3800:RETURN
4810 X2=INT(RND(0)*145+50):ON DR2 GOSU
B 310,315:GOSUB 1000:SCORE2=SCORE2+10:
GOSUB 3800:RETURN
5000 COLOR 1:PLOT 26,5:DRAWTO 26,15:PL
OT 26,5:DRAWTO 31,6:DRAWTO 31,14:DRAWT
O 26,15:GOTO 5990
5100 PLOT 36,5:DRAWTO 36,15:PLOT 35,5:
PLOT 37,5:PLOT 35,15:PLOT 37,15:GOTO 5
990
5200 PLOT 42,15:DRAWTO 42,5:DRAWTO 46,
15:DRAWTO 46,5:GOTO 5990
5300 PLOT 50,5:DRAWTO 50,15:DRAWTO 55,
15:DRAWTO 55,5:DRAWTO 50,5:GOTO 5990
5400 PLOT 66,5:DRAWTO 66,15:DRAWTO 71,
15:DRAWTO 71,5:DRAWTO 66,5:PLOT 66,10:
DRAWTO 71,10:GOTO 5990
5500 PLOT 76,5:DRAWTO 81,5:DRAWTO 81,1
5:PLOT 76,5:DRAWTO 76,15:PLOT 76,10:DR
AWTO 81,10:GOTO 5990
5600 PLOT 85,5:DRAWTO 91,5:PLOT 88,5:D
RAWTO 88,15:GOTO 5990
5700 PLOT 95,5:DRAWTO 101,5:PLOT 98,5:
DRAWTO 98,15:GOTO 5990
5800 PLOT 106,5:DRAWTO 106,15:DRAWTO 1
11,15:GOTO 5990
5900 PLOT 116,5:DRAWTO 116,15:DRAWTO 1
21,15:PLOT 116,10:DRAWTO 121,10:PLOT 1
16,5:DRAWTO 121,5:GOTO 5990
5990 RETURN
7000 DD=17:DIM TF1(DD),TB1(DD),TF2(DD)
,TB2(DD)
7005 FOR G=1 TO DD:TF1(G)=0:TF2(G)=0:T
B1(G)=0:TB2(G)=0:NEXT G
7010 FOR G=1 TO 14:READ C:TF1(G)=C:NEX
T G:FOR G=1 TO 13:READ C:TB1(G)=C:NEXT
 G
7020 FOR G=1 TO 13:READ C:TB2(G)=C:NEX
T G:FOR G=1 TO 14:READ C:TF2(G)=C:NEXT
 G
7025 RET=0:GOSUB 10000
7030 POKE 704,49:POKE 705,49:FOR G=5 T
O 19:POKE DF1+G,TF1(G-4):NEXT G:FOR G=
1 TO 13:POKE DB1+G,TB1(G):NEXT G
7040 FOR X=220 TO 35 STEP -1:POKE 5324
8,X-7:POKE 53249,X:SOUND 0,X,10,6:FOR
H=1 TO 3:NEXT H:NEXT X
7045 FOR G=1 TO 18:POKE DF1+G,0:POKE D
B1+G,0:NEXT G
7051 POKE 704,49:POKE 705,49:FOR G=1 T
O 13:POKE DB1+G,TB2(G):NEXT G:FOR G=4
TO 18:POKE DF1+G,TF2(G-3):NEXT G
7060 FOR X=30 TO 210:POKE 53249,X:POKE
 53248,X+7:SOUND 0,X,10,6
7062 IF X=75 THEN GOSUB 5000
7063 IF X=85 THEN GOSUB 5100
7064 IF X=91 THEN GOSUB 5200
7065 IF X=103 THEN GOSUB 5300
7066 IF X=119 THEN GOSUB 5400
7067 IF X=130 THEN GOSUB 5500
7068 IF X=138 THEN GOSUB 5600
7069 IF X=144 THEN GOSUB 5700
7070 IF X=155 THEN GOSUB 5800
7071 IF X=165 THEN GOSUB 5900
7075 FOR H=1 TO 4:NEXT H:NEXT X
7080 ? "{CLR}          By Art.V.Cestaro III
":SOUND 0,90,12,11:SOUND 1,91,12,12:GO
SUB 10040
7085 COLOR 0:FOR G=5 TO 10:PLOT 25,G:D
RAWTO 125,G:PLOT 25,15-(G-5):DRAWTO 12
5,15-(G-5):NEXT G
7086 SOUND 0,80,12,12:SOUND 1,81,12,14
7090 ? "{CLR}          PRESS START  "
7091 FOR G=1 TO 20:GOSUB 7098:NEXT G
7092 ? "{CLR}            PRESS START  "
7093 FOR G=1 TO 20:GOSUB 7098:NEXT G:G
OTO 7090
7098 IF PEEK(53279)=6 THEN POP :GOTO 7
100
7099 RETURN
7100 ? "{CLR}          OH OH ... FOOTSTEPS
"
7101 FOR G=1 TO 2:FOR H=15 TO 0 STEP -
1:SOUND 0,120,8,H
7105 SOUND 1,122,8,H:FOR J=1 TO 8:NEXT
 J:NEXT H:FOR F=1 TO 60:NEXT F
7107 FOR H=15 TO 0 STEP -1:SOUND 0,110
,8,H:SOUND 1,112,8,H:FOR J=1 TO 8:NEXT
 J:NEXT H:FOR F=1 TO 60:NEXT F:NEXT G
7110 ? "{CLR}":RETURN
8000 FOR G=1 TO 10:POKE 656,0:POKE 657
,15:? " GAME OVER ":SOUND 0,150,10,14:
FOR Z=1 TO 15:NEXT Z
8005 POKE 656,0:POKE 657,15:? "{RVS} GAME O
VER ":SOUND 0,100,10,14:FOR H=1 TO 15:
NEXT H:NEXT G
8009 SOUND 0,0,0,0:POKE 656,0:POKE 657
,13:? "{RVS} PRESS START "
8010 IF SCORE1>SCORE2 THEN 8020
8013 IF SCORE2>SCORE1 THEN 8030
8015 IF SCORE1=SCORE2 THEN 8040
8020 POKE 656,0:POKE 657,3:? "  SCORE
 ":FOR H=1 TO 15:GOSUB 8100:NEXT H
8021 POKE 656,0:POKE 657,3:? "{RVS}  SCORE
 ":FOR H=1 TO 15:GOSUB 8100:NEXT H:GOT
O 8020
8030 POKE 656,0:POKE 657,28:? "  SCORE
 ":FOR H=1 TO 15:GOSUB 8100:NEXT H
8035 POKE 656,0:POKE 657,28:? "{RVS}  SCORE
 ":FOR H=1 TO 15:GOSUB 8100:NEXT H:GO
TO 8030
8040 POKE 656,0:POKE 657,3:? "  SCORE
 ":POKE 656,0:POKE 657,28:? "  SCORE
":FOR H=1 TO 15:GOSUB 8100
8041 NEXT H
8045 POKE 656,0:POKE 657,3:? "{RVS}  SCORE
 ":POKE 656,0:POKE 657,28:? "{RVS}  SCORE
 ":FOR H=1 TO 15:GOSUB 8100
8046 NEXT H:GOTO 8040
8100 IF PEEK(53279)=6 THEN POP :GOTO 8
200
8101 RETURN
8200 SCORE1=0:SCORE2=0:TIM=0:TIME=59
8210 FOR G=250 TO 0 STEP -3:SOUND 0,G+
5,10,15:SOUND 1,G+4,10,14:SOUND 2,G+3,
10,13
8215 SOUND 3,G+2,10,12:POKE 712,RND(0)
*255:NEXT G:FOR G=0 TO 3:SOUND G,0,0,0
:POKE 53248+G,35:NEXT G
8217 POKE 712,197:GOSUB 3930:GOSUB 390
0:GOSUB 3910
8220 POKE 656,0:POKE 657,13:? "
     ":POKE 712,197:GOTO 75
10000 POKE 559,46:I=PEEK(106)-24:POKE
54279,I:POKE 53277,3:POKE 623,1
10010 J=I*256+512:J1=I*256+640:J2=I*25
6+768:J3=I*256+896
10015 FOR G=J TO J3+128:POKE G,0:NEXT
G
10020 POKE 704,165:POKE 705,165:POKE 7
06,220:POKE 707,220
10025 X=100:Y=17:Y1=16
10030 DF1=Y+J:DB1=Y1+J1:DF2=Y+J2:DB2=Y
+J3
10035 IF RET=0 THEN RETURN
10040 DD=22:DIM DINF1(DD),DINF2(DD),DI
NB1(DD),DINB2(DD),DHR(6),DHL(6)
10043 CC=11:DIM DRF(CC),DRB(CC),DLF(CC
),DLB(CC)
10045 FOR G=1 TO DD:DINF1(G)=0:DINF2(G
)=0:DINB1(G)=0:DINB2(G)=0:NEXT G
10050 RESTORE 12500:FOR G=1 TO 18:READ
 C:DINB1(G)=C:NEXT G:FOR G=1 TO 22:REA
D C:DINF1(G)=C:NEXT G
10060 RESTORE 12600:FOR G=1 TO 22:READ
 C:DINF2(G)=C:NEXT G:FOR G=1 TO 18:REA
D C:DINB2(G)=C:NEXT G
10065 RESTORE 12700:FOR G=1 TO 6:READ
C:DHR(G)=C:NEXT G:FOR G=1 TO 6:READ C:
DHL(G)=C:NEXT G
10066 RESTORE 12800:FOR G=1 TO 9:READ
C:DRF(G)=C:NEXT G:FOR G=1 TO CC:READ C
:DRB(G)=C:NEXT G
```

```
10068 RESTORE 12900:FOR G=1 TO 9:READ
C:DLF(G)=C:NEXT G:FOR G=1 TO CC:READ C
:DLB(G)=C:NEXT G
10070 X=100:X1=92:X2=150:X3=158:Y=64:Y
1=68:Y2=64:Y3=68
10071 DF1=Y+J:DB1=Y1+J1:DF2=Y2+J2:DB2=
Y3+J3
11000 RETURN
12000 DATA 1,6,28,47,63,87,175,31,28,5
6,56,24,12,4
12005 DATA 3,6,28,24,56,48,112,112,243
,252,248,249,158
12010 DATA 192,96,56,24,28,12,14,14,20
7,63,31,153,112
12020 DATA 128,96,56,244,252,234,245,2
48,56,28,28,24,48,32
12500 DATA 1,1,1,7,7,3,7,15,7,143,199,
143,198,158,188,240,224,64
12510 DATA 28,52,62,122,245,242,224,25
1,245,240,240,224,192,128,192,224,240,
112,48,96,96,248
12600 DATA 56,44,124,94,175,79,7,223,1
75,15,15,7,3,1,3,7,15,14,12,6,6,31
12610 DATA 128,128,128,224,224,192,224
,240,224,241,227,241,99,121,61,15,7,2
12700 DATA 76,104,208,254,240,224,50,2
2,11,127,15,7
12800 DATA 12,15,229,55,255,254,252,24
8,112,128,128,240,252,31,15,79,39,19,3
0,12
12900 DATA 48,240,160,231,252,127,63,3
1,14,1,1,143,63,248,240,242,228,200,12
0,48
```

•

CHECKSUM DATA
(See pgs. 7-10)

```
0 DATA 909,124,404,671,656,206,629,300
,405,209,3,392,475,873,35,6291
100 DATA 250,499,808,92,769,988,956,83
5,860,956,964,352,969,677,493,10468
175 DATA 493,966,685,140,162,188,78,38
3,495,400,490,657,488,658,495,6778
3500 DATA 5,527,591,975,775,52,885,887
,805,95,78,573,304,310,415,7277
3600 DATA 14,533,598,978,772,61,862,58
5,272,915,927,403,300,809,606,8635
3903 DATA 60,614,285,623,945,631,374,9
4,308,283,99,915,6,940,9,6186
4500 DATA 113,562,330,832,959,40,936,7
,440,600,942,627,924,621,208,8141
5400 DATA 86,69,289,636,440,811,823,78
4,757,768,778,998,189,221,432,8081
7051 DATA 557,619,754,762,766,447,463,
450,469,465,452,456,373,329,212,7574
7086 DATA 417,639,650,803,61,999,818,5
31,923,437,219,391,591,605,357,8441
8010 DATA 215,224,226,614,322,930,194,
757,501,375,797,983,798,779,585,8300
8215 DATA 11,115,197,176,270,801,711,9
73,302,726,404,643,831,619,628,7407
10065 DATA 811,777,768,769,680,43,279,
806,564,311,872,333,488,780,373,8654
12800 DATA 251,140,391
```

•

Moire Demo

```
10 DEG
20 A=INT(1.9*160)
30 GRAPHICS 8+16
40 SETCOLOR 2,0,0
50 FOR I=0 TO 160 STEP 5
60 B=INT(I/2)
70 COLOR 1
80 PLOT 0,B
90 DRAWTO I,160
100 PLOT A,B
110 DRAWTO A-I,160
120 PLOT 0,160-B
130 DRAWTO I,0
140 PLOT A,160-B
150 DRAWTO A-I,0
160 NEXT I
170 IF PEEK(764)<>255 THEN END
180 GOTO 170
```

CHECKSUM DATA
(See pgs. 7-10)

```
10 DATA 217,62,458,287,54,4,732,508,18
7,750,408,289,838,329,363,5486
160 DATA 746,161,728,1635
```

TRIPLE THREAT DICE

16K Cassette 24K Disk

by Michael A. Ivins

Do you like to gamble but can't afford trips to Las Vegas or Atlantic City? If so, then this program is for you. By placing your bets carefully, you can be fairly sure of a high return, while impulse betting on the high odds might make you a big winner — or it might make you go broke.

This game is modeled after a type of gambling machine found in Las Vegas casinos. These machines use three dice to play and give you several options to bet on. Unlike craps, you are betting solely on the outcome of a single roll of the dice. You may bet up to five coins (normally quarters) on each of the betting options, with no limit (other than your total cash) to how many of the options you choose to bet on.

You use your joystick to position the bet cursor next to the option you wish to bet. Pressing the trigger button will enter your bet one coin at a time until you reach five coins, after which it will not accept any more bets on that option. Moving the joystick to the left or right will move the cursor. After you have bet as many options as you wish, hold the joystick to the right until the pointer appears in the box marked "ROLL DICE." Press the trigger again, and the computer will roll the dice.

After each roll of the dice the computer will display your win or say "SORRY" if you did not win. At this time you have an additional option. If you should wish to take your winnings and quit, all you need do is pull the joystick toward you. A push on the trigger will return you to the betting routine. □

```
1 REM TRIPLE THREAT DICE
2 REM BY MICHAEL A. IVINS
3 REM JULY, 1981
10 DIM BET(31):COUNT=0
15 GRAPHICS 0:? "THIS IS A GAME PATTER
NED AFTER A":? "GAMBLING MACHINE IN LA
S VEGAS."
20 ? "YOU BET ON THE OUTCOME OF THE RO
LL OF":? "THREE DICE. YOU HAVE MANY OP
TIONS YOU"
25 ? "CAN BET ON.  TO SELECT THE OPTIO
N ON":? "WHICH YOU WISH TO BET, USE TH
E"
30 ? "JOYSTICK TO MOVE THE '>' UNTILL
IT":? "POINTS TO THE PROPER OPTION.  Y
OU"
35 ? "THEN ENTER YOUR BET BY PRESSING
THE":? "TRIGGER.  YOU MAY BET UP TO FI
VE"
40 ? "DOLLARS ON EACH OPTION."
45 ? :? "WHEN YOU HAVE FINISHED BETTIN
G, HOLD":? "THE JOYSTICK TO THE RIGHT
UNTILL A"
50 ? "POINTER APPEARS IN THE BOX MARKE
D":? "'ROLL DICE' AND PRESS TRIGGER."
60 ? :? "PRESS START TO BEGIN"
70 ? "GOOD LUCK!!!"
75 IF PEEK(53279)<>6 THEN 75
90 GOTO 1000:REM DRAW BETTING LAYOUT
100 M=100:POSITION 7,20:? M;
110 GOSUB 1200:REM CLEAR BETS RESET WI
N
120 IF COUNT=0 THEN M=100
130 IF STICK(0)=9 OR STICK(0)=10 OR ST
ICK(0)=11 THEN B=B-1:GOSUB 1500
132 IF STICK(0)=6 OR STICK(0)=7 OR STI
CK(0)=5 THEN B=B+1:GOSUB 1500
135 IF B<32 THEN IF BET(B)=5 OR M=0 TH
EN 160
140 IF B<32 AND STRIG(0)=0 THEN BET(B)
=BET(B)+1:POSITION X,Y:? BET(B);:SOUND
 0,150,10,15:M=M-1
141 IF B<32 THEN IF BET(B)=0 THEN POSI
TION X,Y:? " ";
142 IF B<32 THEN IF BET(B)>0 THEN POSI
TION X,Y:? BET(B)
145 POSITION 7,20:? M;" ";
148 FOR DELAY=1 TO 20:NEXT DELAY
149 IF B>32 THEN B=32
150 IF B=32 AND STRIG(0)=0 THEN 200
155 FOR DELAY=1 TO 20:NEXT DELAY
160 SOUND 0,0,0,0:GOTO 130
200 COUNT=COUNT+1:REM ROLL AND DRAW DI
CE
205 GOSUB 2700
210 Y=0:A=INT(RND(0)*6+1):X=10:ON A GO
SUB 10000,10010,10020,10030,10040,1005
0
220 X=14:B=INT(RND(0)*6+1):ON B GOSUB
10000,10010,10020,10030,10040,10050
230 X=18:C=INT(RND(0)*6+1):ON C GOSUB
10000,10010,10020,10030,10040,10050
240 D=A+B+C:IF COUNT>1 THEN COUNT=1
250 REM PAY WINNING BETS
260 IF BET(0)=0 OR D<12 THEN 275
265 WIN=WIN+BET(0):POSITION 6,21:? WIN
;
270 POSITION 3,14:? "+";
275 IF BET(1)=0 OR D>9 THEN 290
280 WIN=WIN+BET(1):POSITION 6,21:? WIN
;
285 POSITION 3,15:? "+";
290 IF A<>B OR B<>C OR BET(2)=0 THEN 3
05
295 WIN=WIN+(BET(2)*36):POSITION 6,21:
? WIN;
300 POSITION 3,16:? "+";
305 IF (A<>B AND A<>C AND B<>C) OR BET
(3)=0 THEN 320
310 IF A=B OR B=C OR A=C THEN WIN=WIN+
(BET(3)*6):POSITION 6,21:? WIN;
315 POSITION 3,17:? "+";
320 IF A=B AND B=C THEN GOSUB 2000
```

```
330 IF A=B THEN G=A:GOSUB 2100
335 IF A=C THEN G=A:GOSUB 2100
340 IF B=C THEN G=B:GOSUB 2100
350 IF D<11 THEN GOSUB ((D-3)*10)+2200
352 IF D>10 THEN GOSUB (ABS(D-18)*10)+
2200
360 IF D=17 THEN GOSUB 2210:IF D=18 TH
EN GOSUB 2200
370 M=M+WIN:POSITION 7,20:? M;
372 IF M=0 THEN 420
373 IF M>=5000 THEN 5000
375 IF WIN>0 THEN 400
380 GOSUB 2600
382 POSITION 1,0:? "SORRY";
385 IF STRIG(0)=0 THEN POSITION 1,0:?
"      ";:GOTO 110
390 IF STICK(0)=13 THEN 500
395 GOTO 382
400 GOSUB 2500
402 POSITION 1,0:? "WINNER";
405 IF STRIG(0)=0 THEN POSITION 1,0:?
"      ";:GOTO 110
410 IF STICK(0)=13 THEN 500
415 GOTO 402
420 ? "%I'M SORRY, BUT YOU HAVE GONE B
ROKE";:? "IF YOU WISH TO START AGAIN WI
TH A"
430 ? "NEW BANKROLL PRESS START, TO QU
IT":? "PRESS SELECT"
440 IF PEEK(53279)<>6 AND PEEK(53279)<
>5 THEN 440
450 IF PEEK(53279)=6 THEN COUNT=0:GOTO
 90
460 IF PEEK(53279)=5 THEN ? "GOODBYE A
ND BETTER LUCK NEXT TIME":END
500 ? "%IT IS A WISE GAMBLER WHO KKNOW
S WHEN    TO QUIT."
510 ? :? "THANK YOU FOR PLAYING AND GO
OD LUCK    TO YOU THE NEXT TIME."
520 ? :? "GOODBYE.":END
1000 GRAPHICS 0:POKE 752,1:POKE 82,1:S
ETCOLOR 2,12,12:? "
       USE JOYSTICK";
1002 SETCOLOR 1,12,0:SETCOLOR 4,12,12
1005 ? "                                TO
MOVE BET"
1010 ? "                                POI
NTER"
1020 ? :? "PAYS 216-1  PAYS 18-1   TOT
AL   PAYS"
1025 ? "┌──────────┬──────────┬──────
──────┐"
1030 ? "|  3-ONES  ||  2-ONES  ||  3
  216-1|"
1035 ? "|  3-TWOS  ||  2-TWOS  ||  4
   72-1|"
1040 ? "|  3-THREES||  2-THREES||  5
   36-1|"
1045 ? "|  3-FOURS ||  2-FOURS ||  6
   21-1|"
1050 ? "|  3-FIVES ||  2-FIVES ||  7
   14-1|"
1055 ? "|  3-SIXES ||  2-SIXES ||  8
   10-1|"
1060 ? "└──────────┘└──────────┘|  9
    9-1|"
1065 ? "┌──────────────────────┐|  10
    8-1|"
1070 ? "|  HI (OVER 11)      1-1||  11
    8-1|"
1075 ? "|  LO (UNDER 10)     1-1||  12
    9-1|"
1080 ? "|  ANY 3 OF KIND    36-1||  13
   10-1|"
1085 ? "|  ANY 2 OF KIND     6-1||  14
   14-1|"
1090 ? "└──────────────────────┘|  15
   21-1|"
1095 ? "┌──────────────┐┌─────┐|  16
   36-1|"
1100 ? "|CASH:          || ROLL||  17
   72-1|"
1105 ? "|WIN:           || DICE||  18
  216-1|"
1110 ? "└──────────────┘└─────┘└─────
─────┘"
1120 GOTO 110
1200 POKE 752,1:FOR I=0 TO 31:BET(I)=0
:NEXT I
1210 FOR I=6 TO 11:POSITION 2,I:? "  "
;:POSITION 14,I:? "  ";:NEXT I
1220 FOR I=14 TO 17:POSITION 2,I:? "
";:NEXT I
1230 FOR I=6 TO 21:POSITION 26,I:? "
";:NEXT I
1240 POSITION 19,20:? " ↓€ ";
1270 WIN=0:POSITION 6,21:? "        "
1280 B=0:GOSUB 1500
1290 RETURN
1500 IF B<0 THEN B=0:IF B>32 THEN B=32
1510 IF B=0 THEN POSITION 3,14:? ")↓€
";:X=2:Y=14
1511 IF B=1 THEN POSITION 3,14:? " ↓€)
↓€ ";:X=2:Y=15
1512 IF B=2 THEN POSITION 3,15:? " ↓€)
↓€ ";:X=2:Y=16
1513 IF B=3 THEN POSITION 3,16:? " ↓€)
";:POSITION 3,6:? " ";:X=2:Y=17
1514 IF B=4 THEN POSITION 3,17:? " ";:
POSITION 3,6:? ")↓€ ";:X=2:Y=6
1515 IF B>4 AND B<9 THEN POSITION 3,B+
1:? " ↓€)↓€ ";:X=2:Y=B+2
1516 IF B=9 THEN POSITION 3,10:? " ↓€)
";:POSITION 15,6:? " ";:X=2:Y=11
1517 IF B=10 THEN POSITION 3,11:? " ";
:POSITION 15,6:? ")↓€ ";:X=14:Y=6
1518 IF B>10 AND B<15 THEN POSITION 15
,B-5:? " ↓€)↓€ ";:X=14:Y=B-4
1519 IF B=15 THEN POSITION 15,10:? " ↓
€)";:POSITION 27,6:? " ";:X=14:Y=11
1520 IF B=16 THEN POSITION 15,11:? " "
;:POSITION 27,6:? ")↓€ ";:X=26:Y=6
1521 IF B>16 AND B<31 THEN POSITION 27
,B-11:? " ↓€)↓€ ";:X=26:Y=B-10
1522 IF B=31 THEN POSITION 27,20:? " ↓
€)";:POSITION 19,20:? " ↓€ ";:X=26:Y=2
1
1523 IF B=32 THEN POSITION 27,21:? " "
;:POSITION 19,20:? "▙↓€▛";
1550 RETURN
2000 IF BET(A+3)=0 THEN RETURN
2010 WIN=WIN+(BET(A+3)*216):POSITION 6
,21:? WIN;
2020 POSITION 3,A+5:? "+";
2030 RETURN
2100 IF BET(G+9)=0 THEN RETURN
2110 WIN=WIN+BET(G+9)*18:POSITION 6,21
:? WIN;
2120 POSITION 15,G+5:? "+";
2130 RETURN
2200 IF BET(13+D)=0 THEN RETURN
2202 WIN=WIN+BET(13+D)*216:POSITION 6,
21:? WIN;
2204 POSITION 27,D+3:? "+";
2206 RETURN
2210 IF BET(13+D)=0 THEN RETURN
2212 WIN=WIN+BET(13+D)*72:POSITION 6,2
1:? WIN;
2214 POSITION 27,D+3:? "+";
2216 RETURN
2220 IF BET(13+D)=0 THEN RETURN
2222 WIN=WIN+BET(13+D)*36:POSITION 6,2
1:? WIN;
2224 POSITION 27,D+3:? "+";
2226 RETURN
2230 IF BET(13+D)=0 THEN RETURN
2232 WIN=WIN+BET(13+D)*21:POSITION 6,2
1:? WIN;
2234 POSITION 27,D+3:? "+";
2236 RETURN
2240 IF BET(13+D)=0 THEN RETURN
2242 WIN=WIN+BET(13+D)*14:POSITION 6,2
1:? WIN;
2244 POSITION 27,D+3:? "+";
2246 RETURN
2250 IF BET(13+D)=0 THEN RETURN
2252 WIN=WIN+BET(13+D)*10:POSITION 6,2
1:? WIN;
2254 POSITION 27,D+3:? "+";
2256 RETURN
2260 IF BET(13+D)=0 THEN RETURN
```

```
2262 WIN=WIN+BET(13+D)*9:POSITION 6,21
:? WIN;
2264 POSITION 27,D+3:? "+";
2266 RETURN
2270 IF BET(13+D)=0 THEN RETURN
2272 WIN=WIN+BET(13+D)*8:POSITION 6,21
:? WIN;
2274 POSITION 27,D+3:? "+";
2276 RETURN
2500 FOR I=1 TO 10
2505 FOR S=40 TO 90 STEP 5
2510 SOUND 0,S,10,10
2530 NEXT S
2540 FOR S=90 TO 40 STEP -5
2550 SOUND 0,S,10,10
2570 NEXT S
2580 NEXT I
2590 SOUND 0,0,0,0:RETURN
2600 SOUND 0,200,10,10
2620 FOR DELAY=1 TO 100:NEXT DELAY
2630 SOUND 0,241,10,10
2640 FOR DELAY=1 TO 150:NEXT DELAY
2650 SOUND 0,0,0,0:RETURN
2700 FOR I=1 TO 20
2710 FOR S=0 TO 50 STEP 20
2720 SOUND 0,S,8,15
2730 NEXT S:SOUND 0,0,0,0
2740 NEXT I
2750 POKE 77,0:RETURN
5000 ? CHR$(125);"THIS MACHINE HAS NO
MORE MONEY."
5010 ? :? "IF YOU WISH TO CASH IN YOUR
 BANKROLL":? "AND PLAY AGAIN AFTER THE
 MANAGE-"
5020 ? "MENT HAS REFILLED IT, THEN PRE
SS":? "START.  TO QUIT PRESS SELECT."
5030 IF PEEK(53279)<>6 AND PEEK(53279)
<>5 THEN 5030
5040 IF PEEK(53279)=5 THEN ? "THANK YO
U FOR PLAYING":? "GOODBYE!":END
5050 COUNT=0:GOTO 1000
10000 POSITION X,Y:? "■■ ";
10001 POSITION X,Y+1:? " ■ ";
10002 POSITION X,Y+2:? " ■■";
10003 RETURN
10010 POSITION X,Y:? "■■ ";
10011 POSITION X,Y+1:? " ■ ";
10012 POSITION X,Y+2:? "■■ ";
10013 RETURN
10020 POSITION X,Y:? " ■■";
10021 POSITION X,Y+1:? " ■ ";
10022 POSITION X,Y+2:? "■■ ";
10023 RETURN
10030 POSITION X,Y:? "■ ■";
10031 POSITION X,Y+1:? " ■ ";
10032 POSITION X,Y+2:? "■ ■";
10033 RETURN
10040 POSITION X,Y:? "■ ■";
10041 POSITION X,Y+1:? " ■ ";
10042 POSITION X,Y+2:? " ■■";
10043 RETURN
10050 POSITION X,Y:? "■ ■";
10051 POSITION X,Y+1:? "■ ■";
10052 POSITION X,Y+2:? " ■■";
10053 RETURN
```

•

CHECKSUM DATA

(See pgs. 7-10)

```
1 DATA 503,632,390,141,268,152,941,724
,364,964,357,926,652,485,701,8200
90 DATA 7,758,336,439,680,570,166,814,
491,127,146,339,678,261,339,6151
160 DATA 157,784,804,588,315,334,580,4
56,813,248,964,920,252,975,45,8235
295 DATA 690,951,120,800,962,547,800,8
07,810,363,993,310,143,562,985,9843
375 DATA 859,824,605,727,478,749,801,9
11,496,456,705,583,643,981,307,10125
460 DATA 704,326,95,158,803,846,105,96
5,524,223,490,533,705,549,460,7486
1055 DATA 549,35,101,873,27,87,67,111,
102,404,356,257,868,548,199,4584
1220 DATA 682,780,226,362,614,794,459,
991,183,191,167,944,79,466,436,7374
1518 DATA 720,852,599,718,893,935,799,
182,53,69,784,203,782,193,787,8569
2200 DATA 905,944,198,799,906,30,199,8
00,907,35,200,801,908,25,201,7858
2236 DATA 802,909,31,202,803,910,24,20
3,804,911,751,204,805,912,751,9022
2274 DATA 205,806,347,111,418,534,289,
453,538,509,257,757,623,766,635,7248
2650 DATA 256,356,281,606,436,511,94,1
08,973,526,942,891,148,170,208,6506
10002 DATA 250,49,136,246,240,51,150,2
12,230,53,902,250,220,55,118,3162
10041 DATA 216,222,57,120,218,224,59,1
116
```

•

BICYCLE

16K Cassette 24K Disk

by Dan Devos

Bicycle is a one player game. You are a messenger working for the largest shipping company in the world. As part of your daily routine, you must run memos and invoices from the main shipping offices out to the loading and receiving docks. Leaping on your trusty bicycle, you proceed across the vast parking lot, past rows of idling tractor trailer rigs, dodging the many potholes that impede your progress. However, the potholes are not the only things you have to look out for. The drivers of the trucks are in a hurry to leave, and often they can't bother to watch out for one poor little messenger on a bicycle! Needless to say, you have to be careful where you're going!

Playing the game.

The cyclist is continually proceeding at a fixed rate, and he can also move up and down. Every time you are hit by a truck or fall into a pothole, you lose a cyclist. There is a total of three cyclists in a game.

Scoring.

For every space you move, you get one point. For every section of the parking lot, there are two truck drivers walking to their trucks. If you hit a walking truck driver you get 500 points. Watch out! The truck drivers can stand over the pot holes and when the cyclist hits them he falls into the hole.

The program.

This program uses a machine language subroutine to move player missile graphics. The program draws two rows of trucks in Graphics Mode 1. Then three players, exactly the same as the edited characters, are put on top of three of the trucks. These trucks are erased and the player trucks can then move smoothly. The rest of the program just moves the players. Type in the program and wait until the screen display says "Press Start." □

```
0 GOTO 30060
1 BB=C0:T=C0:U=C0:V=C0:DIM CHAR$(C8),W
HICH(C3,C2):CHAR$="FQUVWXZ"
2 GRAPHICS C17:CHSET=(PEEK(106)-C32)*C
256:CHORG=57344:POKE 623,C1
3 FOR I=C0 TO 511:POKE CHSET+I,PEEK(CH
ORG+I):NEXT I
4 FOR I=C1 TO C7
5 CHPOS=CHSET+(ASC(CHAR$(I))-C32)*C8
6 FOR J=C0 TO C7
7 READ A:POKE CHPOS+J,A
8 NEXT J:NEXT I
9 FOR I=C32 TO 39:POKE CHSET+I,C256-C1
-PEEK(CHORG+I):NEXT I
10 POKE 756,CHSET/C256
15 DATA 0,0,223,149,213,85,223,0
16 DATA 0,16,120,254,127,30,4,0
17 DATA 8,8,28,20,54,0,0,0
18 DATA 0,0,28,28,93,42,28,8
19 DATA 112,112,248,248,252,252,226,22
6
20 DATA 226,226,254,254,255,127,181,24
5
21 DATA 223,95,27,0,0,0,0,0
31 BB=C0:Y=C0:SETCOLOR C2,C3,C4:SETCOL
OR C3,C0,12:SETCOLOR C1,C8,C8:SETCOLOR
 C0,C8,C4
32 BB=C0:W=C3:POSITION C5,19:? #C6;"ME
N:VVV":POSITION C9,C20:? #C6;"UUU"
35 T=C0:U=C0:SETCOLOR C2,C3,C4:SETCOLO
R C3,C0,12
40 POSITION C2,C5:? #C6;"W W W W W W W
 W W   X X X X X X X X X   Z Z Z Z Z Z
 Z Z Z"
50 POSITION C2,12:? #C6;"W W W W W W W
 W W   X X X X X X X X X   Z Z Z Z Z Z
 Z Z Z"
54 SETCOLOR C1,C8,C8:J=INT(C17*RND(C0)
+C2):K=INT(C4*RND(C0)+C8)
55 L=INT(C17*RND(C0)+C2):M=INT(C4*RND(
C0)+C8):POSITION J,K:? #C6;"q"
56 POSITION L,M:? #C6;"q"
57 SETCOLOR C0,C8,C4:N=INT(C17*RND(C0)
+C2):O=INT(C3*RND(C0)+C8)
60 P=INT(C17*RND(C0)+C2):Q=INT(C3*RND(
C0)+C8):POSITION N,O:? #C6;"v"
65 POSITION N,O+C1:? #C6;"U":POSITION
P,Q:? #C6;"v":POSITION P,Q+C1:? #C6;"U
"
70 IF AA THEN 4000
100 AA=C1:POKE 752,C1:POKE 53257,C0:PO
KE 53258,C0:POKE 53259,C0
110 PCOL0=26:PCOL1=52:PCOL2=52:PCOL3=5
2
1000 FOR I=1536 TO 1706:READ A:POKE I,
A:NEXT I
1010 FOR I=1774 TO 1787:POKE I,C0:NEXT
 I
1020 PM=PEEK(106)-C16:PMBASE=C256*PM
1030 FOR I=PMBASE+1023 TO PMBASE+2046:
POKE I,C0:NEXT I
1040 FOR I=PMBASE+1025 TO PMBASE+1034:
READ A:POKE I,A:NEXT I
1050 FOR I=PMBASE+1281 TO PMBASE+1299:
READ A:POKE I,A:NEXT I
1060 FOR I=PMBASE+1537 TO PMBASE+1555:
READ A:POKE I,A:NEXT I
1061 FOR I=PMBASE+1793 TO PMBASE+1811:
READ A:POKE I,A:NEXT I
1070 POKE 704,PCOL0:POKE 705,PCOL1:POK
E 706,PCOL2:POKE 707,PCOL3
```

```
1080 PLX=53248:PLY=1780:PLL=1784
1090 POKE 559,62:POKE 1788,PM+C4:POKE
53277,C3:POKE 54279,PM
1100 X=USR(1696)
2000 DATA 162,3,189,244,6,240,89,56,22
1,240,6,240,83,141,254,6,106,141
2010 DATA 255,6,142,253,6,24,169,0,109
,253,6,24,109,252,6,133,204,133
2020 DATA 206,189,240,6,133,203,173,25
4,6,133,205,189,248,6,170,232,46,255
2030 DATA 6,144,16,168,177,203,145,205
,169,0,145,203,136,202,208,244,76,87
2040 DATA 6,160,0,177,203,145,205,169,
0,145,203,200,202,208,244,174,253,6
2050 DATA 173,254,6,157,240,6,189,236,
6,240,48,133,203,24,138,141,253,6
2060 DATA 109,235,6,133,204,24,173,253
,6,109,252,6,133,206,189,240,6,133
2070 DATA 205,189,248,6,170,160,0,177,
203,145,205,200,202,208,248,174,253,6
2080 DATA 169,0,157,236,6,202,48,3,76,
2,6,76,98,228,0,0,104,169
2090 DATA 7,162,6,160,0,32,92,228,96
3000 DATA 48,48,32,56,36,56,110,181,16
5,66
3010 DATA 112,112,248,248,252,252,226,
226,226,226,254,254,255,127,181,245,22
3,95,27
3020 DATA 112,112,248,248,252,252,226,
226,226,226,254,254,255,127,181,245,22
3,95,27
3030 DATA 112,112,248,248,252,252,226,
226,226,226,254,254,255,127,181,245,22
3,95,27
4000 POKE PLL,C10:POKE PLL+C1,C20:POKE
 PLL+C2,C20:POKE PLL+C3,C20:A=C0:B=80
4010 POKE PLX,A+C48:POKE PLY,B+C32
4020 G=INT(C4*RND(C0))+C1:D=G*C16:E=95
:POKE PLX+C1,D+C48:POKE PLY+C1,E+C32
4030 G=G+G:POSITION G,12:? #C6;" ":POS
ITION G,13:? #C6;" ":POSITION G,14:? #
C6;" "
4040 I=INT(C5*RND(C0))+C5:F=I*C16:G=95
:POKE PLX+C2,F+C48:POKE PLY+C2,G+C32
4050 I=I+I:POSITION I,12:? #C6;" ":POS
ITION I,13:? #C6;" ":POSITION I,14:? #
C6;" "
4060 R=INT(C9*RND(C0))+C1:H=R*C16:I=39
:POKE PLX+C3,H+C48:POKE PLY+C3,I+C32
4070 R=R+R:POSITION R,C5:? #C6;" ":POS
ITION R,C6:? #C6;" ":POSITION R,C7:? #
C6;" "
4071 POKE 53278,C0:IF BB THEN 4080
4072 BB=C1:SOUND C0,200,C10,C8:SOUND C
1,201,C10,C8
4073 POSITION C5,18:? #C6;"press start
"
4074 R=R+C1:SETCOLOR C1,R,C8:IF PEEK(5
3279)<>C6 THEN 4073
4075 POSITION C5,18:? #C6;"
":SETCOLOR C1,C8,C8:SOUND C0,C0,C0,C0:
SOUND C1,C0,C0,C0
4080 POSITION C5,C17:? #C6;"SCORE:";Y,
4081 IF INT((A/C8)+C1)*C8+C4)J*C8+C8 O
R INT((A/C8)+C1)*C8+C4<J*C8 THEN 4120
4090 IF INT((B/C8)+C1)*C8+C4)K*C8 AND
INT((B/C8)+C1)*C8+C4<K*C8+C8 THEN 3000
0
4120 IF INT((A/C8)+C1)*C8+C4)L*C8+C8 O
R INT((A/C8)+C1)*C8+C4<L*C8 THEN 4140
4130 IF INT((B/C8)+C1)*C8+C4)M*C8 AND
INT((B/C8)+C1)*C8+C4<M*C8+C8 THEN 3000
0
4140 IF T THEN 4190
4150 IF INT((A/C8)+C1)*C8+C4)N*C8+C8 O
R INT((A/C8)+C1)*C8+C4<N*C8 THEN 4190
4160 IF INT((B/C8)+C1)*C8+C4)O*C8 AND
INT((B/C8)+C1)*C8+C4<O*C8+C16 THEN 100
00
4190 IF U THEN 4220
4200 IF INT((A/C8)+C1)*C8+C4)P*C8+C8 O
R INT((A/C8)+C1)*C8+C4<P*C8 THEN 4220
4210 IF INT((B/C8)+C1)*C8+C4)Q*C8 AND
INT((B/C8)+C1)*C8+C4<Q*C8+C16 THEN 100
10
4220 IF INT((B/C8)+C1)*C8+C8<63 OR INT
((B/C8)+C1)*C8+C8)96 THEN 30000
4230 IF  NOT STRIG(C0) THEN 9000
4300 E=E-C1:POKE PLX+C1,D+C48:POKE PLY
+C1,E+C32:G=G-C1:POKE PLX+C2,F+C48:POK
E PLY+C2,G+C32
4310 I=I+C1:POKE PLX+C3,H+C48:POKE PLY
+C3,I+C32
4600 IF PEEK(53260) THEN 30000
5000 A=A+C5
5001 IF STICK(C0)=13 THEN B=B+C3
5002 IF STICK(C0)=C7 THEN A=A+C1
5003 IF STICK(C0)=14 THEN B=B-C3
5004 IF A)165 THEN 20000
5005 Y=Y+C1:SOUND C1,50,C10,C8:SOUND C
1,C0,C0,C0:POKE PLX,A+C48:POKE PLY,B+C
32:GOTO 4080
9000 FOR ZZ=C1 TO 30:NEXT ZZ
9001 IF  NOT STRIG(C0) THEN 4300
9002 GOTO 9001
10000 T=C1:POSITION N,O:? #C6;"F":POSI
TION N,O+C1:? #C6;" ":GOTO 10060
10010 U=C1:POSITION P,Q:? #C6;"F":POSI
TION P,Q+C1:? #C6;" ":GOTO 10060
10060 V=V+C1:Y=Y+500:FOR KK=C256 TO C1
 STEP -C1:SOUND C0,KK,C10,14:NEXT KK:S
OUND C0,C0,C0,C0:GOTO 4080
20000 POSITION N,O:? #C6;" ":POSITION
N,O+C1:? #C6;" ":POSITION P,Q:? #C6;"
":POSITION P,Q+C1:? #C6;" "
20010 POSITION J,K:? #C6;" ":POSITION
L,M:? #C6;" "
20020 GOTO 35
30000 W=W-C1:POKE PLX,A+43:POKE PLY,B+
C32:FOR XX=14 TO C0 STEP -C1
30010 SOUND C0,250,C10,XX:FOR YY=C1 TO
 C2:NEXT YY:NEXT XX
30020 POSITION N,O:? #C6;" ":POSITION
N,O+C1:? #C6;" ":POSITION P,Q:? #C6;"
":POSITION P,Q+C1:? #C6;" "
30030 POSITION J,K:? #C6;" ":POSITION
L,M:? #C6;" "
30031 POSITION C5,19:? #C6;"MEN:";FOR
Z=C0 TO W:POSITION C9+W,19:? #C6;" ":P
OSITION C9+W,C20:? #C6;" ":NEXT Z
30040 IF  NOT W THEN 31
30050 GOTO 35
30060 C1=1:C2=2:C3=3:C4=4:C5=5:C6=6:C7
=7:C8=8:C9=9:C10=10:C16=16:C17=17:C20=
20:C32=32:C48=48:C256=256
30070 GRAPHICS C1:SETCOLOR C2,C0,C0:PO
SITION C5,C8:? #C6;"* BICYCLE *":POSIT
ION C9,C10:? #C6;"BY"
30080 POSITION C6,12:? #C6;"DAN DEVOS"
30090 FOR T=C1 TO 200:SETCOLOR C0,T,C8
:POSITION C5,C8:? #C6;"* BICYCLE *":NE
XT T:GOTO 1
```

•

CHECKSUM DATA

(See pgs. 7-10)

```
0 DATA 633,344,996,665,276,726,280,49,
214,585,127,436,580,861,385,7157
19 DATA 280,270,31,833,464,387,401,358
,16,468,564,29,504,789,386,5780
100 DATA 665,304,46,296,542,845,568,59
5,609,612,172,12,272,246,433,6217
2010 DATA 175,662,640,727,304,533,798,
647,2,556,537,538,539,378,590,7626
4020 DATA 957,920,999,940,59,194,351,8
84,906,101,959,678,731,821,740,10240
4130 DATA 822,549,766,915,551,753,920,
839,864,417,868,469,363,814,854,10764
5003 DATA 823,772,423,19,855,743,711,7
33,948,757,303,907,539,631,763,9927
30030 DATA 309,802,886,915,636,321,770
,187,4826
```

•

COLOR SLOT MACHINE

24K Cassette or Disk

by Michael A. Ivins

The re-defined character set is a powerful tool which can be used in many different ways. The characters can be used for special animation effects, and are especially useful when combined with certain types of modified display lists. Finally, they can be used to create colorful graphic displays in the text mode, GR.0. This last application is the subject of this article.

If you have ever done much playing around with GR.8 you know that getting four colors in this mode is not as difficult as you might expect. For the newcomers I include example **Program 1** to show what I mean.

```
10 GRAPHICS 8:SETCOLOR 2,0,15
20 SETCOLOR 1,0,0:COLOR 1
30 FOR X=0 TO 200 STEP 2:PLOT X,0
40 DRAWTO X,10:NEXT X
50 FOR X=1 TO 201 STEP 2:PLOT X,20
60 DRAWTO X,30:NEXT X
70 FOR X=0 TO 200:PLOT X,40
80 DRAWTO X,50:NEXT X
```

•

Program 1.

This may seem to have little to do with re-defined character sets, but bear with me, I'm coming to it. The example should show what appear to be three bars on a white background with blue at the top, red next and black at the bottom. The program was supposed to draw two sets of evenly spaced vertical lines and one solid bar, so what happened? You would expect the bottom bar to be black. The only differences between the top two bars is in the positioning of the vertical lines, yet we get the two colors.

This effect is due to a curious property of the graphics screen whose technical name is "artifacting." Simply stated, the principle is that a single pixel of GR.8 (the smallest the ATARI will generate) will be one color while another pixel one space or any odd number of spaces away will have a different color. By now you are probably asking, "If this guy wants to talk about re-defined character sets, why all this stuff about colors in GR.8?" Every character has eight bytes associated with it, and the pattern made up by those bits which are ones determines the shape of the character. Two examples of this are shown in **Figure 1** with 1A showing the bit pattern of the letter "A" and 1B showing a percent sign. Each pixel has a GR.0 character, whether it be text or control graphic, is identical to a single pixel of GR.8. By applying the same techniques which gave us colors in GR.8 to re-defining characters, we can get many kinds of colored graphic characters.

A	B
00000000	00000000
00011000	01100110
00111100	01101100
01100110	00011000
001100110	00110000
01111110	01100110
01100110	01000110
00000000	00000000

Figure 1.

There is one important factor which should be mentioned at this point. The colors you can get from your special characters (or a GR.8 display) will

depend on the chosen background color and chosen luminosity of the foreground. For your own applications you should experiment with the combinations of foreground and background color which gives the effect you want most. For the purpose of this article and the game program which accompanies it I use a white background (SETCOLOR 2,0,15) and a black foreground (SETCOLOR 1,0,0).

I give two examples in **Figures 2** and **3**. For greater ease of use I have enlarged the pattern of bits so you can see them better than in the previous example. I have also labeled the values of the bits and given the decimal values that you would poke into the character table to make the changes. With the specified colors, the character defined in **Figure 2** will give you a solid blue block while the one in **Figure 3** will make a solid red block.

128	64	32	16	8	4	2	1	Decimal value
1	0	1	0	1	0	1	0	170
1	0	1	0	1	0	1	0	170
1	0	1	0	1	0	1	0	170
1	0	1	0	1	0	1	0	170
1	0	1	0	1	0	1	0	170
1	0	1	0	1	0	1	0	170
1	0	1	0	1	0	1	0	170
1	0	1	0	1	0	1	0	170
1	0	1	0	1	0	1	0	170

Figure 2 (Blue Block).

128	64	32	16	8	4	2	1	Decimal value
0	1	0	1	0	1	0	1	85
0	1	0	1	0	1	0	1	85
0	1	0	1	0	1	0	1	85
0	1	0	1	0	1	0	1	85
0	1	0	1	0	1	0	1	85
0	1	0	1	0	1	0	1	85
0	1	0	1	0	1	0	1	85
0	1	0	1	0	1	0	1	85

Figure 3 (Red Block).

Simple red and blue blocks alone make for rather dull graphics, but I'm sure you can see that by clever arranging of the dots you can create many interesting shapes. If the shape you want is too large to fit into a single 8x8 grid then use two, three or even more characters. To give one example of the kinds of things that can be done with color graphics characters and hopefully have a little fun at the same time, I include my program for **Color Slot Machine.**

Before getting into a description of the game itself there is a comment I would like to make. Calculating out all the numbers for special characters you have drawn on graph paper is very slow work and it tends to be boring. Fortunately this is the sort of task which lends itself to being "computerized." There are, in fact, many character editor programs on the market as well as some which have been published in magazines. These all allow you to make changes in an enlarged matrix and see the effects of these changes on the normal sized character. They let you save the special character set or "font" for use with your own programs. I used such a program which went as far as writing the actual subroutine that does the work in the **Slot Machine** game.

The game.

After the title display, the program will draw a colorful slot machine on the screen making use of several kinds of colored graphics characters. There are two ways to play, which you choose by pressing the OPTION button any time there is no bet placed. For those who might be unfamiliar with slot machines I will describe the options. Single line play uses only those symbols which line up in the center of the pay windows. In this version additional "coins" bet give bigger payouts when a winning combination comes up. The five line version gives more ways to win by adding top, bottom and diagonal paylines for the number of coins played. Single line play can pay more when it pays, but the five line version can give more ways to win so you win more often.

Playing the game itself is simple. To enter a single coin bet, press the trigger button of the joystick and release. If you wish to bet the maximum bet of five, simply hold the trigger button down until the beeps stop. In the single line version the paychart changes to reflect payout for the size of the bet, while the five line version employs line pointers to indicate how many lines are in play. When you have made your desired bet, move the joystick in any direction to spin the reels. More details are given in the program documentation. Happy gambling! □

PROGRAM DOCUMENTATION

The first thing the program does is to jump to the routine which alters the character set and since that is the main thing I wish to illustrate, I will cover this first.

Line 32000 The first step resets RAMTOP. Next a graphics command to set the new top of memory. Now we poke the location of the new character set.

Line 32005 This defines a machine language routine which will copy the old character set out of ROM into the protected area of RAM.

Lines 32010-32015 These lines are here to give you something to look at while the character set initializes. You won't see anything at

this point since the area pointed to by CHBAS (location 764) is blank.

Line 32020 This executes the machine language routine so that the material printed in the previous lines can now be seen on the screen.

Lines 32030-32040 Now we make the actual changes. We first read the number that tells where to start and then put in the new numbers. Some of the characters look a bit funny (like a cluster of cherries with a blue leaf or a purple bell) but this is the best I could do with these colors.

Now we return to the main program and from this point I will take things in the sequence they are shown in the program.

Lines 10-11 These set up the reels of 30 "symbols" on each. If you wished to change the odds of the game, this is the place to do it. You could make it harder to win by changing the symbols or by adding no pay symbols or blanks. If you wanted to, you could set up the reels so that you would win on every play, which I would consider to be boring

Line 30 Jumps to the routine which draws the machine.

Lines 40-100 Here we set the initial values for game counters and display them. This also lets you know you are playing the one line version.

Line 120 This displays the betting prompt.

Line 125 If the bet is the maximum or the bankroll is zero then the betting routine is skipped.

Line 130 Wait for trigger press, increment bet, decrement bankroll and start sound. Also gosub to change the paychart.

Line 135 Jump to the five line version if OPTION is pressed and bet is zero.

Line 145 Erase play prompt if bet reaches maximum.

Line 150 A delay is slow betting.

Lines 160-168 Display bet, shut off sound, display bet.

Lines 170-175 Return to betting loop if stick not moved or if stick moved but bet zero.

Line 180 Zero out the attract mode.

Lines 290-310 Jumps to the routine that animate the handle and spin the reels.

Lines 311-315 Reads the symbols on the payline and jumps to payroutine.

Lines 320-327 Calculates the proper length of windsound and jumps to that routine.

Lines 330-340 Resets bet to zero and if any money is left you are returned to the betting routine.

Lines 350-420 This is the routine that is activated if you go broke. It resets the left margin, erases the paychart and then gives you your quit or start over options. Starting over redraws the machine and paychart and resets all values to beginning levels. Quitting naturally ends the game.

Lines 500-590 The functions here are similar to the betting loop of the single line version. The main difference is in setting line pointers instead of changing the paychart.

Lines 600-610 This reads all payable locations. Caution should be noted here. Be sure when you type these lines in that you use the abbreviation LOC, for LOCATE and POS, for POSITION, or you won't get everything in on those lines.

Lines 620-676 This section checks for winning combinations and jumps to the payout routine if one is present. I originally tried to make this section more brief, but kept getting errors.

Lines 680-698 Checks for win and goes to sound routine if appropriate.

Lines 700-720 Resets bet, erases line pointers and jumps back to betting routine if not bankrupt.

Lines 1000-1055 This creates the siren for winning and is a simple tone with rising and falling pitch.

Lines 1300-1360 This is the routine which resets the paychart according to the size of the bet in the one line play version.

Lines 2000-2090 This is the routine which actually draws the machine. Notation should be made here that the paychart is not truly complete. Most combinations will pay with a bar (single or double) on the last reel. I did not have room to fit this on the screen.

Lines 2300-2390 This animates the handle by first erasing the knob and redrawing it lower and doing the reverse by drawing a section of the handle and the knob one space higher. This routine also clears any old wins.

Lines 2400-2495 This is the payout routine for the single line version.

Lines 2600-2690 This is the super jackpot routine and is triggered if three of the "seven" symbols appear on the center line with maximum bet in the one line version and on the fifth line with maximum bet in the five line version. This has first an explosion sound, a slower siren than the regular windsound followed by another explosion. Then the words "SUPER JACKPOT!!!" are flashed. I suggest that you type in "GOSUB 2600" from the direct mode as you won't be seeing much of this routine unless you change the odds.

Lines 2950-3180 This animates the spin of the reels. I had originally tried to make the reels turn two full spins plus a random bit extra, but this slowed down the action of the game too much, doing it from BASIC. Therefore we just

assume those spins without showing them and take a certain number plus a random amount. From this point the reels are moved visibly by a fixed number of spaces for each reel.

Lines 4000-4080 This is the pay routine for the five line version. □

```
1 REM COLOR SLOT MACHINE
2 REM BY MICHAEL A. IVINS
3 REM NOVEMBER 1981
4 GOSUB 32000
10 DIM L$(60),M$(60),R$(60),PAY$(9):OP
EN #2,12,0,"S:":WINSOUND=1000:FPAY=400
0:SPIN=2950
11 PAY$=" h▙_-●●2H":CH=97:DB=146:SB=16
0
15 L$="abcdefgh■abghefcd═ghcd■efijc
dghcdgh═cdefcdcdabghefcdcdefgh"
20 M$="ab■efghefabghefcd═efcdabefghi
jefabcdgh═efabef■ghabcdefcd"
25 R$="ghcdef═ghefghefcd═cdcdef■cdi
jgh■cdgh═cdcdefghef═efefcd"
30 BANKROLL=100:BET=0:WIN=0:L=1:M=1:R=
1
40 GOSUB 2000:REM DRAW MACHINE
90 POSITION 20,20:? "1 LINE PLAY"
100 POSITION 20,22:PRINT "BANKROLL:";B
ANKROLL:POSITION 20,23:PRINT "BET:";BE
T;:POSITION 30,23:PRINT "WIN:";WIN;
120 IF BET<5 THEN POSITION 20,21:PRINT
 "PLAY 1 TO 5 COINS";
125 IF BANKROLL=0 OR BET=5 THEN 145
130 IF STRIG(0)=0 THEN BET=BET+1:GOSUB
 1300:SOUND 0,50,10,14:BANKROLL=BANKRO
LL-1
135 IF PEEK(53279)=3 AND BET=0 THEN 50
0
145 IF BET=5 THEN POSITION 20,21:? "
                ";
150 FOR DELAY=1 TO 5:NEXT DELAY
160 POSITION 24,23:PRINT BET;
165 SOUND 0,0,0,0
168 POSITION 29,22:PRINT BANKROLL;" ";
170 IF STICK(0)=15 THEN 120
175 IF BET=0 THEN 120
180 POKE 77,0
290 GOSUB 2300:REM PULL HANDLE
310 GOSUB SPIN
311 LOCATE 5,8,LM:POSITION 5,8:? CHR$(
LM)
312 LOCATE 8,8,MM:POSITION 8,8:? CHR$(
MM)
313 LOCATE 11,8,RM:POSITION 11,8:? CHR
$(RM)
315 GOSUB 2400
320 IF WIN>0 AND WIN<BET*10 THEN DUR=2
:GOSUB WINSOUND
325 IF WIN>=BET*10 AND WIN<BET*25 THEN
 DUR=3:GOSUB WINSOUND
326 IF WIN>=BET*25 AND WIN<=BET*50 THE
N DUR=5:GOSUB WINSOUND
327 IF WIN>BET*50 AND WIN<2000 THEN DU
R=10:GOSUB WINSOUND
330 BET=0:POSITION 24,23:PRINT BET;" "
;
340 IF BANKROLL>0 THEN 120
350 POKE 82,20
360 FOR I=0 TO 23:POSITION 20,I:? "
                ";:NEXT I
370 POSITION 20,0:? "I'M SORRY":? "YOU
 HAVE GONE BROKE":? "IF YOU WISH TO BU
Y MORE":? "CHANGE PRESS START"
380 ? "PRESS SELECT IF YOU":? "WISH TO
 QUIT"
390 IF PEEK(53279)<>6 AND PEEK(53279)<
>5 THEN 390
400 IF PEEK(53279)=6 THEN POKE 82,2:GO
TO 11
420 POSITION 20,18:? "THANK YOU":? "FO
R PLAYING, BETTER":? "LUCK NEXT TIME":
END
500 POSITION 20,20:? "5 LINE PLAY";
510 BET=1:GOSUB 1300:BET=0
520 POSITION 20,22:? "BANKROLL:";BANKR
OLL;:POSITION 20,23:? "BET:";BET;:POSI
TION 30,23:? "WIN:";WIN;
530 IF BET<5 THEN POSITION 20,21:PRINT
 "PLAY 1 TO 5 COINS";
532 FOR DELAY=1 TO 5:NEXT DELAY
535 IF BANKROLL=0 OR BET=5 THEN 560
540 IF STRIG(0)=0 THEN BET=BET+1:BANKR
OLL=BANKROLL-1:SOUND 0,50,10,14
545 IF PEEK(53279)=3 AND BET=0 THEN 90
550 IF BET=1 THEN POSITION 4,8:PRINT "
■";
552 IF BET=2 THEN POSITION 4,6:PRINT "
■";
554 IF BET=3 THEN POSITION 4,10:PRINT
"■";
556 IF BET=4 THEN POSITION 4,4:PRINT "
■";
558 IF BET=5 THEN POSITION 4,12:PRINT
"■";
560 POSITION 29,22:PRINT BANKROLL;" ";
:POSITION 24,23:PRINT BET;
562 IF BET=5 THEN POSITION 20,21:PRINT
 "                "
565 FOR DELAY=1 TO 20:NEXT DELAY
566 SOUND 0,0,0,0
570 IF STICK(0)=15 THEN 530
575 IF BET=0 THEN 530
580 GOSUB 2300
590 GOSUB SPIN
600 LOCATE 5,8,LM:POSITION 5,8:? CHR$(
LM):LOCATE 8,8,MM:POSITION 8,8:? CHR$(
MM):LOCATE 11,8,RM:POSITION 11,8:? CHR
$(RM)
605 LOCATE 5,6,LT:POSITION 5,6:? CHR$(
LT):LOCATE 8,6,MT:POSITION 8,6:? CHR$(
MT):LOCATE 11,6,RT:POSITION 11,6:? CHR
$(RT)
610 LOCATE 5,10,LB:POSITION 5,10:? CHR
$(LB):LOCATE 8,10,MB:POSITION 8,10:? C
HR$(MB):LOCATE 11,10,RB:POSITION 11,10
:? CHR$(RB)
620 IF (LM=CH AND MM<>CH) OR (LM=CH AN
D MM=LM) THEN F=LM:S=MM:T=RM:GOSUB FPA
Y
621 IF LM=DB AND MM=SB AND (RM=DB OR R
M=SB) THEN F=LM:S=MM:T=RM:GOSUB FPAY
622 IF LM=MM AND RM=MM THEN F=LM:S=MM:
T=RM:GOSUB FPAY
623 IF LM=DB AND (MM=DB OR MM=SB) AND
RM=SB THEN F=LM:S=MM:T=RM:GOSUB FPAY
624 IF LM<>CH AND LM<>DB AND LM<>SB AN
D LM<>105 THEN IF LM=MM AND (RM=DB OR
RM=SB) THEN 629
625 IF LM=SB AND MM=DB AND (RM=DB OR R
M=SB) THEN F=LM:S=MM:T=RM:GOSUB FPAY
626 IF LM=SB AND (MM=DB OR MM=SB) AND
RM=DB THEN F=LM:S=MM:T=RM:GOSUB FPAY
628 GOTO 630
629 F=LM:S=MM:T=RM:GOSUB FPAY
630 IF BET=1 THEN 680
631 IF LT=DB AND MT=SB AND (RT=DB OR R
T=SB) THEN F=LT:S=MT:T=RT:GOSUB FPAY
632 IF (LT=CH AND MT<>CH) OR (LT=CH AN
D MT=CH) THEN F=LT:S=MT:T=RT:GOSUB FPA
Y
633 IF LT=DB AND (MT=SB OR MT=DB) AND
RT=SB THEN F=LT:S=MT:T=RT:GOSUB FPAY
634 IF LT=MT AND RT=MT THEN F=LT:S=MT:
T=RT:GOSUB FPAY
635 IF LT=SB AND MT=DB AND (RT=DB OR R
T=SB) THEN F=LT:S=MT:T=RT:GOSUB FPAY
636 IF LT<>CH AND LT<>105 AND LT<>DB A
ND LT<>SB THEN IF LT=MT AND (RT=DB OR
RT=SB) THEN 640
637 IF LT=SB AND (MT=SB OR MT=DB) AND
RT=DB THEN F=LT:S=MT:T=RT:GOSUB FPAY
638 GOTO 642
640 F=LT:S=MT:T=RT:GOSUB FPAY
642 IF BET=2 THEN 680
643 IF LB=DB AND MB=SB AND (RB=DB OR R
B=SB) THEN F=LB:S=MB:T=RB:GOSUB FPAY
644 IF (LB=CH AND MB<>CH) OR (LB=CH AN
D MB=CH) THEN 652
645 IF LB=DB AND (MB=DB OR MB=SB) AND
RB=SB THEN F=LB:S=MB:T=RB:GOSUB FPAY
```

```
646 IF LB=MB AND RB=MB THEN 652
647 IF LB=SB AND MB=DB AND (RB=DB OR R
B=SB) THEN F=LB:S=MB:T=RB:GOSUB FPAY
648 IF LB<>CH AND LB<>105 AND LB<>DB A
ND LB<>SB THEN IF LB=MB AND (RB=DB OR
RB=SB) THEN 652
649 IF LB=SB AND (MB=DB OR MB=SB) AND
RB=DB THEN F=LB:S=MB:T=RB:GOSUB FPAY
650 GOTO 654
652 F=LB:S=MB:T=RB:GOSUB FPAY
654 IF BET=3 THEN 680
655 IF LT=DB AND MM=SB AND (RB=DB OR R
B=SB) THEN F=LT:S=MM:T=RB:GOSUB FPAY
656 IF (LT=CH AND MM=CH) OR (LT=CH AND
 MM<>CH) THEN 664
657 IF LT=DB AND (MM=DB OR MM=SB) AND
RB=SB THEN F=LT:S=MM:T=RB:GOSUB FPAY
658 IF LT=MM AND RB=MM THEN 664
659 IF LT=SB AND MM=DB AND (RB=DB OR R
B=SB) THEN F=LT:S=MM:T=RB:GOSUB FPAY
660 IF LT<>CH AND LT<>105 AND LT<>DB A
ND LT<>SB THEN IF LT=MM AND (RB=DB OR
RB=SB) THEN 664
661 IF LT=SB AND (MM=DB OR MM=SB) AND
RB=DB THEN F=LT:S=MM:T=RB:GOSUB FPAY
662 GOTO 665
664 F=LT:S=MM:T=RB:GOSUB FPAY
665 IF BET=4 THEN 680
666 IF LB=105 AND MM=LB AND RT=MM THEN
 WIN=WIN+2000:GOSUB 2600
667 IF LB=DB AND MM=SB AND (RT=DB OR R
T=SB) THEN F=LB:S=MM:T=RT:GOSUB FPAY
668 IF (LB=CH AND MM<>CH) OR (LB=CH AN
D MM=CH) THEN 676
669 IF LB=DB AND (MM=DB OR MM=SB) AND
RT=SB THEN F=LB:S=MM:T=RT:GOSUB FPAY
670 IF LB<>105 AND LB=MM AND RT=MM THE
N 676
671 IF LB=SB AND MM=DB AND (RT=DB OR R
T=SB) THEN F=LB:S=MM:T=RT:GOSUB FPAY
672 IF LB<>CH AND LB<>105 AND LB<>DB A
ND LB<>SB THEN IF LB=MM AND (RT=DB OR
RT=SB) THEN 676
673 IF LB=SB AND (MM=DB OR MM=SB) AND
RT=DB THEN F=LB:S=MM:T=RT:GOSUB FPAY
674 GOTO 680
676 F=LB:S=MM:T=RT:GOSUB FPAY
680 BANKROLL=BANKROLL+WIN:POSITION 29,
22:? BANKROLL;
685 IF WIN>0 AND WIN<10 THEN DUR=2:GOS
UB WINSOUND
690 IF WIN>=10 AND WIN<25 THEN DUR=3:G
OSUB WINSOUND
691 IF WIN>=25 AND WIN<=50 THEN DUR=5:
GOSUB WINSOUND
695 IF WIN>50 AND WIN<2000 THEN DUR=10
:GOSUB WINSOUND
700 POSITION 4,4:? "┘";:POSITION 4,6:?
 "┘";:POSITION 4,10:? "┘";:POSITION 4,
12:? "├";
705 BET=0:POSITION 24,21:PRINT BET;
710 IF BANKROLL>0 THEN 530
720 GOTO 350
1000 REM WINNER SOUND
1010 FOR I=1 TO DUR
1015 FOR S=40 TO 90 STEP 5
1020 SOUND 0,S,10,10
1025 NEXT S
1030 FOR S=90 TO 40 STEP -5
1035 SOUND 0,S,10,10
1040 NEXT S
1050 NEXT I
1055 SOUND 0,0,0,0:RETURN
1300 PS=2
1310 FOR I=1 TO 8
1320 POSITION 34,PS:? ASC(PAY$(I,I))*B
ET;" ";
1325 PS=PS+2
1330 NEXT I
1340 IF BET<5 THEN POSITION 34,18:? AS
C(PAY$(9,9))*BET;" ";:RETURN
1350 IF BET=5 THEN POSITION 34,18:? AS
C(PAY$(9,9))*10;
1360 RETURN
2000 POKE 752,1:? CHR$(125):POSITION 2
,2:PRINT "├┼┼┼┼┼┼┼┼┼┼┼┼┼┤    ab       P
AYS 2"
2005 POKE 756,PEEK(106)+1:SETCOLOR 1,0
,0:SETCOLOR 2,0,15
2010 PRINT "┌┬┬┬┬┬┬┬┬┬┬┬┬┐"
2012 PRINT "├┼┼┼┼┼┼┼┼┼┼┼┼┤    ab ab
 PAYS 5"
2014 PRINT "├┬  ┘  ┘  └┬┬  "
2016 PRINT "├┬  ┘  ┘  └┬┬  "    cd cd cd
 PAYS 10"
2020 PRINT "├┬. ▪  ▪  ▪┬┬▛"
2025 PRINT "├┬. ▪  ▪  ▪┬┬▛"    ef ef ef
 PAYS 14"
2030 PRINT "├┬  ┘  ┘  └┬┬▌"
2035 PRINT "├┬  ┘  ┘  └┬┬▌"    gh gh gh
 PAYS 18"
2040 PRINT "├┬  ┘  ┘  └┬┬▌"
2045 PRINT "├┼┼┼┼┼┼┼┼┼┼┼┼▌"    ■ ■ ■
 PAYS 20"
2046 PRINT "▌▌▌▌▌▌▌▌▌▌▌▌▙"
2048 PRINT "▚▚▚▚▚▚▚▚▚▚▚▚▙"    MIX BARS
 PAYS 20"
2050 PRINT "▚▒▒▒▒▒▒▒▒▒▒▒▛"
2055 PRINT "▚▒▒▒▒▒▒▒▒▒▒▒▛"    = = =
 PAYS 50"
2060 PRINT "▚▒▒▒▒▒▒▒▒▒▒▒▌"
2065 PRINT "▚▒▒▒▒▒▒▒▒▒▒▒▌"    ij ij ij
 PAYS 200"
2066 PRINT "▚▒▒▒▒▒▒▒▒▒▒▒▌"
2068 PRINT "├┼┼┼┼┼┼┼┼┼┼┼┤"
2069 PRINT "├┼┼┼┼┼┼┼┼┼┼┼┤"
2070 PRINT "├┼╱┼┼┼┼┼┼╲┼┤"
2080 PRINT "├┼┼┼┼┼┼┼┼┼┼┼┤";
2090 RETURN
2100 POKE 752,1:FOR I=19 TO 23
2110 POSITION 20,I
2120 PRINT "                    ";
2130 NEXT I
2140 RETURN
2200 FOR I=1 TO 5
2210 POSITION 20,16:PRINT "PLAY 1 TO 5
 COINS";
2220 FOR DELAY=1 TO 10:NEXT DELAY
2230 POSITION 20,16:PRINT "PLAY 1 TO 5
 COINS";
2240 FOR DELAY=1 TO 10:NEXT DELAY
2250 NEXT I
2260 RETURN
2300 POKE 752,1:POSITION 17,7
2310 FOR I=1 TO 5
2320 PRINT " ↓←■←";
2325 FOR DELAY=1 TO 20:NEXT DELAY
2330 NEXT I
2340 FOR I=1 TO 5
2350 PRINT "■↑←■←";
2355 FOR DELAY=1 TO 20:NEXT DELAY
2360 NEXT I
2370 WIN=0:POSITION 34,23:? WIN;"  ";
2390 RETURN
2400 IF LM=CH AND MM<>CH THEN WIN=BET*
2
2410 IF LM=CH AND MM=CH THEN WIN=BET*5
2420 IF LM=99 AND MM=LM AND RM=MM THEN
 WIN=BET*10
2425 IF LM=99 AND MM=99 AND (RM=DB OR
RM=SB) THEN WIN=WIN*10
2430 IF LM=101 AND MM=LM AND RM=MM THE
N WIN=BET*14
2435 IF LM=101 AND MM=101 AND (RM=DB O
R RM=SB) THEN WIN=BET*14
2440 IF LM=103 AND MM=LM AND RM=MM THE
N WIN=BET*18
2445 IF LM=103 AND MM=103 AND (RM=DB O
R RM=SB) THEN WIN=BET*18
2450 IF LM=DB AND MM=LM AND RM=MM THEN
 WIN=BET*50
2452 IF LM=SB AND MM=LM AND RM=MM THEN
 WIN=BET*20
2453 IF LM=DB AND MM=SN AND (RM=DB OR
RM=SB) THEN WIN=BET*20
2454 IF LM=DB AND (MM=DB OR MM=SB) AND
 RM=SB THEN WIN=BET*20
2455 IF LM=SB AND MM=DB AND (RM=DB OR
RM=SB) THEN WIN=BET*20
2456 IF LM=SB AND (MM=DB OR MM=SB) AND
 RM=DB THEN WIN=BET*20
2460 IF LM=105 AND MM=LM AND RM=MM AND
 BET<5 THEN WIN=BET*200
2470 IF LM=105 AND MM=LM AND RM=MM AND
 BET=5 THEN WIN=BET*2000:GOSUB 2600
```

```
2480 POSITION 34,23:PRINT WIN;" ";:BAN
KROLL=BANKROLL+WIN
2490 POSITION 29,22:PRINT BANKROLL;" "
;
2495 RETURN
2600 FOR I=0 TO 200 STEP 5
2605 SOUND 0,I,0,15
2610 NEXT I
2615 FOR I=1 TO 5
2620 FOR S=40 TO 90 STEP 2
2625 SOUND 0,S,10,10
2630 NEXT S
2640 FOR S=90 TO 40 STEP -2
2645 SOUND 0,S,10,10
2650 NEXT S:NEXT I
2655 FOR I=1 TO 20
2660 FOR I=0 TO 200 STEP 5
2665 SOUND 0,I,0,15
2670 NEXT I:SOUND 0,0,0,0
2672 FOR I=1 TO 10
2673 FOR DELAY=1 TO 40:NEXT DELAY
2674 POSITION 20,20:? "SUPER JACKPOT!!
!";
2675 FOR DELAY=1 TO 20:NEXT DELAY
2676 FOR DELAY=1 TO 20:NEXT DELAY
2678 POSITION 20,20:? "
 ";
2680 NEXT I
2685 POSITION 20,20:PRINT "
      ";
2690 RETURN
2800 FOR I=1 TO 200 STEP 25
2810 SOUND 0,I,6,8
2820 NEXT I
2830 SOUND 0,0,0,0:RETURN
2950 L=L+INT(RND(0)*6)*2:IF L>59 THEN
L=L-60
2960 M=M+16+INT(RND(0)*6)*2:IF M>59 TH
EN M=M-60
2970 R=R+22+INT(RND(0)*6)*2:IF R>59 TH
EN R=R-60
3000 POKE 77,0:FOR X=1 TO 15
3010 POSITION 11,10:PRINT R$(R,R+1):R=
R+2:IF R>59 THEN R=1
3020 POSITION 11,8:PRINT R$(R,R+1):R=R
+2:IF R>59 THEN R=1
3030 POSITION 11,6:PRINT R$(R,R+1)
3040 R=R-2:IF R<1 THEN R=R+60
3045 IF X=11 THEN GOSUB 2800
3050 IF X>10 THEN 3110
3060 POSITION 8,10:PRINT M$(M,M+1):M=M
+2:IF M>59 THEN M=1
3070 POSITION 8,8:PRINT M$(M,M+1):M=M+
2:IF M>59 THEN M=1
3080 POSITION 8,6:PRINT M$(M,M+1)
3100 M=M-2:IF M<1 THEN M=M+60
3105 IF X=6 THEN GOSUB 2800
3110 IF X>5 THEN 3160
3120 POSITION 5,10:PRINT L$(L,L+1):L=L
+2:IF L>59 THEN L=1
3130 POSITION 5,8:PRINT L$(L,L+1):L=L+
2:IF L>59 THEN L=1
3140 POSITION 5,6:PRINT L$(L,L+1)
3150 L=L-2:IF L<1 THEN L=L+60
3160 NEXT X
3165 GOSUB 2800
3170 L=L-2:IF L<1 THEN L=L+60:M=M-2:IF
 M<1 THEN M=M+60:R=R-2:IF R<1 THEN R=R
+60
3180 RETURN
4000 IF CHR$(F)="a" AND CHR$(S)<>"a" T
HEN W=2
4010 IF F=CH AND S=CH THEN W=5
4020 IF F=99 AND S=99 AND T=99 THEN W=
10
4025 IF F=99 AND S=99 AND (T=DB OR T=5
B) THEN W=10
4030 IF F=101 AND S=101 AND T=101 THEN
 W=14
4035 IF F=101 AND S=101 AND (T=DB OR T
=5B) THEN W=14
4040 IF F=103 AND S=103 AND T=103 THEN
 W=18
4045 IF F=103 AND S=103 AND (T=DB OR T
=5B) THEN W=18
4050 IF F=DB AND S=F AND T=S THEN W=20
4052 IF F=DB AND S=5B AND (T=DB OR T=5
B) THEN W=20
4053 IF F=DB AND (S=DB OR S=5B) AND T=
5B THEN W=20
4054 IF F=5B AND S=DB AND (T=DB OR T=5
B) THEN W=20
4055 IF F=5B AND (S=DB OR S=5B) AND T=
DB THEN W=20
4058 IF F=5B AND S=F AND T=S THEN W=20
4060 IF F=105 AND S=F AND T=S THEN W=2
00
4065 WIN=WIN+W:POSITION 34,23:PRINT WI
N;
4080 RETURN
10000 V=0:FOR I=0 TO 200 STEP 25
10005 SOUND 0,I,0,15
10006 SOUND 1,I,2,15:SOUND 2,I,4,15
10010 NEXT I
10015 SOUND 0,0,0,0:SOUND 1,0,0,0:SOUN
D 2,0,0,0
10090 STOP
20000 FOR I=1 TO 5
20005 FOR S=0 TO 200 STEP 5
20010 SOUND 0,S,8,15
20015 NEXT S
20020 FOR S=200 TO 0 STEP -5
20025 SOUND 0,S,8,15
20030 NEXT S
20035 NEXT I
20040 SOUND 0,0,0,0
20045 STOP
32000 POKE 106,PEEK(106)-5:GRAPHICS 2:
START=(PEEK(106)+1)*256:POKE 756,START
/256:POKE 752,1
32005 DIM XFR$(38):RESTORE 32016:FOR X
=1 TO 38:READ N:XFR$(X,X)=CHR$(N):NEXT
 X
32010 ? #6;"     ************"
32011 ? #6;"     * COLOR    *"
32012 ? #6;"     * slot     *"
32013 ? #6;"     * machine  *"
32014 ? #6;"     ************"
32015 ? "BY MICHAEL A. IVINS"
32016 DATA 104,169,0,133,203,133,205,1
69,224,133,206,165,106,24,105,1,133,20
4,160,0,177,205,145,203,200
32017 DATA 208,249,230,204,230,206,165
,206,201,228,208,237,96
32020 Z=USR(ADR(XFR$)):RESTORE 32100
32030 READ X:IF X=-1 THEN RESTORE :RET
URN
32040 FOR Y=0 TO 7:READ Z:POKE X+Y+STA
RT,Z:NEXT Y:GOTO 32030
32100 DATA 520,170,170,170,170,170,170
,170,170
32101 DATA 528,170,85,170,85,170,85,17
0,85
32102 DATA 536,170,0,170,0,170,0,170,0
32103 DATA 544,160,160,160,160,10,10,1
0,10
32104 DATA 552,80,80,80,80,5,5,5,5
32105 DATA 560,128,128,160,160,168,168
,170,170
32106 DATA 568,2,2,10,10,42,42,170,170
32107 DATA 584,234,184,46,139,46,186,2
24,170
32108 DATA 600,167,28,114,200,114,156,
39,170
32109 DATA 608,170,0,170,255,255,170,0
,170
32110 DATA 616,1,171,7,175,31,191,127,
255
32111 DATA 624,255,127,191,159,175,167
,171,169
32112 DATA 776,2,82,82,81,1,81,80,80
32113 DATA 784,170,168,128,64,64,64,0,
0
32114 DATA 792,1,5,5,21,21,5,5,1
32115 DATA 800,64,80,80,84,84,80,80,64
32116 DATA 808,2,10,10,42,42,10,10,2
32117 DATA 816,128,160,160,168,168,160
,160,128
32118 DATA 824,1,2,1,2,5,10,21,3
32119 DATA 832,0,128,64,128,64,160,80,
128
32120 DATA 840,85,85,64,0,1,5,4,20
32121 DATA 848,85,84,4,16,80,64,0,0
32122 DATA -1
```

•

CHECKSUM DATA

(See pgs. 7-10)

```
1 DATA 536,632,899,895,184,307,678,879
,220,295,143,351,720,17,548,7304
130 DATA 158,933,148,360,909,92,994,46
8,797,966,134,68,248,261,668,7204
315 DATA 806,170,901,19,825,100,364,78
6,155,968,421,996,516,310,354,7691
510 DATA 331,142,28,364,552,940,749,33
3,334,453,342,473,808,774,350,6973
566 DATA 101,487,816,825,96,930,995,30
6,27,725,148,803,578,759,836,8432
628 DATA 726,175,819,805,105,911,237,8
39,650,945,734,214,825,612,890,9487
645 DATA 646,76,646,390,680,737,111,83
1,733,26,811,137,767,613,838,8042
662 DATA 746,192,836,995,720,954,798,5
90,747,461,825,752,161,432,797,10006
690 DATA 996,245,74,90,719,369,720,55,
744,95,402,526,271,413,518,6237
1050 DATA 489,246,280,158,568,667,496,
408,565,794,71,732,697,143,187,6501
2016 DATA 760,436,815,167,822,168,181,
736,991,278,542,614,61,168,807,7546
2069 DATA 720,241,977,790,616,427,15,4
92,788,147,589,373,111,375,497,7158
2260 DATA 793,627,151,542,387,498,154,
528,390,501,794,799,130,865,610,7769
2425 DATA 469,248,664,267,687,646,662,
567,691,520,680,796,269,418,88,7672
2495 DATA 812,150,565,505,170,97,432,5
37,289,434,540,368,156,571,407,6033
2672 DATA 361,399,971,401,403,22,512,1
64,808,356,320,512,260,328,643,6460
2970 DATA 689,592,351,313,607,554,746,
858,246,17,304,508,372,576,227,6960
3130 DATA 999,295,504,542,960,928,794,
496,822,983,821,378,545,399,564,10030
4050 DATA 649,949,978,953,12,710,997,8
69,793,951,446,245,662,525,488,10227
20000 DATA 262,370,531,689,687,486,678
,683,565,495,288,843,671,391,128,7767
32013 DATA 277,683,590,657,594,768,663
,960,69,606,40,500,932,128,17,7484
32107 DATA 702,619,597,366,154,959,307
,428,122,866,127,406,371,919,754,7697
32122 DATA 832,832
```

•

HALLS OF THE LEPRECHAUN KING

16K Cassette 24K Disk

by Keith Evans and Ted Adkinson

Alas! The Leprechaun King has awakened from his long slumber, and he has taken all of the world's gold. Every nation is bankrupt. The world's only chance is Smiley, the famous gold miner. With his dexterity and wit, Smiley just might be able to recapture all of the gold, pick up the magic key, and put the gold in a sanctuary. But unless he's careful, the Leprechaun King will give him the Midas touch, turning him into a 24-carat gold tombstone.

When the game begins, take some time to notice where everything is positioned. Smiley is in the upper middle of the screen, and the Leprechaun King is in the upper right hand corner. Throughout the maze there are pots of gold. To collect one, just touch it. If you look in the lower right corner, you will see the magic key surrounded by walls. Collect about half of the gold, and the key will move to the center of the maze.

After Smiley gets the key and all of the gold, he goes to the sanctuary chamber at the far lower right corner, directly to the left of the cross. Push the trigger button, and a section of the wall will disappear. This is the entrance to the sanctuary where Smiley has to store the gold. Deposit gold by simply touching the cross.

Another important part of this game is the gold tombstones. When Smiley loses a life, a tombstone appears as a resting place for all of the gold he was carrying. A new Smiley has to touch the tombstone to collect the gold that the old Smiley was carrying.

You start with three lives. The game is over when you use them all up. To see how many lives you have left, look in the upper right or left hand corner of the screen where vertical bars indicate lives remaining (including the one currently in use).

An expert player might get to the third maze and find it is totally different. Two clues about this maze: the key appears in the lower middle of the screen, and the section of disappearing wall lies directly below the cross. □

The program.

Lines 1-10 — Variable initialization, title
Lines 120-372 — Character set redefinition
Lines 395-507 — Maze drawing, placing of the gold
Lines 510-624 — Joystick reading, movement of Smiley
Lines 630-999 — The Leprechaun's logic
Lines 1000-1120 — Maze and character set data
Lines 1150-1154 — Lives left indicator
Lines 1500-1510 — "Midas Touch" sound effects
Lines 2000-2020 — Counts bags of gold taken, places key in the maze if enough has been taken
Line 2500 — Draws tombstone, checks men left
Lines 2510-2570 — Erases Smiley's trail
Line 2575 — Starts game over when all men are used up
Lines 2610-2700 — Puts the gold Smiley was carrying in a tombstone when he is killed
Lines 3000-3050 — Subroutine to flash maze
Lines 4000-4350 — Actually moves Leprechaun.
Line 5000 — Plays "Oh, when the saints...," clears screen
Lines 6000-6007 — Displays score at end of game
Line 6010 — Clears screen
Lines 7000-8030 — Data for "Oh, when the saints..."
Lines 9000-9130 — Subroutine to play "Oh, when the saints..."
Lines 9150-9260 — Plays "Good night, ladies..."

Lines 9270-9290 — Data for "Good night, ladies..."

Line 9300 — Sound effects of gold being cashed in

Lines 10000-10020 — Color rotation subroutine

```
1 CLR :X=10:Y=1:MX=17:MY=2:X1=10:Y1=1
5 GRAPHICS 2+16:? #6;" ":? #6;" "
6 ? #6;"  THE HALLS OF THE":? #6;"   L
EPRECHAN KING":? #6;" "
7 ? #6;"        created":? #6;" "
8 ? #6;"          by":? #6;" "
9 ? #6;"     keith and ted ":? #6;" ":?
 #6;"    PLEASE WAIT "
10 FOR ZZZ=1 TO 20:GOSUB 10000:NEXT ZZ
Z
120 POKE 106,PEEK(106)-2
130 GRAPHICS 1+16
150 A=PEEK(106)*256
190 SET=PEEK(106)
200 POKE 756,SET
220 FOR C=0 TO 7
230 POKE A+C,0
240 NEXT C
250 FOR C=8 TO 63
260 READ CHAR
270 POKE A+C,CHAR
280 NEXT C
369 FOR C=64 TO 71:POKE A+C,146:NEXT C
370 FOR C=72 TO 79:POKE A+C,144:NEXT C
371 FOR C=80 TO 87:POKE A+C,128:NEXT C
372 FOR C=88 TO 95:READ CHAR:POKE A+C,
CHAR:NEXT C
395 IF TIM>=1 AND TIM<3 THEN RESTORE 1
040
396 IF TIM>=3 THEN RESTORE 7000
397 TIM=TIM+1
398 MM=2:IF TIM=1 OR TIM=5 THEN MM=1
400 READ GR1,GR2,GR3,GR4
410 IF GR1=-1 THEN GOTO 440
420 COLOR 35:PLOT GR1,GR2:DRAWTO GR3,G
R4
430 GOTO 400
440 READ G1,G2
450 IF G1=-1 THEN 500
460 COLOR 130:PLOT G1,G2
470 GOTO 440
500 IF TIM<4 THEN COLOR 35:PLOT 3,2:PL
OT 7,3:PLOT 6,3:PLOT 1,16:COLOR 32:PLO
T 12,14
502 BAGS=0:DBAGS=0:GOLD=0:KEY=0:IF TIM
<4 THEN COLOR 37:PLOT 18,22
503 IF TIM<4 THEN RESTORE 1120
504 IF TIM>=4 THEN RESTORE 7090:LOCATE
 10,11,ZZ:IF ZZ=32 THEN COLOR 37:PLOT
10,11
506 X=10:Y=1:READ RMX:READ RMY:MX=RMX:
MY=RMY:X1=10:Y1=1
507 READ SDO,SDO1,SD,SD1,SC,SC1,K,E,AX
,AY,NB
510 X1=X:Y1=Y
515 POKE 711,251
516 POKE 77,0
520 IF STICK(0)=15 THEN GOTO 580
530 J=STICK(0)
540 IF J=11 THEN X=X-1
550 IF J=7 THEN X=X+1
560 IF J=14 THEN Y=Y-1
570 IF J=13 THEN Y=Y+1
580 LOCATE X,Y,I:IF I=35 THEN X=X1:Y=Y
1
590 IF I=130 THEN GOSUB 2000
595 IF I=38 THEN GOLD=GOLD+DGOLD:BAGS=
DBAGS:FOR ZZ=-30 TO 30:SOUND 0,ABS(ZZ)
,10,8:NEXT ZZ:SOUND 0,0,0,0
600 IF I=1 THEN GOSUB 1500:GOTO 2500
605 IF I=37 THEN KEY=1:ZZZ=60:FOR ZZ=6
0 TO 40 STEP -1:SOUND 0,ZZ,10,8:SOUND
1,ZZZ,10,8:ZZZ=ZZZ-1:NEXT ZZ
606 SOUND 0,0,0,0:SOUND 1,0,0,0
615 IF J<>15 THEN COLOR 32:PLOT X1,Y1
620 COLOR 36:PLOT X,Y
622 IF X=SDO AND Y=SDO1 AND KEY=1 AND
STRIG(0)=0 THEN COLOR 32:PLOT SD,SD1
623 IF X=SC AND Y=SC1 THEN PGOLD=PGOLD
+GOLD:COLOR 39:PLOT SC,SC1:X=AX:X1=X:Y
=AY:Y1=Y:GOLD=0:GOSUB 9300
624 IF BAGS<=NB AND I=39 THEN 5000
630 MM=MM*-1
640 IF MM=1 THEN 510
650 LOCATE MX-1,MY,D1
660 LOCATE MX,MY-1,D2
670 LOCATE MX+1,MY,D3
680 LOCATE MX,MY+1,D4
690 IF X<>MX AND Y<>MY THEN 750
700 IF X=MX AND MY>Y THEN FD=2:FD1=0
710 IF X=MX AND MY<Y THEN FD=4:FD1=0
720 IF Y=MY AND MX>X THEN FD=1:FD1=0
730 IF Y=MY AND MX<X THEN FD=3:FD1=0
740 GOTO 790
750 IF MX<X THEN FD=3
760 IF MX>X THEN FD=1
770 IF MY<Y THEN FD1=4
780 IF MY>Y THEN FD1=2
790 REM
795 IF FD1<>0 THEN 900
800 IF FD=4 AND D4<>35 THEN RD=4:GOTO
1150
810 IF FD=3 AND D3<>35 THEN RD=3:GOTO
1150
820 IF FD=2 AND D2<>35 THEN RD=2:GOTO
1150
830 IF FD=1 AND D1<>35 THEN RD=1:GOTO
1150
840 RD=INT(RND(0)*4)+1
850 IF RD=1 AND D1=35 THEN 840
860 IF RD=2 AND D2=35 THEN 840
870 IF RD=3 AND D3=35 THEN 840
880 IF RD=4 AND D4=35 THEN 840
890 GOTO 1150
900 WAYS=0:IF FD=1 AND D1<>35 THEN WAY
S=WAYS+1:W1=1
902 IF FD=2 AND D2<>35 THEN WAYS=WAYS+
1:W2=1
904 IF FD=3 AND D3<>35 THEN WAYS=WAYS+
1:W3=1
906 IF FD=4 AND D4<>35 THEN WAYS=WAYS+
1:W4=1
908 IF FD1=1 AND D1<>35 THEN WAYS=WAYS
+1:W11=1
910 IF FD1=2 AND D2<>35 THEN WAYS=WAYS
+1:W22=1
912 IF FD1=3 AND D3<>35 THEN WAYS=WAYS
+1:W33=1
914 IF FD1=4 AND D4<>35 THEN WAYS=WAYS
+1:W44=1
916 IF WAYS=2 THEN 4000
918 IF W1=1 THEN RD=1
920 IF W2=1 THEN RD=2
922 IF W3=1 THEN RD=3
924 IF W4=1 THEN RD=4
925 GOTO 4070
926 GOTO 1150
999 GOTO 510
1000 DATA 170,84,124,170,146,254,40,10
8
1010 DATA 126,60,66,223,209,219,66,60
1020 DATA 170,85,170,85,170,85,170,85
1030 DATA 60,126,219,255,189,195,126,6
0
1035 DATA 0,0,7,253,85,87,0,0
1037 DATA 28,54,119,65,119,119,119,127
1038 DATA 24,24,126,126,24,24,24,24
1039 DATA 31,35,69,249,137,138,140,248
1040 DATA 13,13,14,13,2,14,4,14,5,15,4
,15,5,16,8,16,15,15,16,15,13,16,14,16,
2,18,5,18,7,18,9,18,15,18,17,18
1041 DATA 1,0,18,0
1050 DATA 2,19,3,19,7,19,9,19,11,19,13
,19,5,20,7,20,16,20,18,20,2,21,3,21,5,
21,7,21,9,21,14,21,2,22,3,22
```

```
1060 DATA 1,1,1,5,18,1,18,7,9,1,9,4,16
,6,16,8,18,12,18,16,16,14,16,17,16,20,
16,22,13,10,13,11,13,17,13,18
1070 DATA 9,9,9,10,8,13,8,14,3,16,3,17
,0,0,0,23,0,23,19,23,19,23,19,0,3,1,9,
1,11,1,16,1,11,2,16,2,11,4,16,4
1080 DATA 4,6,9,6,3,3,5,3,6,4,7,4,3,5,
4,5,11,5,12,5,14,6,16,6,11,7,12,7,2,8,
6,8,8,8,9,8,11,8,14,8,5,9,6,9
1090 DATA 11,9,13,9,16,9,17,9,1,10,3,1
0,6,10,7,10,15,10,17,10,6,11,7,11,9,11
,11,11,2,12,3,12,17,12,18,12,5,13,6,13
1095 DATA 10,13,16,17,11,13,11,17,-1,0
,0,0
1100 DATA 4,2,5,5,13,7,4,9,8,9,12,10,1
8,11,15,12,3,13,9,13,5,14,13,14,12,15,
4,16,15,17,6,18,4,21,12,22,15,20
1110 DATA 2,6,-1,0
1120 DATA 17,2,15,22,16,22,18,22,9,12,
17,22,-19
1150 IF LI=0 THEN COLOR 8:PLOT 19,0:PL
OT 0,0
1151 IF LI=-1 THEN COLOR 9:PLOT 19,0:P
LOT 0,0
1152 IF LI=-2 THEN COLOR 10:PLOT 19,0:
PLOT 0,0
1154 GOTO 4110
1500 COUNT=800:FOR ZZ=20 TO 0 STEP -1:
SOUND 0,COUNT,10,ZZ:SOUND 1,COUNT+(ZZ*
99),10,ZZ:COUNT=COUNT-10:NEXT ZZ
1510 SOUND 0,0,0,0:SOUND 1,0,0,0:RETUR
N
2000 BAGS=BAGS-1:GOLD=GOLD+INT(RND(0)*
100)+1:DBAGS=DBAGS-1
2005 FOR ZZ=20 TO 0 STEP -1:SOUND 0,20
,10,ZZ:NEXT ZZ:SOUND 0,0,0,0
2010 IF DBAGS=-10 OR BAGS=-10 THEN COL
OR 37:PLOT K,E:COLOR 39:PLOT SC,SC1
2020 RETURN
2500 COLOR 38:PLOT X,Y:REM :LI=LI-1:IF
 LI=-3 THEN GOSUB 9150:GOSUB 6000:GOTO
 2570
2510 LOCATE X,Y,ZZ:IF ZZ=36 THEN COLOR
 32:PLOT X,Y
2520 LOCATE X+1,Y,ZZ:IF ZZ=36 THEN COL
OR 32:PLOT X+1,Y
2530 LOCATE X-1,Y,ZZ:IF ZZ=36 THEN COL
OR 32:PLOT X-1,Y
2540 LOCATE X,Y-1,ZZ:IF ZZ=36 THEN COL
OR 32:PLOT X,Y-1
2550 LOCATE X,Y+1,ZZ:IF ZZ=36 THEN COL
OR 32:PLOT X,Y+1
2555 LI=LI-1:IF LI=-3 THEN GOSUB 9150:
GOSUB 6000:GOTO 2570
2560 COLOR 38:PLOT X,Y
2570 X=10:Y=1:X1=X:Y1=Y:MX=17:MY=2:OMX
=MX:OMY=MY
2572 D=32
2575 IF LI=-3 THEN LI=0:GOTO 395
2610 FOR FN=0 TO 500:NEXT FN
2617 DGOLD=GOLD:GOLD=0:X=10:Y=1:X1=10:
Y1=1
2620 MX=RMX:MY=RMY
2630 D=32
2700 GOTO 503
3000 FOR COUNT=0 TO 5
3010 SETCOLOR 0,8,8
3015 FOR ZZ=1 TO 50:NEXT ZZ
3020 SETCOLOR 0,2,8
3025 FOR ZZ=1 TO 50:NEXT ZZ
3030 NEXT COUNT
3040 SETCOLOR 0,2,8
3050 RETURN
4000 RW=INT(RND(0)*2)+1
4010 IF RW=1 THEN 1110
4020 IF W1=1 THEN RD=1
4030 IF W2=1 THEN RD=2
4040 IF W3=1 THEN RD=3
4050 IF W4=1 THEN RD=4
4060 GOTO 1150
4070 IF W11=1 THEN RD=1
4080 IF W22=1 THEN RD=2
4090 IF W33=1 THEN RD=3
4100 IF W44=1 THEN RD=4
4110 IF RD=0 THEN 4300
4120 IF RD=1 THEN MX=MX-1
4130 IF RD=2 THEN MY=MY-1
4140 IF RD=3 THEN MX=MX+1
4150 IF RD=4 THEN MY=MY+1
4155 LOCATE OMX,OMY,ZZ:IF ZZ=36 OR ZZ=
39 THEN 4162
4160 COLOR D:PLOT OMX,OMY
4162 D=32
4165 LOCATE MX,MY,D:IF D=36 THEN GOSUB
 1500:GOTO 2500
4170 COLOR 1:PLOT MX,MY
4175 OMX=MX:OMY=MY
4180 FD=0:FD1=0:RD=0:D1=0:D2=0:D3=0:D4
=0:WAYS=0:RD=0
4190 W1=0:W2=0:W3=0:W4=0:W11=0:W22=0:W
33=0:W44=0
4200 GOTO 510
4300 RD=INT(RND(0)*4)+1
4310 IF RD=1 AND D1=35 THEN 4300
4320 IF RD=2 AND D2=35 THEN 4300
4330 IF RD=3 AND D3=35 THEN 4300
4340 IF RD=4 AND D4=35 THEN 4300
4350 GOTO 4120
5000 GOSUB 3000:GOSUB 9000:GOSUB 6000:
TIM=TIM+1:GOTO 395
6000 COLOR 32:C1=0:C2=0:IF LI<>-3 THEN
 GOTO 6005
6001 IF LI=-3 THEN GOSUB 6010:POKE 756
,224:POSITION 0,5:? #6;"    SCORE= ";P
GOLD
6002 POSITION 4,10:? #6;"push trigger"
:IF LI=-3 THEN TIM=0
6003 SETCOLOR 1,12,10:IF STRIG(0)=0 TH
EN 6005
6004 FOR ZZ=1 TO 50:NEXT ZZ:SETCOLOR 1
,0,0:FOR ZZ=1 TO 50:NEXT ZZ:GOTO 6003
6005 RESTORE 1040:IF LI=-3 THEN PGOLD=
0
6007 COLOR 32:GOSUB 6010:POKE 756,SET:
RETURN
6010 FOR C1=0 TO 23:PLOT 0,C1:DRAWTO 1
9,C1:NEXT C1:RETURN
7000 DATA 0,0,19,0,19,0,19,23,19,23,0,
23,0,23,0,0,3,3,5,3,5,2,5,2,9,1,11,1,1
4,3,16,3,14,2,14,2
7010 DATA 9,4,11,6,9,6,11,4,16,5,14,5,
14,6,14,6,9,10,11,10,11,10,11,12,11,12
,9,12,9,12,9,10,3,5,5,5,5,6,5,6
7020 DATA 3,9,3,10,5,9,5,10,14,9,14,10
,16,9,16,10,3,12,3,13,5,12,5,13,14,12,
14,13,16,12,16,13
7030 DATA 10,15,10,17,9,16,11,16,3,17,
5,17,14,17,16,17,3,19,5,19,14,19,16,19
,8,20,9,20
7040 DATA 11,20,12,20,8,22,9,22,11,22,
12,22,2,22,2,23,17,22,17,23,12,21,12,2
1,8,21,8,21,3,19,3,20
7050 DATA 3,16,3,17,16,16,16,17,16,19,
16,20,2,10,3,10,6,10,5,10,2,12,3,12,6,
12,5,12,13,10,14,10
7060 DATA 16,10,17,10,13,12,14,12,17,1
2,17,12,-1,-1
7070 DATA 4,2,15,2,4,6,15,6,10,4,10,6,
9,5,11,5,4,9,15,9,2,11,4,11,6,11,13,11
,15,11,17,11,4,13,15,13
7080 DATA 9,15,11,15,9,17,11,17,4,16,1
5,16,4,20,15,20,9,21,11,21,4,2,-1
7090 DATA 10,21,10,13,10,12,10,11,10,2
2,10,12,-28
8000 DATA 121,6,96,6,91,6,81,1,0,8,121
,8,96,8,91,8,81,1
8010 DATA 0,8,121,8,96,8,91,8,81,2,96,
2,121,2,96,2,108,1
8020 DATA 0,8,96,8,96,8,108,8,121,2,12
1,6,96,2,81,4,81,4,91,2
8030 DATA 0,8,91,8,96,8,91,8,81,2,96,2
,108,4,108,4,121,1,-1
9000 RESTORE 8000
9010 READ PITCH
9020 IF PITCH=-1 THEN 9130
9040 READ DURATION:DURATION=INT(50/DUR
ATION)
9050 SOUND 0,PITCH,10,8
9060 IF PITCH=0 THEN 9080
9070 SOUND 1,PITCH+1,10,8
9080 FOR ZZ=1 TO DURATION:NEXT ZZ
9090 SOUND 0,0,0,0
```

```
9100 SOUND 1,0,0,0
9110 FOR ZZ=1 TO 5:NEXT ZZ
9120 GOTO 9010
9130 RETURN
9150 RESTORE 9270
9160 READ DURATION:IF DURATION=-1 THEN
 RETURN
9170 DURATION=INT(DURATION*10)
9180 READ PITCH:IF PITCH=0 THEN 9200
9190 PITCH=PITCH*3
9200 SOUND 0,PITCH,10,8
9210 SOUND 1,PITCH+1,10,8
9220 FOR ZZ=1 TO DURATION:NEXT ZZ
9230 SOUND 0,0,0,0
9240 SOUND 1,0,0,0
9250 FOR ZZ=1 TO 3:NEXT ZZ
9260 GOTO 9160
9270 DATA 2,47,2,60,1,81,3,60,2,47,2,6
0,1,53,3,53,2,47,2,60,1,45,2,45,1,45
9280 DATA 1,47,1,47,1,53,1,53,3,60,1,0
,1.5,47,.5,53,1,60,1,53,1,47,1,47,2,47
,1,53,1,53,2,53
9290 DATA 1,47,1,40,2,40,1.5,47,.5,53,
1,60,1,53,1,47,1,47,2,47,1,53,1,53,1,4
7,1,53,3,60,1,0,-1
9300 FOR ZZZ=1 TO 3:FOR ZZ=20 TO 0 STE
P -1:SOUND 0,20,10,ZZ:NEXT ZZ:SOUND 0,
0,0,0:NEXT ZZZ:RETURN
10000 FOR Z=1 TO 3
10010 A=PEEK(708)
10020 POKE 708,PEEK(711):POKE 711,PEEK
(710):POKE 710,PEEK(709):POKE 709,A:NE
XT Z:FOR B=1 TO 10:NEXT B:RETURN
```

●

CHECKSUM DATA
(See pgs. 7-10)

```
1 DATA 0,703,82,669,687,337,397,37,199
,539,211,426,288,786,730,6091
250 DATA 162,99,303,742,713,712,721,64
2,282,323,121,875,275,455,697,7122
430 DATA 704,833,696,281,728,445,337,5
1,429,826,926,151,238,959,36,7560
530 DATA 918,74,785,93,91,205,828,681,
484,731,983,17,973,483,857,8203
624 DATA 488,689,493,829,812,831,814,1
82,910,911,917,918,746,924,929,11393
770 DATA 167,170,113,918,228,224,220,2
16,977,846,853,860,867,983,660,8302
902 DATA 660,667,674,27,30,40,50,918,8
10,807,811,815,974,970,746,8999
1000 DATA 226,165,214,287,97,68,763,80
,750,137,418,548,616,647,412,5428
1095 DATA 418,873,132,916,490,600,859,
724,251,119,657,327,491,783,339,7979
2510 DATA 724,239,248,210,201,494,202,
783,213,871,731,517,774,208,903,7318
3000 DATA 810,677,667,660,668,195,662,
788,842,933,0,5,10,15,719,7651
4070 DATA 706,715,724,726,908,549,557,
551,559,311,753,204,808,251,772,9094
4180 DATA 625,95,887,819,607,611,615,6
19,730,1,991,985,398,561,836,9380
6005 DATA 105,951,595,618,605,393,695,
974,457,283,942,188,257,974,783,8820
8020 DATA 596,86,199,110,118,997,893,4
00,342,698,223,218,588,743,801,7012
9150 DATA 226,485,338,695,558,894,342,
698,223,225,593,761,514,393,665,7610
9300 DATA 852,309,60,626,1847
```

●

For those interested, here are some of the techniques used in this program. First is "redefined character sets," which when carefully laid out can simulate a high resolution graphics screen, but requiring much less memory. They are fairly easy to design. Each character can be one of four different colors. Step one is to design some characters. Here's Smiley as an example:

Make an 8 by 8 grid, mark the blocks to be filled in, then add up the corresponding numbers to determine its POKE value.

Following is a program which defines a space and a Smiley character, and then prints out a picture of Smiley on the screen.

```
10 GRAPHICS 2+16:REM START OUT WITH A
GRAPHICS STATEMENT
20 POKE 106,PEEK(106)-2:REM SET ASIDE
2 PAGES OF MEMORY FOR THE CHARACTER SE
T
30 CHBASE=PEEK(106)*256:REM THIS IS WH
ERE THE CHARACTER SET WILL BE POKED IN
TO MEMORY
40 SET=PEEK(106):REM THIS IS WHERE THE
 CHARACTERS WILL GO IN TERMS OF PAGES
OF MEMORY
50 READ VALUE:IF VALUE=-1 THEN 95:REM
READ IN PART OF A CHARACTER
60 POKE CHBASE+C,VALUE:REM PUT THE NUM
BER IN MEMORY
70 C=C+1:GOTO 50
75 REM DATA FOR SPACE
80 DATA 0,0,0,0,0,0,0,0
85 REM DATA FOR SMILEY
90 DATA 60,126,219,255,189,195,126,60,
-1
95 POKE 756,SET:REM TELL THE COMPUTER
WHERE THE NEW CHARACTER SET IS LOCATED
100 COLOR 1:PLOT 5,5:REM PUTS SMILEY O
N SCREEN AT 5,5
110 GOTO 110:REM ENDLESS LOOP FOR DISP
LAY PURPOSES
```

To determine the number for the COLOR statement in line 100: first, Smiley is to be green. Color register number 1 normally contains green, so it is used. Smiley has been defined in the program above as the second character in the redefined set. (The space was the first.)

With these pieces of information I looked up the number in a chart, like the following one:

COLOR REGISTER 0 = 32, 33 through 95
COLOR REGISTER 1 = 0, 1 through 124, (125*), 126, 127
COLOR REGISTER 2 = 160, 161 through 223
COLOR REGISTER 3 = 128, 129 through 154, (155*), 156 through 255

*155 selects the same thing as 32. 125 has no effect.

Smiley's color is set by color register 1, so look in

the second row. Since he is the second character, use the 2nd number in the 2nd row, which is 1. As another example, if Smiley were to be controlled by color register 2, the correct number would be 161. Try 161 in the example program above and see what happens.

Before you get too carried away, remember that the example program will not allow text to be displayed on the screen. To switch back to text only, type POKE 756, 224.

Another section of the **Halls of the Leprechaun King** which is interesting is its color rotation subroutine (10000-10020). Adding this to a program's title makes it very colorful. Here is how it works. Memory locations 708-711 contain the numbers which determine the colors which will be displayed from each color register. The subroutine rotates the colors from one register to another so that everything on the screen flashes through each color. Try it in one of your programs. □

Circle Radius Demo

```
10 XCENTER=310/2:YCENTER=192/2
100 GRAPHICS 8
110 COLOR 1
120 ? "ENTER RADIUS:";:INPUT RADIUS
130 LET RADIUS=RADIUS+3-1
140 LET X=0
150 LET Y=RADIUS
160 LET DIAMETER=3-2*RADIUS
170 IF X<=Y THEN GOSUB 1000:IF DIAMETE
R<0 THEN DIAMETER=DIAMETER+4*X+6:X=X+1
:GOTO 170
180 IF X>Y THEN END
190 DIAMETER=DIAMETER+4*(X-Y)+10
200 Y=Y-1
210 X=X+1:GOTO 170
1000 REM
1010 PLOT XCENTER+X,YCENTER+Y
1020 PLOT XCENTER+Y,YCENTER+X
1030 PLOT XCENTER+Y,YCENTER-X
1040 PLOT XCENTER+X,YCENTER-Y
1050 PLOT XCENTER-X,YCENTER-Y
1060 PLOT XCENTER-Y,YCENTER-X
1070 PLOT XCENTER-Y,YCENTER+X
1080 PLOT XCENTER-X,YCENTER+Y
1090 RETURN
```

CHECKSUM DATA
(See pgs. 7-10)

```
10 DATA 50,908,474,753,651,436,523,779
,371,580,245,356,504,275,95,7000
1020 DATA 95,100,102,105,105,102,104,7
88,1501
```

STUNTMAN

16K Cassette 24K Disk

by Steven Pogatch

Your stunt man has been hired to climb to the top of every building he can find. This is not as easy as it may seem, though, because the tenants of the buildings will do anything to get you off the building. There are six (6) levels to each building, each progressing in difficulty.

In the first section, windows constantly close to keep you from getting past them. Next, men stick their heads out of the windows, trying to get in your way. After that, flower pots fall from the window ledges, closing all windows in their way. After passing this section, a crazy bird drops girders on you. Be careful here — they can be deadly if they hit you on the head. Once you get past the bird, you have to avoid King Kong, waiting for you on his part of the building. He is very angry and is throwing down anything he can find on top of you. Last (but not easiest), girders (3 lanes wide) come crashing down from the building. Look out!

If you are lucky enough to get through all of this, there will be a brief intermission telling you to go on to the next building.

On the top left corner of the screen are three numbers. The first one represents the section, the second represents the building number, and the third represents the number of men you have left. If you manage to score 10,000, 30,000 or 50,000 points, you will be awarded a free stunt man. The score is displayed in the lower left hand corner. You can move left, right and up with the joystick. For every movement you make, you are rewarded 50 points. You start out with 6 stunt men. Good luck climbing — you'll need it. □

The program.

Lines 1-30 — Initialization

Lines 40-1000 — Movement of a player, activate obstacle(s)

Lines 1000-2000 — Death (fall) of stunt man

Lines 2000-3000 — Section 1 (windows & men)

Lines 3000-4000 — Section 2 (flower pots)

Lines 4000-5000 — Section 3 (bird)

Lines 5000-6000 — Section 4 (King Kong)

Lines 6000-10000 — Section 5 (girders)

Lines 10000-11000 — Bonus stunt man

Lines 11000-32000 — Go on to next building (intermission)

Lines 32000-325000 — Redefines character set

Lines 32500-32700 — Title

Lines 32700-32750 — End of game

```
1 GOSUB 32000:CLR
2 GOSUB 32500:SH=6:B=1
5 GRAPHICS 1:POKE 756,PEEK(106)+1
10 SETCOLOR 2,0,0:POKE 710,94:POKE 711
,45:FOR A=0 TO 19:POSITION 5,A:? #6;"e
eeeeeeeeeee":NEXT A
20 FOR A=5 TO 16:F=RND(0)*19:IF F>1 TH
EN POSITION A,F:? #6;"f":NEXT A
30 X=10:Y=18:OX=X:OY=Y
40 POSITION OX,OY:? #6;"e":POSITION OX
,OY+1:? #6;"e"
41 LOCATE X,Y,Z:IF Z=102 OR Z=225 OR Z
=66 THEN GOSUB 1000
42 POSITION X,Y:? #6;"[G]":POSITION X,Y+
1:? #6;"[H]"
43 IF SC=10000 OR SC=50000 OR SC=10000
0 THEN SH=SH+1:GOSUB 10000
44 IF Y=0 THEN L=L+1:GOTO 5
45 OX=X:OY=Y
46 SOUND 0,Y+20,3,15:FOR A=1 TO 15:NEX
T A:SOUND 0,0,0,0
47 G=RND(0)*4:IF G>3.7 THEN FOR A=5 TO
 16:POSITION A,RND(0)*18:? #6;"f":NEXT
 A
48 ON L GOSUB 2000,3000,4000,5000,6000
:IF L=6 THEN L=0:B=B+1:GOSUB 11000
49 POSITION 0,19:? #6;SC:POSITION 1,1:
? #6;L:POSITION 1,2:? #6;B:POSITION 1,
3:? #6;SH
50 IF STICK(0)=14 AND Y>0 THEN Y=Y-1:S
C=SC+50:GOTO 40
60 IF STICK(0)=11 AND X>5 THEN X=X-1:S
C=SC+50:GOTO 40
70 IF STICK(0)=7 AND X<16 THEN X=X+1:S
C=SC+50:GOTO 40
100 GOTO 42
1000 SOUND 0,40,6,10:FOR A=1 TO 25:NEX
T A:SOUND 0,0,0,0:SOUND 1,0,0,0
1010 FOR A=Y TO 18:POSITION X,A:? #6;"
[G]":POSITION X,A+1:? #6;"[H]":POSITION X,
A-1:? #6;"e":POSITION X,A:? #6;"e"
1020 SOUND 0,A+20,10,10:NEXT A:SOUND 0
,0,0,0:SH=SH-1
1030 IF SH<0 OR SH=0 THEN GOTO 32700
```

```
1040 GOTO 10
2000 REM DOCTORS
2010 C=C+1:IF C=11 THEN C=1:DC=RND(0)*
10
2020 IF DC>6 THEN POSITION C+5,RND(0)*
18:? #6;"a"
2040 RETURN
3000 REM POTS
3010 D=D+1:IF D=11 THEN D=1:DC=RND(0)*
10
3020 IF DC>7 THEN FOR A=2 TO Y:POSITIO
N D+5,A:? #6;"B":POSITION D+5,A-1:? #6
;"f":NEXT A:POSITION D+5,Y:? #6;"e"
3030 RETURN
4000 REM BIRD
4010 BD=BD+1:IF BD=11 THEN BD=1
4020 POSITION BD+5,2:? #6;"C":POSITION
 BD+6,2:? #6;"D":POSITION BD+4,2:? #6;
"e":BDD=RND(0)*10
4022 IF BD=1 THEN POSITION 15,2:? #6;"
ee"
4025 IF BDD>3 THEN 4030
4027 RETURN
4030 FOR BDDA=4 TO 19:POSITION BD+5,BD
DA:? #6;"em":POSITION BD+5,BDDA-1:? #6
;"ee"
4040 LOCATE X,Y,Z:IF Z=109 THEN GOSUB
1000
4050 SOUND 0,BDDA+100,10,8:NEXT BDDA:S
OUND 0,0,0,0
4060 RETURN
5000 REM KONG
5010 KN=KN+1:IF KN=12 THEN KL=KL+1:KN=
1
5015 IF KN=1 THEN KN=KN+1:IF KL=12 THE
N KL=1
5020 POSITION KN+5,KL+1:? #6;"H":POSIT
ION KN+5,KL+2:? #6;"K":POSITION KN-1+5
,KL+1:? #6;"e":POSITION KN-1+5,KL+2:?
#6;"e"
5025 POSITION 16,KL+1:? #6;"e":POSITIO
N 16,KL+2:? #6;"e"
5026 REM FOR A=1 TO 12POSITION 15,A? #
6;"e"NEXT A
5027 LOCATE X,Y,Z:IF Z=235 THEN GOTO 1
000
5030 QQQ=RND(0)*10+1
5040 IF QQQ<6 THEN RETURN
5045 IF QQQ>6 AND QQQ<7 THEN 5050
5046 IF QQQ>7 AND QQQ<8 THEN 5200
5047 IF QQQ>8 AND QQQ<9 THEN 5300
5048 IF QQQ>9 AND QQQ<10 THEN 5400
5050 FOR A=KL+4 TO 17
5060 POSITION KN+5,A:? #6;"B":POSITION
 KN+5,A-1:? #6;"e":LOCATE X,Y,ZZ:IF ZZ
=66 THEN 1000
5070 SOUND 0,A+200,10,8:NEXT A:SOUND 0
,0,0,0:POSITION KN+5,17:? #6;"e"
5080 RETURN
5200 FOR A=KL+4 TO 17
5210 POSITION KN+5,A:? #6;"I":POSITION
 KN+5,A-1:? #6;"e":LOCATE X,Y,ZZ:IF ZZ
=73 THEN 1000
5220 SOUND 0,A+200,10,8:NEXT A:SOUND 0
,0,0,0:POSITION KN+5,17:? #6;"e"
5230 RETURN
5300 FOR A=KL+4 TO 17
5310 POSITION KN+5,A:? #6;"M":POSITION
 KN+5,A-1:? #6;"e":LOCATE X,Y,ZZ:IF ZZ
=77 THEN 1000
5320 SOUND 0,A+200,10,8:NEXT A:SOUND 0
,0,0,0:POSITION KN+5,17:? #6;"e"
5330 RETURN
5400 FOR A=KL+4 TO 17
5410 POSITION KN+5,A:? #6;"A":POSITION
 KN+5,A-1:? #6;"e":LOCATE X,Y,ZZ:IF ZZ
=65 THEN 1000
5420 SOUND 0,A+200,10,8:NEXT A:SOUND 0
,0,0,0:POSITION KN+5,17:? #6;"e"
5430 RETURN
5500 RETURN
6000 REM GIRDERS ARE MEAN
6005 GG=RND(0)*10+1:IF PL=1 THEN PL=19
6010 IF GG<4 THEN RETURN
6020 TTT=RND(0)*12+1
6030 IF TTT<5 THEN RETURN
6040 FOR A=2 TO 19:POSITION TTT,A:? #6
;"MMM":POSITION TTT,A-1:? #6;"eee":SOU
ND 0,A,TTT,12:SOUND 1,TTT,A,12
6045 LOCATE X,Y,ZZ:IF ZZ=77 THEN GOTO
1000
6047 NEXT A
6050 SOUND 0,0,0,0:SOUND 1,0,0,0
6060 POSITION TTT,19:? #6;"eee"
6070 RETURN
10000 RESTORE 10500
10010 READ SO:IF SO=-1 THEN SOUND 0,0,
0,0:RETURN
10020 SOUND 0,SO,10,14:FOR A=1 TO 2:NE
XT A:GOTO 10010
10500 DATA 243,4,162,4,121,6,96,2,102,
4,243,4,162,4,121,6,81,2,60,8,-1
11000 GRAPHICS 0:POKE 752,1:POSITION 1
,1:? "      GO ON TO BUILDING ";B:FOR
A=1 TO 3:FOR Q=1 TO 50:SOUND 0,Q,10,8
11005 NEXT Q:NEXT A:SOUND 0,0,0,0
11010 FOR A=0 TO SC STEP 150:SOUND 0,1
9,92,8:FOR Q=1 TO 20:NEXT Q:SOUND 0,0,
0,0:POSITION 10,5:? "SCORE:";A:NEXT A:
GOTO 5
31999 END
32000 POKE 106,PEEK(106)-5:GRAPHICS 0:
START=(PEEK(106)+1)*256:POKE 756,START
/256:POKE 752,1
32010 DIM XFR$(38):RESTORE 32015:FOR X
=1 TO 38:READ N:XFR$(X)=CHR$(N):NEXT X
32015 DATA 104,169,0,133,203,133,205,1
69,224,133,206,165,106,24,105,1,133,20
4,160,0,177,205,145,203,200
32016 DATA 208,249,230,204,230,206,165
,206,201,228,208,237,96
32020 Z=USR(ADR(XFR$)):RESTORE 32100
32030 READ X:IF X=-1 THEN RESTORE :RET
URN
32040 FOR Y=0 TO 7:READ Z:POKE X+Y+STA
RT,Z:NEXT Y:GOTO 32030
32100 DATA 264,60,126,219,255,231,189,
195,126
32101 DATA 272,106,50,36,255,126,126,1
26,126
32102 DATA 280,30,207,255,255,127,15,3
0,60
32103 DATA 288,30,26,255,254,224,0,0,0
32104 DATA 296,255,129,129,129,129,129
,129,255
32105 DATA 304,0,126,126,126,126,126,1
26,0
32106 DATA 312,195,153,153,231,60,60,6
0,0
32107 DATA 320,60,60,36,36,36,231,231,
0
32108 DATA 328,60,36,60,8,24,16,24,8
32109 DATA 336,60,90,126,129,165,129,1
26,60
32110 DATA 344,60,102,165,165,165,219,
60,231
32111 DATA 352,126,102,102,126,8,8,40,
56
32112 DATA 360,0,0,255,102,255,0,0,0
32113 DATA -1
32500 GRAPHICS 17
32510 FOR A=1 TO 22:POSITION 0,A:? #6;
"0*0*0*0*0*0*0*0*0*0*0*0*":SOUND 0,A+25,10
,12:NEXT A:POSITION 4,10
32511 SOUND 0,0,0,0
32512 ? #6;"..stunt.man.."
32520 POSITION 2,13:? #6;"by STEVEN PO
GATCH":POSITION 5,14:? #6;"PRESS START
"
32525 POSITION 7,18:? #6;"(c)1982"
32530 IF PEEK(53279)=6 THEN RETURN
32540 GOTO 32530
32700 GRAPHICS 18:POSITION 1,2:? #6;"G
AME  OVER":POSITION 1,5:? #6;"SCORE=";
SC
32710 POSITION 1,6:? #6;"PRESS START "
32715 IF PEEK(53279)<>6 THEN 32715
32755 CLR :GOTO 2
```

CHECKSUM DATA

(See pgs. 7-10)

```
1 DATA 405,17,266,468,407,384,263,358,
118,164,663,967,893,549,529,6451
49 DATA 998,821,816,783,492,194,510,48
5,459,615,525,997,243,785,81,8804
3010 DATA 6,681,786,5,59,187,45,657,80
1,289,454,979,791,45,102,5887
5015 DATA 475,635,855,276,239,438,357,
196,192,201,498,693,954,593,795,7397
5200 DATA 694,964,594,796,697,983,597,
799,700,957,600,802,802,620,636,11241
6010 DATA 517,459,374,919,585,488,737,
366,796,650,292,498,329,393,275,7678
11010 DATA 684,592,284,910,654,591,768
,663,960,879,634,535,45,170,526,8895
32106 DATA 354,269,972,836,628,478,887
,833,312,711,575,597,203,329,343,8327
32540 DATA 224,776,906,306,215,2427
```

DUNGEONS & DRAGONS™ CHARACTER GENERATOR

24K Cassette 32K Disk

by Bob Curtin

When I first bought my ATARI, one of the things I put high on my list of priorities was to try one of the computer adventure games on the market. I wasn't impressed with the game, but I was impressed with the ease of play. Pressing a few buttons took care of movement, combat, encumbrance, game time and all the rest, and it dawned on me that my computer could be a big help to me in my ongoing DUNGEONS & DRAGONS campaign. I set to work writing a series of utility programs for it. This, the first, generates both player and non-player characters in an average of about four minutes. Normally, it takes anywhere from twenty to forty minutes to generate a character "by hand," and then there's a strong possibility of missing a few modifiers along the way. The computer always remembers.

Though the program was written to take the work out of generating characters, the Dungeon Master and players are still left with choices to make. As in D&D, the player still has choice of name, gender, race, class, and character level. Those categories greatly affect the final character statistics, and it would be an injustice to randomly choose them for the player. By the same token, there are certain minimum ability scores, or racial requirements, which must be met to assume the role of a particular race or class. The user doesn't have to know or worry about it; the computer will figure it all out and tell the player if he or she doesn't measure up. The player may continue to choose alternatives until one of his choices meets all requirements. The program will then continue on.

The system used is based on the standard ADVANCED DUNGEONS & DRAGONS game. There is an omission, however — by choice, not error. I didn't incorporate the maximum level restrictions imposed on certain races, such as an Elf being able to rise no higher than 7th level as a fighter. If a Dungeon Master wants to adhere to those limits, it's a simple matter to just look it up; while it's not so simple to get the computer to do what it's told not to. For those of you who want to ignore the limits, the computer doesn't know any better. Indulge.

I fudged a couple of other values, too. For instance, line 195 contains the random number generator for the characters' basic abilities. Notice that variables A and C have a +2 for the add-on number. I did this to give the players a break. All you hardline Dungeon Masters out there gnashing your teeth can switch back to +1 if you want. (Essentially, they're now rolling 3D6+2.)

The program.

As I said, there are five inputs. They are, in order: name, gender, race, class and character level. Here is an example of each.

Name — after the title, the computer will ask for a character name. This is the only "open" input, and — although you have to work at it — it can be screwed up. For example, entering a couple of control characters through the escape key will cause some grief later on down the line. Other than that, anything but an input of YES, NO, Y or N will be taken as the character name. If you don't want a name, just hit the return key. Entering NO or N will fetch a list of names from memory as suggestions to the player.

GENDER — The computer will only accept M or F. Lower case letters will not work.

RACE and CLASS — Only the exact initials listed in parentheses on the respective menus will be accepted.

CHARACTER LEVEL — Any level between 1 and 18 (inclusive) will be accepted. If a value below 1 is entered, the value will be upped to 1. If a value over 18 is entered, a short message will be displayed and the program will loop back for another input. Any illegal entry, such as a letter instead of a number, will also cause the loop back for re-entry.

As the character builds, the computer does the appropriate calculations, comparisons and modifications between inputs, and then displays the results. After the information has been copied from the screen, the player may continue the program by pressing any key.

Program outline.

Lines 5-26 — Initialization

Lines 50-75 — Character Race Modifier Routine

Lines 80-82 — Custom Display List

Lines 86-88 — Title (so, my vanity's showing). This can be deleted by eliminating lines 80 through 96 and changing the last statement in line 20 to GOTO 100.

Lines 100-111 — Thief, Magic-user, and Monk Data

Lines 159-179 — Name Input

Lines 180-187 — Gender Input

Lines 190-192 — Race Menu

Line 195 — Basic Ability Scores

Lines 200-225 — Race Input

Lines 226-229 — Ability Score Display

Lines 235-243 — Class Menu and Class Input

Lines 245-254 — Class Trigger and Gold Piece Generator

Lines 263-269 — Exceptional Strength Routine

Lines 276-332 — Hit, Damage, Armor Class, and Dexterity Modifiers

Lines 335-341 — Modifier Display

Lines 345-374 — Height and Weight Routine (modified by race and gender)

Lines 375-438 — Hit Point Generation Routine (modified by race and ability)

Lines 460-475 — Thieves Ability

Lines 500-530 — Magic-user Abilities

Lines 550-599 — Monk Abilities

Lines 2000-2020 — Name List

Lines 2550-2730 — Race Limitations

Lines 5000-5975 — Class Limitations

Lines 6132-6200 — Thief Abilities Modifiers (by race and ability scores)

Lines 7000-7055 — Psionics Routine

Lines 8000-8020 — Input Error Routine

A few suggestions for the DM.

Never lose sight of the fact that the only reason a player will participate in one of your D&D sessions is to have FUN! Nothing will dampen the enthusiasm of a new player faster than being forced to assume the role of a character too weak to take any kind of initiative, do any exploring, or even stand fast with the rest of the party. Force your players into a position of constant impotence and you'll soon find your dungeon devoid of adventurers.

Although I'm certainly not in favor of the give-away dungeon, killer dungeons are, if not worse, at least as bad. Surviving and advancing up the ladder of experience — *developing* a character is what D&D is all about. To have a developed character snuffed out by the undetectable, unseeable or unknowable is bound to cause you to gain a reputation as a "cheap shot" dungeon master. Having a character killed because of one's own recklessness or bad luck or *a bad choice between alternatives* can be lived with. But the skewering of some hapless player for no rhyme or reason is unforgivable.

Give your players a break. Pick a number — I use five — and let each player run off that many characters. The player can then choose one of them to start the game with, and should that character come to an untimely end, there are four more from which to choose. That way, no more valuable playing time is taken up generating characters.

Normally, novice players start at level one. However, after a player has campaigned for some time, it's usually the practice to let him or her start higher than that. If they have a character killed off, you could, for instance, have them start a couple of levels lower than the character who was killed. Another way is to roll a six or eight-sided die.

Above all, be fair. Remember that you, and consequently all of the creatures you control, have perfect intelligence. Your players do not; they only know what you tell them. It behooves you to give that little extra. If a player can't see something, don't wait for him to ask; tell him.

Good luck. Good dungeoning. □

```
5 TRAP 8000
10 DIM N$(40),Z$(30),R$(10),P$(10),E$(
20),DW$(20),GN$(20),HE$(22),ST$(9),WI$
(7),IN$(20),DX$(10),CN$(20),CH$(10)
12 DIM HA$(22),HO$(22),B$(10),Y$(19),T
(19,8),MU(20,9),F(6),J(15),G$(10),X(10
),M(33),MK(17,4),M$(34),D$(10)
15 Z$=" DOES NOT HAVE ENOUGH":ST$="STR
ENGTH":IN$="INTELLIGENCE":WI$="WISDOM"
:DX$="DEXTERITY":CN$="CONSTITUTION"
18 CH$="CHARISMA":B$=" TO BE A":E$="EL
VES CANNOT BE ":DW$="DWARVES CANNOT BE
 ":GN$="GNOMES CANNOT BE "
20 HE$="HALF-ELVES CANNOT BE ":HA$="HA
LFLINGS CANNOT BE ":HO$="HALF-ORCS CAN
NOT BE ":Y$=" NO.ATTACKS"
25 K1=1:K2=K1+K1:K3=K1+K2:K4=K1+K3:K5=
K1+K4:K6=K3+K3:K7=K4+K3:K8=K2+K6:K9=K1
+K8:K10=K9+K1:K11=25:K12=50:K13=100
26 K14=75:K15=125:K16=150:K17=200:K241
=241:K0=K1-K1:GOTO 80
50 FOR E=K1 TO K6:J(E)=F(E):NEXT E:O=0
:IF R$="H" THEN Y=K1:O=K5:RETURN
54 IF R$="E" THEN Y=K2:O=K5:J(K4)=J(K4
)+K1:J(K5)=J(K5)-K1:RETURN
56 IF R$="D" THEN Y=K3:O=K5:J(K5)=J(K5
)+K1:J(K6)=J(K6)-K1:RETURN
58 IF R$="G" THEN Y=K4:O=K5:RETURN
60 IF R$="HE" THEN Y=K5:O=K5:GOTO 75
62 IF R$="HA" THEN Y=K6:O=K5:J(K1)=J(K
1)-K1:J(K4)=J(K4)+K1:RETURN
64 IF R$="HO" THEN Y=K7:O=K5:J(K1)=J(K
1)+K1:J(K5)=J(K5)+K1:J(K6)=J(K6)-K2:RE
TURN
```

```
75 RETURN
80 POKE 712,128:? "{CLEAR}":DL=PEEK(560)+256
*PEEK(561):POKE 752,K1:POKE 559,K0
81 Z1=PEEK(DL+K4):Z2=PEEK(DL+K5):POKE
DL+K3,71:POKE DL+K4,Z1:POKE DL+K5,Z2:P
OKE DL+K6,K7:POKE DL+K7,K6
82 POKE DL+K8,K6:POKE DL+K9,K6:POKE DL
+K10,K6:POKE DL+K11+K2,65:POKE DL+K11+
K3,PEEK(560):POKE DL+29,PEEK(561)
88 POKE 82,0:POKE 559,34:POKE 710,128:
? "{CLEAR} DUNGEONS & DRAGONS":? "  RANDOM C
HARACTER":? " GENERATION PROGRAM"
92 ? "                BY BOB CURTIN"
93 ? :? :? "   THIS PROGRAM WAS WRITTE
N TO TAKE":? "      SOME OF THE BURDEN
OFF OF THE"
94 ? "    USUALLY HARRIED DUNGEON MAST
ER."
95 ? :? "    PLEASE BE SURE TO PRESS R
ETURN    ":? "          AFTER EACH INPU
T.":?
96 ? :? :? "     GOOD LUCK!  GOOD DUNG
EONING!":FOR E=K1 TO K10^3*5:NEXT E
100 FOR I=K1 TO K8:FOR X=K1 TO K10+K8:
READ N:T(X,I)=N:NEXT X:NEXT I
101 FOR I=K1 TO K3:FOR X=K9 TO K10+K9:
READ N:MU(X,I)=N:NEXT X:NEXT I
102 FOR I=K1 TO K4:FOR X=K1 TO K10+K7:
READ N:MK(X,I)=N:NEXT X:NEXT I
103 DATA 30,35,40,45,50,55,60,65,70,80
,90,100,105,110,115,125,125,125,25,29,
33,37,42,47,52,57,62,67,72,77,82,87
104 DATA 92,97,99,99,20,25,30,35,40,45
,50,55,60,65,70,75,80,85,90,95,99,99,1
5,21,27,33,40,47,55,62,70,78,86,94,99
105 DATA 99,99,99,99,99,10,15,20,25,31
,37,43,49,56,63,70,77,85,93,99,99,99,9
9,10,10,15,15,20,20,25,25,30,30,35,35
106 DATA 40,40,50,50,55,55,85,86,87,88
,90,92,94,96,98,99,99.1,99.2,99.3,99.4
,99.5,99.6,99.7,99.8,0,0,0,20,25,30
107 DATA 35,40,45,50,55,60,65,70,75,80
,80,80,35,45,45,45,55,55,65,65,75,85,9
5,4,5,5,5,6,6,7,7,8,9,10,6,7,7,7,9,9
108 DATA 11,11,14,18,99
109 DATA 10,9,8,7,7,6,5,4,3,3,2,1,0,-1
,-1,-2,-3,150,160,170,180,190,200,210,
220,230,240,250,260,270,280,290,300
110 DATA 320,1,1,1,54,54,32,32,32,2,2,
52,52,52,3,3,4,4,13,14,16,16,27,28,39,
212,312,313,413,416,517,520,624
111 DATA 530,832
159 POKE 82,2:GRAPHICS 1:POKE 752,1:PO
KE 712,128:POKE 710,128
160 RESTORE :? #6:? #6:? #6;" DUNGEONS
 & DRAGONS":? #6;"CHARACTER GENERATION
"
170 ? "HAVE YOU THOUGHT OF A NAME":? "
FOR YOUR CHARACTER";:INPUT N$
175 IF N$="YES" OR N$="Y" THEN ? "{CLEAR}WEL
L, WHAT IS IT";:INPUT N$
179 IF N$="NO" OR N$="N" THEN GRAPHICS
 0:POKE 710,6:POKE 709,0:POKE 752,1:GO
SUB 2000
180 ? "{CLEAR}WHAT GENDER IS ";N$;" (M/F)";:
INPUT G$:O=0:IF G$="M" OR G$="F" THEN
O=K5
187 IF O<>K5 THEN ? "{CLEAR}M/F ONLY, PLEASE
!":FOR E=K1 TO 1500:NEXT E:GOTO 180
190 ? #6:? #6:? #6;"   HUMAN       (H)"
:? #6;"   ELF         (E)":? #6;"   DWA
RF        (D)"
192 ? #6;"   GNOME       (G)":? #6;"
HALFLING   (HA)":? #6;"   HALF-ELF  (HE
)":? #6;"   HALF-ORC   (HO)"
195 FOR E=K1 TO K6:A=INT(K6*RND(K1)+K2
):B=INT(K6*RND(K1)+1):C=INT(K6*RND(K1)
+K2):D=A+B+C:F(E)=D:NEXT E:GOTO 205
200 POP :?
205 ? "WHAT RACE";:INPUT R$:GOSUB K12
210 IF O<>K5 THEN ? "{CLEAR}INITIALS ONLY, P
LEASE!":GOTO 205
215 GOSUB 2550
220 FOR E=K1 TO K6:IF J(E)>K9+K9 THEN
J(E)=K9+K9
224 IF J(E)<K3 THEN J(E)=K3
225 F(E)=J(E):NEXT E
226 GRAPHICS K1:POKE 712,50:POKE 710,5
0:? #6:? #6:? #6
227 ? #6:? #6:? #6:? #6;"  STRENGTH
   ";F(K1):? #6;"  INTELLIGENCE  ";F(K
2):? #6;"  WISDOM        ";F(K3)
228 ? #6;"  DEXTERITY     ";F(K4):? #6
;"  CONSTITUTION  ";F(K5):? #6;"  CHAR
ISMA       ";F(K6):? #6:? #6
229 ? #6;"  BASIC ABILITIES":POKE 752,
1:? "     PRESS ANY KEY TO CONTINUE"
230 OPEN #1,4,0,"K:":GET #1,E:CLOSE #1
:IF E>0 THEN 235
235 GRAPHICS 1:POKE 709,96:POKE 710,16
8:POKE 712,98:POKE 752,1
236 ? #6:? #6:? #6
237 ? #6;"  FIGHTER       (F)":? #6;"
PALADIN      (P)":? #6;"  RANGER
 (R)":? #6;"  CLERIC      (C)"
238 ? #6;"  DRUID         (D)":? #6;"
MONK         (M)":? #6;"  THIEF
 (I)":? #6;"  ASSASSIN     (A)"
239 ? #6;"  MAGIC-USER  (MU)":? #6;"
ILLUSIONIST  (I)":? #6:? #6:? #6;"  CH
OOSE FROM":? #6;"  THE ABOVE LIST"
240 GOTO 243
241 POP :?
243 Z=K0:O=K0:E5=K0:? "WHAT CLASS";:IN
PUT P$
245 IF P$="F" THEN O=K5:Z=K1:GP=INT(15
0*RND(1)+50)
246 IF P$="R" THEN O=K5:Z=K2:GP=INT(K1
6*RND(K1)+50)
247 IF P$="P" THEN O=K5:Z=K3:GP=INT(K1
6*RND(1)+50)
248 IF P$="C" THEN O=K5:Z=K4:GP=INT(K1
6*RND(1)+30)
249 IF P$="D" THEN O=K5:Z=K5:GP=INT(K1
6*RND(1)+30)
250 IF P$="T" THEN O=K5:Z=K6:GP=INT(K1
3*RND(1)+20)
251 IF P$="A" THEN O=K5:Z=K7:GP=INT(K1
3*RND(1)+20)
252 IF P$="MU" THEN O=K5:Z=K8:GP=INT(6
0*RND(1)+20)
253 IF P$="I" THEN O=K5:Z=K9:GP=INT(60
*RND(1)+20)
254 IF P$="M" THEN O=K5:Z=K10:GP=INT(1
5*RND(1)+5)
255 IF O<>K5 THEN ? "CORRECT INITIALS
ONLY, PLEASE!":? :GOTO 243
262 GOSUB 5000
263 IF P$="F" OR P$="R" OR P$="P" THEN
 IF F(K1)=K10+K8 THEN 265
264 GOTO 276
265 GRAPHICS 2+16:POKE 711,4:? #K6:? #
K6:? #K6:? #K6;" ";N$;" HAS ":? #K6;"
EXCEPTIONAL":? #K6;" STRENGTH"
269 ? #6:E5=INT(K13*RND(K1)+K1):? #6;"
 E.S.RATING  18/";E5:FOR E=K1 TO 2000:
NEXT E
276 MH=0:MD=0:MA=0:MR=0:K325=325:K335=
335
310 IF E5=K13 THEN MH=K3:MD=K6:GOTO K3
25
311 IF E5>=K13-K9 THEN MH=K2:MD=K5:GOT
O K325
312 IF E5>=3*K11+K1 THEN MH=K2:MD=K4:G
OTO K325
313 IF E5>=K12+K1 THEN MH=K2:MD=K3:GOT
O K325
314 IF E5>=K1 THEN MH=K1:MD=K3:GOTO K3
25
315 IF A=K9+K9 THEN MH=K1:MD=K2:GOTO K
325
316 IF A=K10+K7 THEN MH=K1:MD=K1:GOTO
K325
317 IF A=K10+K6 THEN MD=K1:GOTO K325
318 IF A=K3 THEN MH=-K3:MD=-K2:GOTO K3
25
319 IF A=K4 THEN MH=-K2:MD=-K2:GOTO K3
25
320 IF A<=K6 THEN MH=-K1:GOTO K325
325 IF D=K9+K9 THEN MR=K3:MA=-K4:GOTO
K335
326 IF D=K9+K8 THEN MR=K2:MA=-K3:GOTO
```

```
K335
327 IF D=K8+K8 THEN MR=K1:MA=-K2:GOTO
K335
328 IF D=K7+K8 THEN MA=-K1:GOTO K335
329 IF D=K6 THEN MA=K1:GOTO K335
330 IF D=K5 THEN MR=-K1:MA=K2:GOTO K33
5
331 IF D=K4 THEN MR=-K2:MA=K3:GOTO K33
5
332 IF D=K3 THEN MR=-K3:MA=K4:GOTO K33
5
335 GRAPHICS 1:POKE 712,128:POKE 708,2
2:POKE 709,22:POKE 752,1:POKE 710,128
336 ? #6:? #6;" ";N$;"'S"
337 ? #6;" MODIFIERS:":? #6
338 ? #6;" HIT            ";MH
339 ? #6;" DAMAGE         ";MD
340 ? #6;" A/C ADJUSTMENT ";MA
341 ? #6;" R/A BONUS      ";MR
345 X(5)=INT(7*RND(1)):X(6)=INT(9*RND(
1)):X(7)=INT(11*RND(1)):X(8)=INT(13*RN
D(1)):X(9)=INT(25*RND(1))
346 X(6)=INT(9*RND(1))
350 M(5)=INT(40*RND(1)):M(6)=INT(30*RN
D(1)):M(7)=INT(20*RND(1)):M(8)=INT(50*
RND(1)):M(9)=INT(66*RND(1))
355 IF G$="F" THEN 365
356 IF Y=K3 THEN H=K2*K11-K6+X(K7):W=K
13+K11+K9+M(K5)
357 IF Y=K2 THEN H=K12+K6+X(K7):W=K13-
K10+M(K6)
358 IF Y=K4 THEN H=K12-K10-K1+X(K5):W=
K3*K11-K3+M(K7)
359 IF Y=K5 THEN H=K12+K10+X(K8):W=110
+M(K5)
360 IF Y=K7 THEN H=K12+K10+K2+X(K6):W=
K16+M(K8)
361 IF Y=K6 THEN H=K11+K10+K1+X(K6):W=
80+M(K5)
362 IF Y=K1 THEN H=K12+K10+X(K9):W=K13
+K11+K5+M(K9)
363 GOTO 372
365 IF Y=K3 THEN H=42+X(K6):W=K14+K4+M
(K6)
366 IF Y=K2 THEN H=K12+X(K7):W=K13-K5+
M(K7)
367 IF Y=K4 THEN H=K6*K6+X(K5):W=K6*K1
0+K7+M(K7)
368 IF Y=K5 THEN H=K12+K6+X(K8):W=K8*K
10+M(K6)
369 IF Y=K7 THEN H=K12+K9+X(K5):W=K14+
K5+M(K8)
370 IF Y=6 THEN H=30+X(5):W=42+M(7)
371 IF Y=K1 THEN H=K12+K6+X(K4):W=K14+
M(K9)
372 Q1=INT(H/12):Q2=Q1*12:Q3=H-Q2
373 ? #6:? #6:? #6;" HEIGHT    ";Q1;"'"
;Q3;CHR$(34)
374 ? #6;" WEIGHT    ";W;"LBS."
375 HPT=K0:O=K0:GOTO 400
380 HP=INT(K4*RND(K1)+K2):RETURN
385 HP=INT(K6*RND(K1)+K2):RETURN
390 HP=INT(K8*RND(K1)+K2):RETURN
395 HP=INT(K10*RND(K1)+K2):RETURN
400 ? "WHAT LEVEL IS ";N$;:INPUT L:IF
Z=K2 THEN L=L+K1
406 IF L>18 THEN ? "YOU CAN'T START A
 CHARACTER":? "OVER LEVEL 18.  TRY AGA
IN.":? :GOTO 400
407 IF L<=0 THEN L=1
408 FOR J=K1 TO L:IF Z=K1 OR Z=K3 THEN
 GOSUB 395
410 IF Z=K2 OR Z=K4 OR Z=K5 THEN GOSUB
 390
415 IF Z=K6 OR Z=K7 THEN GOSUB 385
420 IF Z=K8 OR Z=K9 OR Z=K10 THEN GOSU
B 380
422 HPT=HPT+HP:NEXT J:GOTO 431
427 IF E=K9+K9 THEN HPT=HPT+(L*K4):GOT
O 438
428 IF E=K9+K8 THEN HPT=HPT+(L*K3):GOT
O 438
429 IF E=K8+K8 THEN HPT=HPT+(L*K2):GOT
O 438
430 GOTO 432
431 IF E>=K8+K8 THEN HPT=HPT+(L*K2):GO
TO 438
432 IF E=K9+K6 THEN HPT=HPT+L:GOTO 438
433 IF E=K3 THEN HPT=HPT-(L*K2):GOTO 4
38
434 IF E<K8 THEN HPT=HPT-L
438 ? #6:? #6;" HIT POINTS    ";HPT
440 IF Z=K1 OR Z=K3 THEN IF L>=12 THEN
 ? #6;Y$;"    2/1":GOTO 456
445 IF Z=K1 OR Z=K3 THEN IF L>=K6 THEN
 ? #6;Y$;"    3/2":GOTO 456
446 IF Z=K2 THEN IF L>=16 THEN ? #6;Y$
;"    2/1":GOTO 456
447 IF Z=K2 THEN IF L>=K7 THEN ? #6;Y$
;"    3/2"
456 IF Y=K1 OR Y=K3 OR Y=K6 THEN GOSUB
 7000
457 ? "R";N$;" HAS ";GP;" GOLD PIECES"
458 GOSUB 6130:? :? "   PRESS ANY KEY
TO CONTINUE"
459 OPEN #1,4,0,"K:":GET #1,I:CLOSE #1
:IF I>0 THEN 460
460 B$=INT(L/K4)+K2:IF Z=K6 OR Z=K7 TH
EN 462
461 GOTO 500
462 GRAPHICS 1+16
463 ? #6:? #6:? #6:? #6
465 ? #6;"    BS - - - - X";B$
466 ? #6;"    PP - - - - ";T(L,K1)
467 ? #6;"    LOCKS  - - ";T(L,K2)
468 ? #6;"    TRAPS  - - ";T(L,K3)
469 ? #6;"    MS - - - - ";T(L,K4)
470 ? #6;"    HS - - - - ";T(L,K5)
471 ? #6;"    HEAR - - - ";T(L,K6)
472 ? #6;"    CLIMB  - - ";T(L,K7)
473 ? #6;"    LANGUAGES  ";T(L,K8)
474 ? #6:? #6:? #6;" THIEVES ABILITIES
"
475 FOR I=K1 TO K10^3*K5:NEXT I
500 IF Z=K8 OR Z=K9 THEN 505
501 IF Z=K10 THEN 550
502 GOTO 4999
505 GRAPHICS 2+16:POKE 712,160
510 ? #6:? #6:? #6:? #6;" CHANCE TO KN
OW  ";MU(B,1)
515 ? #6:? #6;" MINIMUM SPELLS  ";MU(B
,2)
520 ? #6:? #6;" MAXIMUM SPELLS  ";MU(B
,3)
530 FOR I=K1 TO 4000:NEXT I
535 GOTO 4999
550 GRAPHICS 1+16:POKE 712,212:POKE 71
0,224
551 ? #6:? #6:? #6:? #6;"   MONKS TABL
E":? #6
552 ? #6:? #6;" ARMOR CLASS     ";MK(L
,1)
553 ? #6;" MOVE           ";MK(L,2);"'"
554 IF MK(L,K3)=K1 THEN M$="1"
555 IF MK(L,K3)=54 THEN M$="5/4"
556 IF MK(L,K3)=32 THEN M$="3/2"
557 IF MK(L,K3)=K2 THEN M$="2"
558 IF MK(L,K3)=52 THEN M$="5/2"
559 IF MK(L,K3)=K3 THEN M$="3"
560 IF MK(L,K3)=K4 THEN M$="4"
561 ? #6;" ATTACKS/ROUND ";M$
562 IF MK(L,K4)=13 THEN D$="1D3"
563 IF MK(L,K4)=14 THEN D$="1D4"
564 IF MK(L,K4)=16 THEN D$="1D6"
565 IF MK(L,K4)=27 THEN D$="1D6+1"
566 IF MK(L,K4)=28 THEN D$="2D4"
567 IF MK(L,K4)=39 THEN D$="3D3"
568 IF MK(L,K4)=212 THEN D$="2D6"
569 IF MK(L,K4)=312 THEN D$="3D4"
570 IF MK(L,K4)=413 THEN D$="3D4+1"
571 IF MK(L,K4)=416 THEN D$="4D4"
572 IF MK(L,K4)=517 THEN D$="4D4+1"
573 IF MK(L,K4)=520 THEN D$="5D4"
574 IF MK(L,K4)=624 THEN D$="6D4"
575 IF MK(L,K4)=530 THEN D$="5D6"
576 IF MK(L,K4)=832 THEN D$="4D8"
577 ? #6;" DAMAGE/ATTACK ";D$
578 ? #6:? #6;" SEE THE PLAYERS"
579 ? #6;" HANDBOOK FOR"
580 ? #6;" SPECIAL ABILITIES"
```

```
599 FOR I=K1 TO 5000:NEXT I
1999 GOTO 4999
2000 ? "↓↓IF YOU'RE HAVING TROUBLE PIC
KING";? "A NAME FOR YOUR CHARACTER, PE
RHAPS"
2005 ? "YOU'D LIKE A FEW SUGGESTIONS.
"
2010 ? "YOU'RE WELCOME TO USE ONE OF T
HESE:"
2015 ? "↓↓SETH THE HUGE","BUCKTHORN"
2016 ? "AARON THE SWIFT","ELLIDE"
2017 ? "BRIAN OF BLACKMOOR","JANO"
2018 ? "ALONSO THE HOOK","TAPHENESE"
2019 ? "SIR BAGLEY","BAAREN SATO"
2020 ? "↓↓IF YOU WANT ONE OF THESE, JU
ST";? "TYPE IN THE NAME AND PRESS RETU
RN."
2022 ? "↓IF YOU DON'T, TYPE 'NO' AND P
RESS";? "RETURN.";? "↓NAME";:INPUT N$:
IF N$="NO" OR N$="N" THEN N$="WHOOZIT"
2028 GRAPHICS 1:POKE 708,40:POKE 752,1
:RETURN
2550 A=J(K1):B=J(K2):C=J(K3):A1=J(K4):
B1=J(K5):C1=J(K6):? "R"
2555 ON Y-K1 GOTO 2600,2580,2630,2650,
2670,2700
2576 RETURN
2580 IF A<K8 THEN ? N$;Z$:? ST$;B$;" D
WARF.":GOTO K17
2585 IF B1<K6*K2 THEN ? N$;Z$:? CN$;B$
;" DWARF.":GOTO K17
2590 IF G$="F" THEN IF J(K1)>K9+K8 THE
N J(K1)=K10*K7
2595 IF J(K4)>K9+K8 THEN J(K4)=K9+K8
2597 IF J(K6)>K8+K8 THEN J(K6)=K8+K8
2599 RETURN
2600 IF B<K8 THEN ? N$;Z$:? IN$;B$;"N
ELF.":GOTO K17
2605 IF A1<K7 THEN ? N$;Z$:? DX$;B$;"N
 ELF.":GOTO K17
2610 IF B1<K6 THEN ? N$;Z$:? CN$;B$;"N
 ELF.":GOTO K17
2615 IF C1<K8 THEN ? N$;Z$:? CH$;B$;"N
 ELF.":GOTO K17
2620 IF G$="F" THEN IF J(K1)>K8+K8 THE
N J(K1)=K8+K8
2625 RETURN
2630 IF A<K6 THEN ? N$;Z$:? ST$;B$;" G
NOME.":GOTO K17
2635 IF B<K7 THEN ? N$;Z$:? IN$;B$;" G
NOME.":GOTO K17
2640 IF B1<K8 THEN ? N$;Z$:? CN$;B$;"
GNOME.":GOTO K17
2645 IF G$="F" THEN IF J(K1)>K3*K5 THE
N J(K1)=K3*K5
2648 RETURN
2650 IF B<K4 THEN ? N$;Z$:? IN$;B$;" H
ALF-ELF.":GOTO K17
2655 IF A1<K6 THEN ? N$;Z$:? DX$;B$;"
HALF-ELF.":GOTO K17
2660 IF B1<K6 THEN ? N$;Z$:? CN$;B$;"
HALF-ELF.":GOTO K17
2665 IF G$="F" THEN IF J(K1)>K9+K8 THE
N J(K1)=K9+K8
2668 RETURN
2670 IF A<K6 THEN ? N$;Z$:? ST$;B$;" H
ALFLING.":GOTO K17
2675 IF B<K6 THEN ? N$;Z$:? IN$;B$;" H
ALFLING.":GOTO K17
2680 IF A1<K8 THEN ? N$;Z$:? DX$;B$;"
HALFLING.":GOTO K17
2685 IF B1<K10 THEN ? N$;Z$:? CN$;B$;"
 HALFLING.":GOTO K17
2690 IF G$="M" THEN IF J(K1)>K9+K8 THE
N J(K1)=K9+K8
2694 IF G$="F" THEN IF J(K1)>K7+K7 THE
N J(K1)=K7+K7
2695 IF J(K3)>K9+K8 THEN J(K3)=K9+K8
2696 RETURN
2700 IF A<K6 THEN ? N$;Z$:? ST$;B$;" H
ALF-ORC.":GOTO K17
2705 IF B1<K6+K7 THEN ? N$;Z$:? CN$;B$
;" HALF-ORC.":GOTO K17
2710 IF J(K2)>K9+K9 THEN J(K2)=K9+K8
2715 IF J(K3)>K7+K7 THEN J(K3)=K7+K7
2720 IF J(K4)>K7+K7 THEN J(K4)=K7+K7
2725 IF J(K6)>K6+K6 THEN J(K6)=K6+K6
2730 RETURN
4999 GRAPHICS 1:SETCOLOR 2,L,4:POKE 75
2,1:SETCOLOR 4,L,4:GOTO 160
5000 A=F(K1):B=F(K2):C=F(K3):D=F(K4):E
=F(K5):F=F(K6):? "R"
5005 ON Z GOTO 5100,5200,5300,5400,550
0,5600,5700,5800,5900,5950
5055 RETURN
5100 IF A<K9 THEN ? N$;Z$:? ST$;B$;" F
IGHTER.":GOTO K241
5105 IF E<K7 THEN ? N$;Z$:? CN$;B$;" F
IGHTER.":GOTO K241
5110 RETURN
5200 IF A<K10+K3 THEN ? N$;Z$:? ST$;B$
;" RANGER.":GOTO K241
5205 IF B<K10+K3 THEN ? N$;Z$:? IN$;B$
;" RANGER.":GOTO K241
5210 IF C<K10+K4 THEN ? N$;Z$:? WI$;B$
;" RANGER.":GOTO K241
5215 IF E<K10+K4 THEN ? N$;Z$:? CN$;B$
;" RANGER.":GOTO K241
5220 IF Y=K3 THEN ? DW$;"RANGERS.":GOT
O K241
5225 IF Y=K2 THEN ? E$;"RANGERS.":GOTO
 K241
5230 IF Y=K4 THEN ? GN$;"RANGERS.":GOT
O K241
5235 IF Y=K6 THEN ? HA$;"RANGERS.":GOT
O K241
5240 IF Y=K7 THEN ? HO$;"RANGERS.":GOT
O K241
5245 RETURN
5300 IF A<K10+K2 THEN ? N$;Z$:? ST$;B$
;" PALADIN.":GOTO K241
5305 IF B<K9 THEN ? N$;Z$:? IN$;B$;" P
ALADIN.":GOTO K241
5310 IF C<K10+K3 THEN ? N$;Z$:? WI$;B$
;" PALADIN.":GOTO K241
5315 IF E<K9 THEN ? N$;Z$:? CN$;B$;" P
ALADIN.":GOTO K241
5320 IF F<K9+K8 THEN ? N$;Z$:? CH$;B$;
" PALADIN.":GOTO K241
5325 IF Y<>K1 THEN ? "ONLY HUMANS CAN
BE PALADINS.":GOTO K241
5330 RETURN
5400 IF C<K9 THEN ? N$;Z$:? WI$;B$;" C
LERIC.":GOTO K241
5405 IF Y=K6 THEN ? HA$;"CLERICS.":GOT
O K241
5410 RETURN
5500 IF C<K10+K2 THEN ? N$;Z$:? WI$;B$
;" DRUID.":GOTO K241
5505 IF F<K10+K5 THEN ? N$;Z$:? CH$;B$
;" DRUID.":GOTO K241
5510 IF Y=K3 THEN ? DW$;"DRUIDS.":GOTO
 K241
5515 IF Y=K2 THEN ? E$;"DRUIDS.":GOTO
K241
5520 IF Y=K4 THEN ? GN$;"DRUIDS.":GOTO
 K241
5525 IF Y=K7 THEN ? HO$;"DRUIDS.":GOTO
 K241
5530 RETURN
5600 IF D<K9 THEN ? N$;Z$:? DX$;B$;" T
HIEF.":GOTO K241
5605 RETURN
5700 IF A<K10+K2 THEN ? N$;Z$:? ST$;B$
;"N ASSASSIN.":GOTO K241
5705 IF B<K10+K1 THEN ? N$;Z$:? IN$;B$
;"N ASSASSIN.":GOTO K241
5710 IF D<K10+K2 THEN ? N$;Z$:? DX$;B$
;"N ASSASSIN.":GOTO K241
5720 IF Y=K6 THEN ? HA$;"ASSASSINS.":G
OTO K241
5725 RETURN
5800 IF B<K9 THEN ? N$;Z$:? IN$;B$;" M
AGIC-USER.":GOTO K241
5805 IF D<K6 THEN ? N$;Z$:? DX$;B$;" M
AGIC-USER.":GOTO K241
5810 IF Y=K3 THEN ? DW$;"MAGIC-USERS."
:GOTO K241
5815 IF Y=K4 THEN ? GN$;"MAGIC-USERS."
:GOTO K241
5820 IF Y=K7 THEN ? HO$;"MAGIC-USERS."
:GOTO K241
```

```
5825 IF Y=K6 THEN ? HA$;"MAGIC-USERS."
:GOTO K241
5830 RETURN
5900 IF B<K10+K5 THEN ? N$;Z$:? IN$;B$
;"N ILLUSIONIST.":GOTO K241
5905 IF D<K10+K6 THEN ? N$;Z$:? DX$;B$
;"N ILLUSIONIST.":GOTO K241
5910 IF Y=K3 THEN ? DW$;"ILLUSIONISTS.
":GOTO K241
5915 IF Y=K2 THEN ? E$;"ILLUSIONISTS."
:GOTO K241
5920 IF Y=K7 THEN ? HO$;"ILLUSIONISTS.
":GOTO K241
5925 IF Y=K6 THEN ? HA$;"ILLUSIONISTS.
":GOTO K241
5930 RETURN
5950 IF A<K10+K5 THEN ? N$;Z$:? ST$;B$
;" MONK.":GOTO K241
5955 IF C<K10+K5 THEN ? N$;Z$:? WI$;B$
;" MONK.":GOTO K241
5960 IF D<K10+K1 THEN ? N$;Z$:? DX$;B$
;" MONK.":GOTO K241
5965 IF E<K10+K1 THEN ? N$;Z$:? CN$;B$
;" MONK.":GOTO K241
5970 IF Y<>K1 THEN ? "ONLY HUMANS CAN
BE MONKS.":GOTO K241
5975 RETURN
6130 IF D=18 THEN T(L,K1)=T(L,K1)+K10:
T(L,K2)=T(L,K2)+15:T(L,K3)=T(L,K3)+K5:
T(L,K4)=T(L,K4)+10:T(L,K5)=T(L,K5)+10
6131 IF D=K10+K7 THEN T(L,K1)=T(L,K1)+
K5:T(L,K2)=T(L,K2)+K10:T(L,K4)=T(L,K4)
+K5:T(L,K5)=T(L,K5)+K5
6132 IF D=K10+K6 THEN T(L,K2)=T(L,K2)+
K5
6133 IF D=K10+K2 THEN T(L,K4)=T(L,K4)-
K5
6134 IF D=K10+K1 THEN T(L,K1)=T(L,K1)-
K5:T(L,K3)=T(L,K3)-K5:T(L,K4)=T(L,K4)-
K10
6135 IF Y=K3 THEN T(L,K2)=T(L,K2)+K10:
T(L,K3)=T(L,K3)+15:T(L,K7)=T(L,K7)-K10
:T(L,K8)=T(L,K8)-K5
6136 IF Y=K2 THEN T(L,K1)=T(L,K1)+K5:T
(L,K2)=T(L,K2)-K5:T(L,K4)=T(L,K4)+K5:T
(L,K5)=T(L,K5)+K10:T(L,K6)=T(L,K6)+K5
6137 IF Y=K4 THEN T(L,K2)=T(L,K2)+K5:T
(L,K3)=T(L,K3)+K10:T(L,K4)=T(L,K4)+K5:
T(L,K5)=T(L,K5)+K5:T(L,K6)=T(L,K6)+10
6138 IF Y=K4 THEN T(L,K7)=T(L,K7)-K15
6139 IF Y=K5 THEN T(L,K1)=T(L,K1)+K10:
T(L,K5)=T(L,K5)+K5
6140 IF Y=K6 THEN T(L,K1)=T(L,K1)+K5:T
(L,K2)=T(L,K2)+K5:T(L,K3)=T(L,K3)+K5:T
(L,K4)=T(L,K4)+K10
6141 IF Y=K6 THEN T(L,K5)=T(L,K5)+K10+
K5:T(L,K6)=T(L,6)+K5:T(L,K7)=T(L,K7)-K
10+K5:T(L,K8)=T(L,K8)-K5
6142 IF Y=K7 THEN T(L,K1)=T(L,K1)-K5:T
(L,K2)=T(L,K2)+K5:T(L,K3)=T(L,K3)+K5
6143 IF Y=K7 THEN T(L,K6)=T(L,K6)+K5:T
(L,K7)=T(L,K7)+K5:T(L,K8)=T(L,K8)-K10
6200 RETURN
7000 AI=INT(2.5*B-16):AW=INT(1.5*C-16)
:AC=INT(0.5*F-16)
7001 IF AI<0 THEN AI=0
7002 IF AW<0 THEN AW=0
7003 IF AC<0 THEN AC=0
7004 AT=AI+AW+AC
7005 PS=INT(K13*RND(K1)+AT+1):IF PS>=K
13 THEN ? #6:? #6;" ";N$;" HAS":? #6;"
 PSIONIC ABILITY"
7010 AI=B-12:AW=C-12:AC=F-12:IF AI<0 T
HEN AI=K0
7011 IF AW<K0 THEN AW=K0
7012 IF AC<K0 THEN AC=K0
7013 AT=AI+AW+AC
7015 MP=K0:OT=K0:IF B>16 THEN OT=OT+K1
7020 IF C>16 THEN OT=OT+K1
7025 IF F>16 THEN OT=OT+K1
7030 IF OT=K2 THEN MP=K2
7035 IF OT=K3 THEN MP=K4
7040 PST=INT(K13*RND(K1)+K1)+AT*MP
7045 IF PS>=K13 THEN ? "%PSIONIC ABILI
TY = ";PST*K2
7050 IF PS>=K13 THEN ? "PSIONIC STRENG
TH = ";PST:FOR I=1 TO 2000:NEXT I
7055 RETURN
8000 ERLN=256*PEEK(187)+PEEK(186)
8010 CUR=PEEK(90):? "%":? "INPUT ERROR
 -- TRY AGAIN!":FOR I=1 TO 50:SOUND 0,
I+50,10,8:NEXT I:SOUND 0,0,0,0
8020 TRAP 8000:GOTO (ERLN)
```

•

CHECKSUM DATA

(See pgs. 7-10)

```
5 DATA 400,646,294,906,573,733,840,75,
164,290,307,670,642,162,878,7580
75 DATA 779,953,859,964,820,531,652,63
8,208,981,850,829,789,454,917,11224
105 DATA 903,725,877,652,209,722,201,7
01,592,73,764,366,606,497,591,8479
192 DATA 58,773,366,178,279,815,74,452
,522,485,817,848,185,465,554,6871
236 DATA 17,249,288,686,714,379,685,3,
317,65,53,57,65,49,159,3786
253 DATA 226,160,424,821,503,736,291,8
0,10,461,559,78,528,442,207,5526
316 DATA 446,660,381,382,729,393,386,3
79,603,93,33,37,41,107,809,5479
337 DATA 299,54,232,810,475,878,10,256
,793,350,626,276,254,643,782,6738
362 DATA 281,735,415,515,209,881,846,1
72,507,122,333,644,301,424,433,6818
390 DATA 435,617,205,269,511,269,567,6
76,691,234,736,733,730,715,605,7993
432 DATA 984,355,534,141,605,697,631,5
02,807,705,217,512,468,716,216,8090
463 DATA 441,2,498,674,716,504,486,653
,658,959,331,418,835,807,4,7986
505 DATA 780,272,560,572,448,16,452,84
0,621,800,806,820,803,813,817,9420
559 DATA 819,817,582,799,803,810,269,8
16,822,996,995,48,2,62,997,9637
574 DATA 10,6,25,452,399,156,871,478,8
13,773,57,674,653,290,427,6084
2018 DATA 800,287,536,605,93,285,185,8
15,280,630,207,394,403,823,421,6764
2605 DATA 76,45,48,13,811,288,264,342,
996,819,628,703,674,30,821,6558
2670 DATA 713,688,778,379,37,22,393,82
0,698,25,376,379,374,386,805,6873
4999 DATA 303,943,27,802,770,728,791,8
26,815,813,815,540,635,527,514,9849
5240 DATA 534,807,928,742,915,728,81,9
55,799,880,472,800,951,948,634,11174
5515 DATA 236,621,637,805,451,815,412,
399,409,46,820,183,187,80,76,6177
5820 DATA 73,54,814,924,953,628,256,62
1,602,817,565,559,554,544,576,8540
5975 DATA 831,197,387,875,881,473,366,
480,328,827,796,449,131,857,108,7986
6200 DATA 795,648,7,79,981,290,152,703
,56,998,289,913,365,378,102,6756
7035 DATA 116,555,697,952,806,33,188,2
50,3597
```

•

DUNGEONS & DRAGONS™ HOUSEKEEPING 2

32K Cassette or Disk

by Bob Curtin

With the addition of the random access capability of a disk drive, **Dungeons & Dragons Housekeeping** (disk version) comes into its own, with several new functions, bilateral combat, and a drastically cleaned-up act. The following is a detailed rundown of each of the program functions and, where applicable, some suggestions on their use. This program is quite flexible and, with a little imagination, can make your job as a Dungeon Master a whole lot easier.

Incidentally, when I use the term "monster," I'm referring to all non player-character creatures, whether human, drooling beast, or anything in between. Players and player-characters are, of course, the people for whom you're running the dungeon, and the characters they're playing.

I do have one word of caution. If, when you're typing in this program, you're rewarded with an ERROR 4 (too many variables), don't panic. First, find your typing error, change it, and then follow the procedure outlined in pages 2 and 3 of your BASIC Reference Manual for wiping out excess variable names. The reason for the problem in the first place is the fact that I've used all 128 variable names available, and if you inadvertently add one of your own, you'll get that error.

M - MELEE (combat)

Where the cassette version of **D&D Housekeeping** handled the combat for the player characters only and left the monster combat to be done manually, the disk version does it all.

The MELEE function works very closely with the ENTER ROOM and LOAD ROOM functions. Those two functions make the monster data available for combat purposes as needed. The player character data is always contained in memory, but the monster data is loaded as encountered, either from disk or on the fly. The computer asks only which opponent is to be fought. The players, of course, make that choice for themselves, and you as the Dungeon Master make that choice for the monsters. As in a normal dungeon, you'd have some sort of graphic representation of the battle set up on the table top — usually with miniature figures.

In order, the computer makes the following computations: The attacker's class and level are looked up and the appropriate combat table consulted to obtain a "to hit" number against the opponent's armor class. A random number is generated and any hit modifiers added. If a hit is obtained, damage is "rolled" based on the weapon used, and then any damage modifiers are added on. Hit points lost are automatically deducted. The computer then checks to see if the attacker is entitled to more than one attack this round, either because of weapon type or level, and if so, repeats the procedure. Otherwise play passes to the next combatant.

The combat alternates back and forth between the players and monsters, that is, all players make their attacks, then all monsters make theirs, etc. Killed or unconscious combatants are automatically removed from the cycle, and combat can be broken off at any time, either individually or en masse.

Each player character may have up to five weapons, but weapon number 5 is reserved for missile weapons. As such, it's the only weapon modified by the R/A Bonus, and conversely not modified by normal hit and damage modifiers. Weapon number 5 may also be assigned multiple hits per round. For instance, D&D rules allow a player with a short bow to fire two arrows per combat round. It's conceivable for a player with multiple rounds of combat to fire up to six arrows in a turn!

By entering '33' as an opponent number and pressing RETURN, spell damage may be inflicted on the monster of your choice. If the spell affects more than one monster, you can repeat the procedure as many times as desired by pressing 'S'. To pass play to the next player, just press any other key. You may add hit points to the monster of your choice by en-

tering the amount of hit points you wish to add in the form of a minus number. For instance, suppose the party were fighting a particularly nasty troll which regenerated ten hit points a turn. After every turn, simply enter the MELEE mode, use the spell function for the first available character, and enter minus ten (–10) as the amount of damage inflicted by the spell. Ten hit points will be added to the monster whose number you designated. After returning to the main menu (by entering '32' and pressing RETURN) you may resume play normally.

Entering '34' as an opponent number will take you directly into the monster combat. This is useful for those times when the bad guys have the initiative. Note that, once into monster combat, you may pass play onto the next monster by pressing 'P' or return to the main menu by pressing 'R'. The only other commands recognized in that mode are the numbers 0 to 9, corresponding to the player characters' numbers.

Damage inflicted by non-combat means may be entered directly through the character sheet mode, which is why I didn't include a spell function in the monster side of the combat.

A word to the purists out there. D&D combat can get as complex as you could possibly want, with innumerable modifiers, zone hits, partial armor destruction, *ad nauseum*. This program definitely does not take all that there is in D&D combat rules into account. It'd take a program all by itself to do that. The basic combat system is retained without modification, with the different combat tables for the different classes, hit and damage modifiers, multiple attacks, missile attacks, and so on. Those of you who're sticklers for detail and thrive on complexity would perhaps be better served by doing your combat manually and using the rest of the program for housekeeping.

- CHARACTER SHEET

From the main menu, pressing a number between 0 and 9 inclusive will display a character sheet on the screen. The sheet contains the essential information on each character which is needed as a reference throughout the game.

Hit points, money, and weapon status can be changed directly in this mode, and all other information can be changed through the initialization mode. Additional information can be kept in an individual file accessed through the ROOMS function. This additional file would contain listings of weapons, armor, equipment, special items, magical spells, etc., as well as any pertinent information on the character itself, like special talents, skills, and so on.

You may directly enter MELEE, ROOMS, or return to the main menu by pressing M, Z, or R, respectively.

S - DUNGEON STATUS

The status display lists the dungeon time, the date, weather conditions (outside, of course), temperature, and wind. The weather conditions are random and not subject to control by the DM, although they can certainly be ignored if they don't fit into the scheme of things.

The time and date, on the other hand, can be changed. Pressing '1' will increment the time by ten minutes, '2' by an hour, and '3' by a day. In addition, one minute is added to the time for every combat round played.

Press 'R' to return to the main menu.

E - ENTER ROOM

To explain the ENTER ROOM function, I also have to explain the LOAD ROOM function at the same time. Pressing 'E' will cause the question, "What is the room number?" to be displayed on the screen. It is asking you which of the files created by LOAD ROOM you want to be dumped into memory. At this point, I'll explain the LOAD ROOM function and come back to ENTER ROOM.

L - LOAD ROOM is a routine which allows you to load monster data into files to be called up later by the ENTER ROOM function. This data is used in the MELEE function, and is actually all of the monster statistics for combat resolution. The data includes the number of monsters, hit dice, individual hit points, number of attacks, and the damage per attack.

Pressing L will fetch up the same question: "What is the room number?" You may enter any 3-character code you wish. (This is actually the filename extender for a file called D:COMBAT.) The code can be any combination of three numbers and/or letters, but if you were smart, you'd code them to correspond to the room numbers on your dungeon map, or the numbered encounter areas in your outdoor dungeon. You can use this function to store the statistics for taverns full of troublemakers, and you can also use it to load statistics on the spot for unexpected or random encounters.

As an example of how these two functions work, I give you the following. Every good Dungeon Master has a map of his dungeon and some sort of key to tell him what's in each of the rooms. The rooms are normally numbered, and the key describes what's in each of the numbered rooms, including traps, monsters, treasure, etc. As the dungeon party explores the different rooms, they battle the monsters lurking within, with the Dungeon Master taking the monster statistics from his key and conducting the combat manually. With this program, you'd simply type in the room number to load the statistics into memory and then use the MELEE function to conduct combat.

By loading the monster data into files ahead of time, the dungeon runs smoothly and with a minimum of time spent on game mechanics. Your players get more actual playing time, and your computer does most of the work.

After you've assigned a room number, the LOAD ROOM function will then ask, "How many monsters?" Simply type in the number of monsters in that particular room or encounter area. You will then be asked for the monsters' hit dice (equivalent to a character's level). You may not have monsters of different levels in the same room. If you insist on it, add up the levels and divide by the number of monsters to get an average. It works out pretty close. You'll be asked next for the monster armor class. If you have monsters with different armor classes, take an average, or do the odd ones by hand.

Next, give the hit points for each monster. Here, you can compensate for averaged armor class or level by adding hit points.

After all of the monsters have been assigned their hit points, you'll be asked the number of attacks per turn for each monster (you know, the old claw/claw/bite routine). Again, you must average out the damage for attacks which are different. For instance, if the claws did 4 hit points damage and the bite did 8, add them together (4 + 4 + 8) and divide by the number of attacks (3 in this case), rounding up any fractions.

After you type in the damage per attack, a file will automatically be created, and the program will return to the main menu. Note that the statistics you just entered are still in memory, so you could go right to MELEE if you wished.

Lastly, the maximum number of monsters which can be in a single file is thirty.

D - DICE

Entering a number of any magnitude, including negative numbers, will generate a random number between that number and one, inclusive. By entering anything other than a number, the program will return to the command menu.

FILE DATA and GET DATA

Pressing 'F' or 'G' will conjure up the respective routines for saving or retrieving the character statistics and dungeon status data stored in memory. I made this a two-step process so that, if either letter is pressed accidentally, you have a chance to override the command. 'Y' will initiate the command, and 'N' will override and return the program to the command menu.

To file data, type in any legal filename, but without the device call. In other words, if you wanted your filename to be "D:MURRAY.123", simply type MURRAY.123.

If, when retrieving data, you call for a non-existent file, the program will list the files on the disk in the disk drive and then return to the command menu.

To get data, follow the same procedure as filing data.

I - INITIALIZATION

This mode is used to initially enter the player character statistics, change statistics, or add new player characters. Note that, before any data is entered, the player number must be specified. Once you've identified a particular player, you may enter any number of statistics. If you want to enter data for another player, you must re-identify him.

Most of the entries are self-explanatory, but a few need some words of clarification.

As I stated before, each character may have a maximum of five weapons in this program, and each weapon must be assigned a number. Weapons one through four are hand held weapons, and the number you'll be asked to enter is the maximum possible damage that the weapon can inflict. For instance, player 2's number one weapon is a longsword capable of 1-8 hit points of damage. You'd simply enter an 8 when called upon to do so. If no weapon exists for a particular number, enter 0.

The number of attacks per round must be entered as follows. Enter 1 for one attack per round. Enter 2 for two attacks per round. Enter 3 for three attacks every two rounds.

Finally, enter exceptional strength bonuses in the form of a decimal (18/77 would be entered as 18.77).

ROOMS and WRITE

The biggest headache in running a dungeon is organizing the maps, room descriptions, character sheets, combat tables, reference books, index cards, notes and dice into a system where information can be looked up and processed quickly enough to keep the game from bogging down. It isn't easy or even always possible. This part of the program can help, though, by eliminating the need for a lot of that paperwork.

The WRITE function allows you to write room descriptions, non-player character sheets, artifact or treasure descriptions, or virtually anything you want, and save it on disk. You'd code the files with the same 3-digit code used for the LOAD ROOM files, except this filename extender belongs to a file named D:WITCHES. The reason for the different filenames is so that you could have a room description and a monster data file with the same code number. The intent of this feature was to enable a Dungeon Master to type his room descriptions and save them on disk. At the same time, any monsters in those rooms would have data files created for them with the same room numbers. The DM would then only have to type 'Z' and a room number to see what was in the room, and, if combat was likely, to call up the monster data file through the ENTER ROOM function to resolve it.

You're not limited to a single screen of text, either. You can write up to 300 lines of text, or roughly thirteen screens on a single file. When the file is called back, only one screen at a time will be displayed; pressing any key will display subsequent lines.

To use the WRITE function, type in a code number. A data file will be created. The screen will clear, except for a question mark, which is actually an input prompt. Simply enter one line at a time, pressing RETURN at the end of the line. If you want to leave blank lines, just press RETURN. When you've finished and want to file it, press RETURN, type one asterisk (*), and press RETURN again.

To retrieve what you've written, press 'Z' - ROOMS and type the same code number. What you'd written will be displayed on the screen.

The applications for this particular function are many and varied. For example, you could possibly use the keyboard graphics symbols to create room diagrams along with the descriptions. Use your imagination, experiment with it, and above all practice using it until the commands become second nature. I think you'll find that the speed and accuracy are well worth the effort. □

Program outline.

Lines 4-24 — Initialization
Lines 25-75 — Combat tables
Lines 80-86 — More Initialization
Lines 89-98 — Input number of players
Lines 100-550 — Combat routine, combat table adjustment & multiple attacks routine
Lines 1000-2050 — Race input and display
Lines 3000-3030 — Save character data
Lines 3500-3525 — Retrieve character data
Lines 4000-6045 — Store character data (Initialization Mode)
Lines 6200-6900 — Monster Combat
Lines 6905-6915 — ENTER ROOM routine
Lines 7000-7175 — Weather, time, date, etc. routines
Lines 7500-7600 — ROOMS routine
Lines 8050-8075 — DICE routine
Lines 8500-8590 — WRITE routine
Lines 9000-9019 — Main Command Menu
Lines 9505-10065 — Character Sheet display and input
Lines 15000-15040 — LOAD ROOM routine
Lines 20000-20005 — Input Error Handler
Lines 30000-30055 — Disk directory lister

Command table.

MODE	COMMAND	RESULT
	0 to 29	Indicates the number of the player character's opponent in combat
MELEE		
	31	Pass play to next player
	32	Return to main menu
	33	Activate the Spell Function ('S' to repeat)
	34	Go directly to monster combat
Monster	'N'	Return to main menu
Combat?	'Y'	Go to monster combat
	'P'	Pass play to next monster
	'R'	Return to main menu
	0 to 9	Indicates the monster's opponent in combat
	0 to 9	Displays the indicated player's statistics
CHARACTER SHEET	'W'	Prepare/put away weapon. 0 - put away weapon. 1 to 5 - weapon number prepared
	'H'	Change Hit Points
	'G'	Change Gold Pieces
	'S'	Change Silver Pieces
	'C'	Change Copper Pieces
	'E'	Change Electrum Pieces
	'P'	Change Platinum Pieces
	'Z'	Go to ROOMS function
	'M'	Go to MELEE function
	'R'	Return to Main Menu
	1	Increments minutes by ten
DUNGEON	2	Increments hours by one
STATUS	3	Increments days by one
	'R'	Return to Main Menu
ENTER ROOM	XXX	Any three-character alphanumeric code calls up and loads the monster combat statistics filed under that code by the LOAD ROOM function (filename: "D:COMBAT.XXX)
DICE	Any number	Generates a random number between 1 and the number entered. Will accept negative numbers.
	Any letter	Return to main menu
FILE DATA	Any legal filename	Files player character statistics and Dungeon Status data under given filename (see text for details)
GET DATA	Any legal filename	Retrieves player character statistics and the Dungeon Status Data filed under the given filename (see text for details)
INITIALIZATION		Stores statistics in memory per the listed items. Note that the player number must be specififed first, before any other data is entered.
LOAD ROOM	XXX	See ENTER ROOM and text
WRITE	XXX	Allows you to type up to thirteen screens full of text and save it to disk under a three-character alphanumeric code (filename: "D:WITCHES.XXX)
ROOMS	XXX	Allows you to retrieve the text filed under the given code by the WRITE function

```
4 GRAPHICS 1:POSITION 4,12:POKE 709,0:
POKE 710,0:? #6;"PLEASE WAIT":TRAP 200
00
10 DIM CT(20,18),A$(10),B$(10),C$(10),
D$(10),E$(10),F$(10),G$(10),H$(10),J$(
10),K$(10),Y$(3),AC(10),L(10),HP(10)
15 DIM GP(10),SP(10),CP(10),PP(10),EP(
10),HM(10),DM(10),RA(10),ST(10),WD(10,
5),C(10),T(3),AZ$(15),SB(10)
16 DIM ACA(10),W(10),AL$(2),MON(40)
20 DIM CMD$(125),IN(10),WI(10),DX(10),
CN(10),CH(10),N$(2),T$(5),R$(5),AA$(2)
,BB$(2),CC$(2),DD$(2),EE$(2),FF$(2)
22 DIM GG$(2),HH$(2),JJ$(2),KK$(2),NUM
$(5),AM$(2),CL$(15),RA$(10),GE$(7),GE(
10),HE(10,2),WE(10),ATT(10),ATT1(10)
23 AM$="AM":T(1)=0:T(2)=0:AZ$="D:"
24 DIM GA$(12),GB$(12),GC$(12),GD$(12)
,GL$(12),GF$(12),GH$(12),GI$(12),GJ$(1
2),GK$(12),MT(20,16)
25 FOR E=1 TO 18:FOR X=20 TO 1 STEP -1
:READ N:CT(X,E)=N:NEXT X:NEXT E:AC1=AC
+9
26 FOR E=1 TO 16:FOR X=20 TO 1 STEP -1
:READ N:MT(X,E)=N:NEXT X:NEXT E
30 DATA 10,11,12,13,14,15,16,17,18,19,
20,20,20,20,20,20,21,22,23,24,10,11,12
,13,14,15,16,17,18,19,20,20,20,20,20
35 DATA 20,21,22,23,24,8,9,10,11,12,13
,14,15,16,17,18,19,20,20,20,20,20,20,2
1,22,8,9,10,11,12,13,14,15,16,17,18
40 DATA 19,20,20,20,20,20,20,21,22,6,7
,8,9,10,11,12,13,14,15,16,17,18,19,20,
20,20,20,20,20,6,7,8,9,10,11,12,13,14
45 DATA 15,16,17,18,19,20,20,20,20,20,
20,4,5,6,7,8,9,10,11,12,13,14,15,16,17
,18,19,20,20,20,20,4,5,6,7,8,9,10,11
50 DATA 12,13,14,15,16,17,18,19,20,20,
20,20,2,3,4,5,6,7,8,9,10,11,12,13,14,1
5,16,17,18,18,20,20,2,3,4,5,6,7,8,9
55 DATA 10,11,12,13,14,15,16,17,18,19,
20,20,0,1,2,3,4,5,6,7,8,9,10,11,12,13,
14,15,16,17,18,19,0,1,2,3,4,5,6,7,8
60 DATA 9,10,11,12,13,14,15,16,17,18,1
9,-2,-1,0,1,2,3,4,5,6,7,8,9,10,11,12,1
3,14,15,16,17,-2,-1,0,1,2,3,4,5,6,7
65 DATA 8,9,10,11,12,13,14,15,16,17,-4
,-3,-2,-1,0,1,2,3,4,5,6,7,8,9,10,11,12
,13,14,15,-4,-3,-2,-1,0,1,2,3,4,5,6,7
66 DATA 8,9,10,11,12,13,14,15,-6,-5,-4
,-3,-2,-1,0,1,2,3,4,5,6,7,8,9,10,11,12
,13,-6,-5,-4,-3,-2,-1,0,1,2,3,4,5,6
67 DATA 7,8,9,10,11,12,13,10,11,12,13,
14,15,16,17,18,19,20,20,20,20,20,20,21
,22,23,24,9,10,11,12,13,14,15,16,17
68 DATA 18,19,20,20,20,20,20,20,21,22,
23,8,9,10,11,12,13,14,15,16,17,18,19,2
0,20,20,20,20,20,21,22,7,8,9,10,11,12
69 DATA 13,14,15,16,17,18,19,20,20,20,
20,20,20,21,6,7,8,9,10,11,12,13,14,15,
16,17,18,19,20,20,20,20,20,20
70 DATA 5,6,7,8,9,10,11,12,13,14,15,16
,17,18,19,20,20,20,20,20,3,4,5,6,7,8,9
,10,11,12,13,14,15,16,17,18,19,20,20
71 DATA 20,3,4,5,6,7,8,9,10,11,12,13,1
4,15,16,17,18,19,20,20,20,3,4,5,6,7,8,
9,10,11,12,13,14,15,16,17,18,19,20,20
72 DATA 20,2,3,4,5,6,7,8,9,10,11,12,13
,14,15,16,17,18,19,20,20,0,1,2,3,4,5,6
,7,8,9,10,11,12,13,14,15,16,17,18,19
73 DATA 0,1,2,3,4,5,6,7,8,9,10,11,12,1
3,14,15,16,17,18,19,-1,0,1,2,3,4,5,6,7
,8,9,10,11,12,13,14,15,16,17,18
74 DATA -1,0,1,2,3,4,5,6,7,8,9,10,11,1
2,13,14,15,16,17,18,-2,-1,0,1,2,3,4,5,
6,7,8,9,10,11,12,13,14,15,16,17
75 DATA -3,-2,-1,0,1,2,3,4,5,6,7,8,9,1
0,11,12,13,14,15,16
80 A1=1:A0=A1-A1:A2=A1+A1:A3=A2+A1:A4=
A3+A1:A5=A3+A2:A6=A5+A1:A7=A5+A2:A8=A4
+A4:A9=A5+A4:A10=A5+A5:A11=A6+A5
85 A12=A2*A6:A13=A6+A7:A14=A2*A7:A15=A
3*A5:A16=A2*A8:A17=A9+A8:A18=A3*A6:A19
=A10+A9:A20=A10*A2:GRAPHICS 0
86 POKE 82,2
89 TRAP 89:POKE 712,128:POKE 710,128:P
OKE 752,A1:? "K":POSITION 11,12:? "HO
W MANY PLAYERS";:INPUT NUM:NUM=NUM-A1
95 IF NUM>=A0 THEN IF NUM<A10 THEN 900
0
98 GOTO 89
100 CMD$=STR$(P):CMD$(1,1)=CHR$(ASC(CM
D$(1,1))+128)
105 RETURN
110 IF J=34 THEN 6200
112 RETURN
150 ? "K":FOR P=A0 TO NUM:? :POKE 710,
50:POKE 709,60:POKE 712,50:GOSUB 100
151 IF HP(P)<=A0 THEN IF HP(P)>-A10 TH
EN ? "PLAYER ";CMD$;" IS UNCONSCIOUS":
? :GOTO 310
152 IF HP(P)<-A9 THEN ? "PLAYER ";CMD$
;" HAS BEEN KILLED.":? :GOTO 310
153 ? ,,"    31 = PASS":? ,,"    32 =
RETURN":? ,,"    33 = SPELL":? ,,"
34 = MONSTER"
154 IF W(P)=5 THEN FOR CMD=1 TO SB(P)
155 TRAP 20000:? "PLAYER ";CMD$;"'S OP
PONENT";:INPUT J:TRAP 40000:O=A0:IF J=
31 THEN 310
156 GOSUB 110:IF J=32 THEN 9000
157 IF J=33 THEN ? "HOW MANY HP DAMAGE
";:INPUT X:? "AGAINST WHICH MONSTER";:
INPUT E:MON(E+4)=MON(E+4)-X:GOTO 159
158 GOTO 161
159 ? :? "TO REPEAT SPELL, PRESS 'S'":
J=E:OPEN #1,4,0,"K:":GET #1,X:CLOSE #1
:IF X=83 THEN ? :J=33:GOTO 157
160 GOTO 310
161 AC=MON(3):IF AC<=A10 THEN IF AC>=-
A9 THEN O=A5
162 IF J<0 OR J>MON(1)-1 THEN ? "0 - "
;MON(1)-1;".  TRY AGAIN.":? :GOTO 154
164 IF O<>A5 THEN ? "RTRY AGAIN.":? :F
OR I=A1 TO 500:NEXT I:? "K":GOTO 154
165 IF W(P)=A0 THEN ? "PLAYER ";P;" DO
ESN'T HAVE A WEAPON READY":? "WEAPON";
:INPUT W:IF W<A0 OR W>A5 THEN 165
166 IF W(P)=0 THEN W(P)=W:GOTO 310
200 IF C(P)=A1 OR C(P)=A2 OR C(P)=A3 T
HEN AC1=AC+10:H=CT(AC1,L(P)):GOTO 300
205 IF C(P)=A7 OR C(P)=A8 THEN 219
206 IF C(P)=A9 OR C(P)=A10 THEN 229
210 IF L(P)=A3 OR L(P)=A4 OR L(P)=A5 T
HEN D=-1
211 IF L(P)=A6 OR L(P)=A7 OR L(P)=A8 T
HEN D=-A2
212 IF L(P)=A9 OR L(P)=A10 OR L(P)=11
THEN D=-A3
213 IF L(P)=12 OR L(P)=13 OR L(P)=14 T
HEN D=-A4
214 IF L(P)=15 OR L(P)=16 OR L(P)=17 T
HEN D=-A5
215 IF L(P)=18 THEN D=-A6
216 AC1=AC+10:D1=L(P)+D:H=CT(AC1,D1):G
OTO 300
219 IF L(P)<A3 THEN D=A0
220 IF L(P)=A3 OR L(P)=A4 OR L(P)=A5 O
R L(P)=A6 THEN D=-A2
221 IF L(P)=A7 OR L(P)=A8 OR L(P)=A9 O
R L(P)=A10 THEN D=-A4
222 IF L(P)=11 OR L(P)=12 OR L(P)=13 O
R L(P)=14 THEN D=-A6
223 IF L(P)=15 OR L(P)=16 OR L(P)=17 O
R L(P)=18 THEN D=-A8
224 D1=L(P)+D:AC1=AC+A10:H=CT(AC1,D1):
FOR E=A1 TO A8:IF D1=E THEN H=H+A1
225 NEXT E:GOTO 300
229 IF L(P)<A3 THEN D=A0
230 IF L(P)=A3 THEN D=-A1
231 IF L(P)=A4 OR L(P)=A5 OR L(P)=A6 O
R L(P)=A7 THEN D=-A3
232 IF L(P)=A8 OR L(P)=A9 THEN D=-A4
233 IF L(P)=A10 OR L(P)=11 OR L(P)=12
THEN D=-A6
234 IF L(P)=13 OR L(P)=14 THEN D=-A8
235 IF L(P)=15 OR L(P)=16 OR L(P)=17 T
HEN D=-A9
236 IF L(P)=18 THEN D=-A10
237 D1=L(P)+D:AC1=AC+A10:H=CT(AC1,D1):
FOR E=A1 TO A10:IF D1=E THEN H=H+A1
238 NEXT E:FOR E=16 TO 18:IF D1=E THEN
 H=H+A1
239 NEXT E
```

```
300 SWING=INT(20*RND(A1)+A1)+HM(P):IF
W(P)=5 THEN SWING=SWING-HM(P):SWING=SW
ING+RA(P)
301 ? "H=";H:? "SWING=";SWING
302 IF SWING)=H THEN DAM=INT(WD(P,W(P)
)*RND(A1)+A1)+DM(P):IF W(P)=5 THEN DAM
=DAM-DM(P)
305 IF SWING)=H THEN ? "A HIT!":? "DAM
AGE=";DAM;" HIT POINTS"
306 IF SWING)=H THEN MON(J+4)=MON(J+4)
-DAM:IF MON(J+4)(=0 THEN ? "YOU'VE KIL
LED IT!!"
307 IF W(P)=5 THEN NEXT CMD
308 IF ATT1(P)=0 THEN 550
309 IF ATT1(P))0 THEN 500
310 NEXT P:T(A2)=T(A2)+A1
312 ? :? "PRESS ANY KEY TO CONTINUE":O
PEN #1,4,0,"K:":GET #1,E:CLOSE #1:IF E
)=0 THEN 6200
500 IF ATT(P)=2 THEN ATT1(P)=0:GOTO 15
1
503 IF ATT1(P)=2 THEN ATT1(P)=0:GOTO 1
51
505 IF ATT1(P)=3 THEN ATT1(P)=2:GOTO 3
10
550 ATT1(P)=ATT(P):GOTO 310
1000 ON P+A1 GOTO 1005,1010,1015,1020,
1025,1030,1035,1040,1045,1050
1005 INPUT GA$:RETURN
1010 INPUT GB$:RETURN
1015 INPUT GC$:RETURN
1020 INPUT GD$:RETURN
1025 INPUT GL$:RETURN
1030 INPUT GF$:RETURN
1035 INPUT GH$:RETURN
1040 INPUT GI$:RETURN
1045 INPUT GJ$:RETURN
1050 INPUT GK$:RETURN
2000 ON P+1 GOTO 2005,2010,2015,2020,2
025,2030,2035,2040,2045,2050
2005 ? "↑↑↑↑     ";GA$:RETURN
2010 ? "↑↑↑↑     ";GB$:RETURN
2015 ? "↑↑↑↑     ";GC$:RETURN
2020 ? "↑↑↑↑     ";GD$:RETURN
2025 ? "↑↑↑↑     ";GL$:RETURN
2030 ? "↑↑↑↑     ";GF$:RETURN
2035 ? "↑↑↑↑     ";GH$:RETURN
2040 ? "↑↑↑↑     ";GI$:RETURN
2045 ? "↑↑↑↑     ";GJ$:RETURN
2050 ? "↑↑↑↑     ";GK$:RETURN
3000 ? "K":POSITION 8,11:? "L     FILE
DATA?":POSITION 8,12:? "   ARE YOU SUR
E?":OPEN #1,4,0,"K:":GET #1,J:CLOSE #1
3001 IF J=89 THEN TRAP 3004:GOTO 3004
3002 IF J=78 THEN 9000
3003 GOTO 3000
3004 T$=CHR$(155):? "K":POSITION A8,11
:? "LWHAT IS THE NAME OF THE":POSITION
 8,12:? "DATA FILE";:INPUT CMD$
3005 TRAP 3004:AZ$="D:":AZ$(3,3+LEN(CM
D$))=CMD$
3007 OPEN #1,8,0,AZ$:FOR J=A1 TO 128:P
UT #1,0:NEXT J
3010 FOR E=A0 TO NUM:PRINT #1;AC(E);T$
;C(E);T$;HP(E);T$;HM(E);T$;DM(E);T$;RA
(E)
3012 PRINT #1;ACA(E);T$;SB(E);T$;L(E);
T$;ST(E);T$;IN(E);T$;WI(E);T$;DX(E)
3015 PRINT #1;W(E);T$;GP(E);T$;SP(E);T
$;CP(E);T$;PP(E);T$;EP(E);T$;CN(E)
3016 PRINT #1;CH(E);T$;ATT(E);T$;WE(E)
;T$;HE(E,0);T$;HE(E,1);T$;GE(E):NEXT E
3017 PRINT #1;AA$;T$;BB$;T$;CC$;T$;DD$
;T$;EE$;T$;FF$;T$;GG$
3018 PRINT #1;HH$;T$;JJ$;T$;KK$;T$;A$;
T$;B$;T$;C$;T$;D$;T$;E$;T$;F$;T$;G$
3020 PRINT #1;H$;T$;J$;T$;K$;T$;Z;T$;M
;T$;T(1)
3021 PRINT #1;T(2):PRINT #1;AM$;T$;Y
3022 FOR J=A1 TO A5:FOR E=A0 TO NUM:PR
INT #1;WD(E,J):NEXT E:NEXT J
3025 PRINT #1;GA$;T$;GB$;T$;GC$;T$;GD$
;T$;GL$;T$;GF$;T$;GH$;T$;GI$;T$;GJ$;T$
;GK$
3030 CLOSE #1:END
3500 ? "K":POSITION 8,11:? "L     GET
DATA?":POSITION 8,12:? "   ARE YOU SUR
E?":OPEN #1,4,0,"K:":GET #1,J:CLOSE #1
3501 IF J=89 THEN 3505
3502 IF J=78 THEN 9000
3503 GOTO 3500
3505 CLOSE #3:TRAP 30000:? "K":POSITIO
N A8,11:? "WHAT IS THE NAME OF THE":PO
SITION A8,12:? "DATA FILE";:INPUT CMD$
3506 AZ$="D:":AZ$(3,3+LEN(CMD$))=CMD$:
XIO 3,#3,4,0,AZ$:FOR J=A1 TO 128:GET #
3,R:NEXT J
3507 FOR E=A0 TO NUM:INPUT #3,J:AC(E)=
J:INPUT #3,J:C(E)=J:INPUT #3,J:HP(E)=J
:INPUT #3,J:HM(E)=J:INPUT #3,J:DM(E)=J
3508 INPUT #3,J:RA(E)=J:INPUT #3,J:ACA
(E)=J:INPUT #3,J:SB(E)=J:INPUT #3,J:L(
E)=J:INPUT #3,J:ST(E)=J
3509 INPUT #3,J:IN(E)=J:INPUT #3,J:WI(
E)=J:INPUT #3,J:DX(E)=J:INPUT #3,J:W(E
)=J:INPUT #3,J:GP(E)=J
3511 INPUT #3,J:SP(E)=J:INPUT #3,J:CP(
E)=J:INPUT #3,J:PP(E)=J:INPUT #3,J:EP(
E)=J:INPUT #3,J:CN(E)=J
3512 INPUT #3,J:CH(E)=J:INPUT #3,J:ATT
(E)=J:INPUT #3,J:WE(E)=J:INPUT #3,J:HE
(E,0)=J:INPUT #3,J:HE(E,1)=J
3514 INPUT #3,J:GE(E)=J:NEXT E
3516 INPUT #3;AA$,BB$,CC$,DD$,EE$,FF$,
GG$,HH$,JJ$,KK$,A$,B$,C$,D$,E$,F$,G$,H
$,J$,K$,Z,M,T:T(1)=T
3517 INPUT #3;K:T(2)=K:INPUT #3;AM$,Y
3520 FOR J=1 TO 5:FOR E=0 TO NUM:INPUT
 #3;WD:WD(E,J)=WD:NEXT E:NEXT J
3522 INPUT #3;GA$,GB$,GC$,GD$,GL$,GF$,
GH$,GI$,GJ$,GK$
3525 CLOSE #3:FOR E=0 TO NUM:ATT1(E)=A
TT(E):NEXT E:GOTO 9000
4000 ? "A PLAYER NUMBER","L GENDER":
? "B PLAYER NAME","M HEIGHT":? "C CLAS
S"," ","N WEIGHT"
4010 ? "D ALIGNMENT","O LEVEL":? "E ST
RENGTH","P ARMOR CLASS":? "F INTELLIGE
NCE","Q HIT MODIFIER"
4020 ? "G WISDOM"," ","R DAM MODIFIER"
:? "H DEXTERITY","S AC ADJUSTMENT":? "
I CONSTITUTION","T R/A BONUS"
4030 ? "J CHARISMA","U ATTACKS/ROUND":
? "K RACE"," ","V WEAPON":? " ",,"W MO
NTH"
4035 ? ,,"X YEAR":? "↓Y RETURN TO MENU
"
4100 POSITION 3,17:? "HEADING?":OPEN #
3,4,0,"K:":GET #3,J:CLOSE #3:IF J(65 O
R J)89 THEN 4100
4105 POSITION 22,18
4106 ON J-64 GOTO 4110,4115,4120,4125,
4130,4135,4140,4145,4150,4155,4160,416
5,4170,4175,4180,4185,4190,4195,4200
4107 ON J-83 GOTO 4205,4210,4215,4285,
4290,4295
4108 GOTO 4000
4110 ? "↑PLAYER NUMBER";:INPUT P
4112 IF P(A0 OR P)A9 THEN ? "↑0 TO 9,
PLEASE - TRY AGAIN":FOR E=A1 TO A50:NE
XT E:? "↑":? :GOTO 4110
4113 IF P)NUM THEN NUM=P
4114 GOTO 4000
4115 ? "↑PLAYER NAME";:ON P+A1 GOSUB 6
000,6005,6010,6015,6020,6025,6030,6035
,6040,6045:GOTO 4000
4120 ? "↑CLASS";:GOSUB 5500:GOTO 4000
4125 ? "↑ALIGNMENT";:INPUT AL$:GOSUB 5
000:GOTO 4000
4130 ? "↑STRENGTH";:INPUT E:ST(P)=E:GO
TO 4000
4135 ? "↑INTELLIGENCE";:INPUT E:IN(P)=
E:GOTO 4000
4140 ? "↑WISDOM";:INPUT E:WI(P)=E:GOTO
 4000
4145 ? "↑DEXTERITY";:INPUT E:DX(P)=E:G
OTO 4000
4150 ? "↑CONSTITUTION";:INPUT E:CN(P)=
E:GOTO 4000
4155 ? "↑CHARISMA";:INPUT E:CH(P)=E:GO
TO 4000
4160 ? "↑RACE";:GOSUB 1000:GOTO 4000
4165 ? "↑GENDER";:INPUT GE$:IF GE$="M"
 OR GE$="MALE" THEN GE(P)=1:GOTO 4000
4167 IF GE$="F" OR GE$="FEMALE" THEN G
E(P)=2:GOTO 4000
4170 ? "↑HEIGHT (FEET)";:INPUT J:HE(P,
```

```
0)=J:POSITION 29,19:? "↑(INCHES)";:INP
UT J:HE(P,1)=J:GOTO 4000
4175 ? "↑WEIGHT";:INPUT J:WE(P)=J:GOTO
 4000
4180 ? "↑LEVEL";:INPUT E:L(P)=E:GOTO 4
000
4185 ? "↑ARMOR CLASS";:INPUT E:AC(P)=E
:GOTO 4000
4190 ? "↑HIT MODIFIER";:INPUT E:HM(P)=
E:GOTO 4000
4195 ? "↑DAMAGE MODIFIER";:INPUT E:DM(
P)=E:GOTO 4000
4200 ? "↑AC ADJUSTMENT";:INPUT E:ACA(P
)=E:GOTO 4000
4205 ? "↑R/A BONUS";:INPUT E:RA(P)=E:G
OTO 4000
4210 ? "↑ATTACKS/TURN";:INPUT E:ATT(P)
=E:GOTO 4000
4215 ? "↑WEAPON DAMAGE";? "↓→→WEAPON #
1";:INPUT E:WD(P,A1)=E:? "→→WEAPON #2"
;:INPUT E:WD(P,A2)=E
4270 ? "→→WEAPON #3";:INPUT E:WD(P,A3)
=E:? "  WEAPON #4";:INPUT E:WD(P,A4)=E
4280 ? "  WEAPON #5";:INPUT E:WD(P,A5)
=E:? "NUMBER OF HITS PER ROUND";:INPUT
 E:SB(P)=E:GOTO 4000
4285 ? "↑MONTH";:INPUT M:GOTO 4000
4290 ? "↑YEAR";:INPUT Y:GOTO 4000
4295 GOTO 9000
5000 IF P=A0 THEN AA$=AL$
5002 IF P=A1 THEN BB$=AL$
5004 IF P=A2 THEN CC$=AL$
5006 IF P=A3 THEN DD$=AL$
5008 IF P=A4 THEN EE$=AL$
5010 IF P=A5 THEN FF$=AL$
5012 IF P=A6 THEN GG$=AL$
5014 IF P=A7 THEN HH$=AL$
5016 IF P=A8 THEN JJ$=AL$
5018 IF P=A9 THEN KK$=AL$
5020 RETURN
5500 INPUT CL$:IF CL$="FIGHTER" OR CL$
="F" THEN C(P)=A1
5505 IF CL$="RANGER" OR CL$="R" THEN C
(P)=A2
5510 IF CL$="PALADIN" OR CL$="P" THEN
C(P)=A3
5515 IF CL$="CLERIC" OR CL$="C" THEN C
(P)=A4
5520 IF CL$="DRUID" OR CL$="D" THEN C(
P)=A5
5525 IF CL$="MONK" OR CL$="M" THEN C(P
)=A6
5530 IF CL$="THIEF" OR CL$="T" THEN C(
P)=A7
5540 IF CL$="ASSASSIN" OR CL$="A" THEN
 C(P)=A8
5545 IF CL$="MAGIC-USER" OR CL$="MU" O
R CL$="MAGIC USER" THEN C(P)=A9
5550 IF CL$="ILLUSIONIST" OR CL$="I" T
HEN C(P)=A10
5555 RETURN
6000 INPUT A$:RETURN
6005 INPUT B$:RETURN
6010 INPUT C$:RETURN
6015 INPUT D$:RETURN
6020 INPUT E$:RETURN
6025 INPUT F$:RETURN
6030 INPUT G$:RETURN
6035 INPUT H$:RETURN
6040 INPUT J$:RETURN
6045 INPUT K$:RETURN
6200 ? "ß":POSITION 12,12:? "MONSTER C
OMBAT?":OPEN #1,4,0,"K:":GET #1,J:CLOS
E #1:IF J=78 THEN 9000
6201 IF J=89 THEN 6210
6203 GOTO 6200
6210 FOR E=0 TO MON(1)-1:? :IF MON(E+4
)(=0 THEN ? "MONSTER";E;" HAS BEEN KIL
LED":? :? :GOTO 6900
6212 FOR X=1 TO MON(35):? "MONSTER ";E
;"'S OPPONENT":OPEN #3,4,0,"K:":GET #3
,J:CLOSE #3:IF J=80 THEN 6900
6215 IF J=82 THEN 9000
6218 IF J(48 OR J)57 THEN 6212
6220 IF VAL(CHR$(J)))NUM THEN 6212
6225 P=VAL(CHR$(J)):SWING=INT(20*RND(A
1)+A1):H=MT(AC(P)+10,MON(2))
6227 IF SWING)=H THEN ? "A HIT!!":DAM=
INT(MON(36)*RND(A1)+A1):? "DAMAGE = ";
DAM;" HP":? :?
6230 IF SWING(H THEN ? "A MISS!":? :?
:GOTO 6895
6235 HP(P)=HP(P)-DAM:DAM=0
6895 NEXT X
6900 NEXT E:? :? "PRESS ANY KEY TO CON
TINUE":OPEN #1,4,0,"K:":GET #1,E:CLOSE
 #1:IF E)=0 THEN 9000
6905 TRAP 30000:? "ß":POSITION 2,12:?
"WHAT IS THE ROOM NUMBER";:INPUT CMD$
6910 CLOSE #1:AZ$="D:COMBAT.":AZ$(10,1
0+LEN(CMD$))=CMD$:XIO 3,#1,4,0,AZ$
6915 FOR E=1 TO 40:INPUT #1,J:MON(E)=J
:NEXT E:CLOSE #1:GOTO 9000
7000 F=INT(100*RND(A1))+A1:E=INT(30*RN
D(A1))+A1
7002 ON M GOTO 7003,7003,7004,7004,700
4,7005,7005,7005,7005,7004,7004,7003
7003 X=INT(25*RND(A1)):GOTO 7008
7004 X=INT(25*RND(A1)+25):GOTO 7008
7005 X=INT(33*RND(A1)+60):GOTO 7008
7008 DL=PEEK(560)+256*PEEK(561):B=PEEK
(DL+A4):C=PEEK(DL+A5):POKE 559,A0:POKE
 DL+A4,B:POKE DL+A5,C:POKE DL+A3,66
7009 POKE DL+A6,A6:POKE DL+A7,A6:POKE
DL+13,A6:POKE DL+14,A6:POKE DL+15,A6:P
OKE DL+16,A6:POKE DL+29,65
7010 POKE DL+30,PEEK(560):POKE DL+31,P
EEK(561):POKE 559,34:POKE 710,128:POKE
 712,128:? "ß↓ DUNGEON STATUS":?
7011 IF D=A3 THEN D=A0:GOTO 7015
7012 IF X(30 THEN IF F(=60 THEN IF A)=
100 THEN ? "↑WEATHER:   SNOW     ":D=D
+1:GOTO 7025
7013 IF X)=30 THEN IF X(=34 THEN IF F(
=60 THEN IF A)=100 THEN ? "↑WEATHER:
SLEET    ":D=D+1:GOTO 7025
7014 IF F(=60 THEN IF A)=100 THEN ? "↑
WEATHER:   RAIN      ":D=D+1:GOTO 7025
7015 IF F)30 THEN ? "WEATHER:   FAIR":
D=0:A=0:GOTO 7025
7020 IF F(=30 THEN ? "WEATHER: CLOUDY"
:LET A=A+100
7025 ? "WIND:     ";E;"MPH"
7046 ? "TEMPERATURE: ";X
7050 ? "↓↓  DUNGEON TIME"
7053 ? "↓YEAR : ";Y:? "MONTH: ";M:? "D
AY  : ";Z:? :? "TIME : ";T(A1);":";T(A
2);" ";AM$
7100 J=A0:OPEN #1,4,0,"K:":GET #1,J:CL
OSE #1:IF CHR$(J)="R" THEN 9000
7105 IF J=49 THEN 7115
7112 IF J=50 THEN T(A1)=T(A1)+A1:GOTO
7120
7113 IF J=51 THEN Z=Z+A1:GOTO 7120
7114 GOTO 7100
7115 T(A2)=T(A2)+VAL(CHR$(J))*A10
7120 IF T(A2)>=60 THEN T(A2)=T(A2)-60:
T(A1)=T(A1)+A1
7125 IF T(A1)=11 THEN T(A3)=A3
7130 IF T(A1)=12 THEN IF T(A3)=A3 THEN
 IF AM$="AM" THEN AM$="PM":T(A3)=A0:GO
TO 7145
7135 IF T(A1)=A12 THEN IF T(A3)=A3 THE
N IF AM$="PM" THEN AM$="AM":Z=Z+A1:T(A
3)=A0
7145 IF T(A1)=13 THEN T(A1)=A1
7150 IF Z=31 THEN Z=A1:M=M+A1
7155 IF M=13 THEN M=A1:Y=Y+A1
7175 GOTO 7008
7500 AZ$="D:WITCHES.":? "ßROOM NUMBER"
;:INPUT CMD$:AZ$(11,11+LEN(CMD$))=CMD$
:POKE 710,200:POKE 709,194:POKE 712,0
7505 TRAP 30000:CLOSE #1:OPEN #1,4,0,A
Z$:? "ß"
7510 TRAP 7600
7520 INPUT #1,CMD$
7530 ? CMD$
7535 IF PEEK(84)=23 THEN POSITION 1,23
:? "THERE'S MORE--PRESS ANY KEY WHEN R
EADY↑":GOTO 7538
7536 GOTO 7540
7538 OPEN #4,4,0,"K:":GET #4,J:CLOSE #
4:IF J)=0 THEN ? "ß"
7540 GOTO 7520
```

```
7600 CLOSE #1:OPEN #2,4,0,"K:":GET #2,
J:CLOSE #2:IF J>=0 THEN 9000
8050 DL=PEEK(560)+256*PEEK(561):B=PEEK
(DL+4):C=PEEK(DL+5):POKE 559,0:POKE DL
+4,B:POKE DL+5,C:POKE DL+3,66
8055 POKE DL+12,7:POKE DL+13,7:POKE DL
+14,7:POKE DL+15,7:POKE DL+23,65:POKE
DL+24,PEEK(560):POKE DL+25,PEEK(561)
8060 POKE 559,34:POKE 87,0:POKE 710,19
2:POKE 712,192
8065 TRAP 9000:? "{CLEAR}":POSITION 2,8:? "R
ANDOM NUMBER";:INPUT RN
8070 ? "{CLEAR}":POSITION 2,8:E=INT(RN*RND(0
)+1):? "NUMBER = ";E
8075 FOR E=1 TO 150:NEXT E:GOTO 8065
8500 POKE 710,200:POKE 709,192:POKE 71
2,0
8520 AZ$="D:WITCHES.":? "{CLEAR}":? "WHAT IS
 THE ROOM NUMBER";:INPUT CMD$
8530 AZ$(11,11+LEN(CMD$))=CMD$:OPEN #1
,8,0,AZ$
8540 ? "{CLEAR}"
8550 INPUT CMD$
8560 IF CMD$="*" THEN 8590
8570 ? #1;CMD$
8580 GOTO 8550
8590 CLOSE #1:GOTO 9000
9000 TRAP 20000:GRAPHICS 0:POKE 712,12
8:POKE 710,128:POKE 709,140:POKE 752,1
9001 ? "{CLEAR}":POSITION 0,5:? "}M = MELEE
":? "}# = CHARACTER SHEET":? "}S = DUN
GEON STATUS":? "}E = ENTER ROOM"
9002 ? "}D = DICE":? "}F = FILE DATA":
? "}G = GET DATA":? "}I = INITIALIZATI
ON":? "}L = LOAD ROOM"
9003 ? "}Z = ROOMS":? "}W = WRITE"
9004 ? "}↓↓COMMAND?";
9007 CLOSE #1:TRAP 20000:OPEN #1,4,0,"
K:":GET #1,CMD:CLOSE #1:IF CMD=69 THEN
 6905
9008 IF CMD=83 THEN 7000
9009 IF CMD=68 THEN 8050
9010 IF CMD=71 THEN 3500
9012 IF CMD=77 THEN 150
9013 IF CMD=73 THEN 4000
9014 IF CMD=70 THEN 3000
9015 IF CMD=76 THEN 15000
9016 IF CMD=90 THEN 7500
9017 IF CMD=87 THEN 8500
9018 IF CMD<48 OR CMD>57 THEN 9000
9019 P=VAL(CHR$(CMD)):GOSUB 100
9505 POKE 712,P*16+10:POKE 710,P*16+10
:POKE 709,P*16:? "{CLEAR}↓↓PLAYER ";CMD$;"
 ";
9506 ON P+1 GOTO 9600,9601,9602,9603,9
604,9605,9606,9607,9608,9609
9510 ON C(P) GOTO 9610,9611,9612,9613,
9614,9615,9616,9617,9618,9619
9511 ON P+1 GOTO 9800,9802,9804,9806,9
808,9810,9812,9814,9816,9818
9512 ? "↓ARMOR CLASS          ";AC(P):? "
↑↑↑↑↑ST ";ST(P)
9514 ? "HIT POINTS          ";HP(P):? "↑
↑↑↑↑IN ";IN(P)
9516 ? "HIT MODIFIER        ";HM(P):? "↑
↑↑↑↑WI ";WI(P)
9518 ? "DAMAGE MODIFIER     ";DM(P):? "↑
↑↑↑↑DX ";DX(P)
9520 ? "R/A BONUS           ";RA(P):? "↑
↑↑↑↑CN ";CN(P)
9522 ? "AC ADJUSTMENT       ";ACA(P):? "
↑↑↑↑↑CH ";CH(P)
9524 ? "MISSILE MULTIPLE    ";SB(P)
9526 ? "LEVEL               ";L(P):? "↑}
}}     HT ";HE(P,0);"'";" ";HE(P,1);CHR
$(34)
9528 ? "WEAPON READY        ";W(P):? "↑}
}}     WT ";WE(P);" LBS"
9530 ? "↓GOLD                ";GP(P)
9533 ? "SILVER              ";SP(P):IF G
E(P)=1 THEN ? "↑↑↑↑     MALE":GOTO 9535
9534 IF GE(P)=2 THEN ? "↑↑↑↑     FEMALE
"
9535 ? "COPPER              ";CP(P):GOSU
B 2000
9537 ? "PLATINUM            ";PP(P)
9540 ? "ELECTRUM            ";EP(P)
9599 GOTO 9700
9600 ? A$:GOTO 9510
9601 ? B$:GOTO 9510
9602 ? C$:GOTO 9510
9603 ? D$:GOTO 9510
9604 ? E$:GOTO 9510
9605 ? F$:GOTO 9510
9606 ? G$:GOTO 9510
9607 ? H$:GOTO 9510
9608 ? J$:GOTO 9510
9609 ? K$:GOTO 9510
9610 ? "}}}↑FIGHTER":GOTO 9511
9611 ? "}}}↑RANGER":GOTO 9511
9612 ? "}}}↑PALADIN":GOTO 9511
9613 ? "}}}↑CLERIC":GOTO 9511
9614 ? "}}}↑DRUID":GOTO 9511
9615 ? "}}}↑MONK":GOTO 9511
9616 ? "}}}↑THIEF":GOTO 9511
9617 ? "}}}↑ASSASSIN":GOTO 9511
9618 ? "}}}↑MAGIC-USER":GOTO 9511
9619 ? "}}}↑ILLUSIONIST":GOTO 9511
9700 POSITION 3,22:? "COMMAND?";:OPEN
#1,4,0,"K:":GET #1,X:CLOSE #1
9702 IF X=77 THEN 150
9704 IF X=82 THEN 9000
9705 IF X=71 THEN 10000
9706 IF X=83 THEN 10010
9707 IF X=67 THEN 10020
9708 IF X=80 THEN 10030
9709 IF X=69 THEN 10040
9710 IF X=72 THEN 10050
9711 IF X=87 THEN 10060
9712 IF X=90 THEN 7500
9715 IF X<48 OR X>57 THEN 9700
9730 P=VAL(CHR$(X)):GOSUB 100:GOTO 950
5
9800 POSITION 36,2:? AA$:GOTO 9512
9802 POSITION 36,2:? BB$:GOTO 9512
9804 POSITION 36,2:? CC$:GOTO 9512
9806 POSITION 36,2:? DD$:GOTO 9512
9808 POSITION 36,2:? EE$:GOTO 9512
9810 POSITION 36,2:? FF$:GOTO 9512
9812 POSITION 36,2:? GG$:GOTO 9512
9814 POSITION 36,2:? HH$:GOTO 9512
9816 POSITION 36,2:? JJ$:GOTO 9512
9818 POSITION 36,2:? KK$:GOTO 9512
10000 ? "  GOLD PIECES";:INPUT J:GP(P)
=J:GOTO 9505
10010 ? "  SILVER PIECES";:INPUT J:SP(
P)=J:GOTO 9505
10020 ? "  COPPER PIECES";:INPUT J:CP(
P)=J:GOTO 9505
10030 ? "  PLATINUM PIECES";:INPUT J:P
P(P)=J:GOTO 9505
10040 ? "  ELECTRUM PIECES";:INPUT J:E
P(P)=J:GOTO 9505
10050 ? "  HIT POINTS";:INPUT J:HP(P)=
J:GOTO 9505
10060 ? "  WHICH WEAPON";:INPUT J:IF J
<1 OR J>5 THEN POSITION 11,22:? "
          ":POSITION 11,22:GOTO 10060
10065 W(P)=J:GOTO 9505
15000 TRAP 30000:FOR J=1 TO 40:MON(J)=
0:NEXT J:? "{CLEAR}":? "WHAT IS THE ROOM NUM
BER";:INPUT CMD$
15005 AZ$="D:COMBAT.":AZ$(10,10+LEN(CM
D$))=CMD$
15010 ? "HOW MANY MONSTERS";:INPUT J:I
F J>30 THEN ? "NO MORE THAN 30 MONSTER
S";:? :GOTO 15010
15015 MON(1)=J:? "MONSTER HIT DICE";:I
NPUT J:MON(2)=J
15020 ? "MONSTER ARMOR CLASS";:INPUT J
:MON(3)=J:FOR E=0 TO MON(1)-1:? "MONST
ER ";E;"'S HIT POINTS";:INPUT J
15030 MON(E+4)=J:NEXT E:? "NUMBER OF A
TTACKS";:INPUT J:MON(35)=J:? "DAMAGE P
ER ATTACK";:INPUT J:MON(36)=J
15040 OPEN #1,8,0,AZ$:FOR E=1 TO 40:?
#1;MON(E):NEXT E:CLOSE #1:GOTO 9000
20000 ERLN=PEEK(186)+256*PEEK(187):FOR
 E=1 TO 10:? "↑       INPUT ERROR -- TR
Y AGAIN      ";:SOUND 0,30,10,10
20001 FOR I=1 TO 2
20003 NEXT I:? "↑       INPUT ERROR --
TRY AGAIN      ";:SOUND 0,10,10,10:FOR
I=1 TO 2:NEXT I:NEXT E:SOUND 0,0,0,0
```

```
20004 ? "↑
      "
20005 TRAP 20000:GOTO ERLN
30000 IF PEEK(195)=170 THEN ? "Ⓚ":POSI
TION 2,5:? "THERE'S NO SUCH FILE ON TH
IS DISK!!"
30003 ? :? "THE FILES ARE:":?
30005 TRAP 30055:CLOSE #1:OPEN #1,6,0,
"D:*.*"
30010 INPUT #1;CMD$
30015 PRINT CMD$:GOTO 30010
30055 CLOSE #1:ERLN=PEEK(186)+256*PEEK
(187):FOR E=1 TO 200:NEXT E:TRAP 20000
:GOTO 9000
```

•

CHECKSUM DATA
(See pgs. 7-10)

```
4 DATA 621,406,367,983,822,218,174,155
,820,805,162,205,155,288,197,6378
55 DATA 49,105,192,71,986,297,476,477,
321,471,689,638,873,193,308,6146
86 DATA 660,496,27,666,904,586,644,586
,924,660,622,446,679,761,51,8712
157 DATA 686,725,351,708,148,696,494,9
14,525,199,289,188,822,713,835,8293
213 DATA 601,623,295,534,95,149,112,13
,54,880,828,98,303,164,479,5228
233 DATA 783,412,642,126,63,759,740,97
7,741,779,927,936,648,239,236,9008
310 DATA 380,863,7,104,112,634,235,155
,148,160,153,179,158,172,165,3625
1045 DATA 177,170,906,646,640,653,647,
681,654,670,664,677,671,571,796,9223
3002 DATA 875,712,894,498,547,553,723,
238,454,2,653,693,451,435,573,8301
3030 DATA 340,271,905,890,732,607,162,
297,639,290,540,49,893,651,885,8151
3520 DATA 971,107,719,361,460,227,384,
18,890,687,817,135,730,169,621,7296
4113 DATA 203,723,705,762,542,855,515,
676,146,649,710,298,173,460,483,7900
4175 DATA 666,43,203,429,808,545,943,4
42,527,329,178,545,203,751,327,6939
5002 DATA 337,347,357,367,358,368,378,
393,403,789,898,228,523,175,239,6160
5525 DATA 122,296,680,725,993,817,918,
929,921,932,924,935,927,938,931,11988
6045 DATA 942,602,890,735,449,465,891,
630,410,92,941,790,173,582,42,8634
6905 DATA 699,590,349,575,741,889,180,
186,597,518,567,937,524,105,846,8303
7015 DATA 902,139,37,213,155,370,42,89
4,971,505,739,866,79,605,781,7298
7135 DATA 687,603,622,637,770,864,570,
711,365,316,487,769,921,754,745,9821
8050 DATA 758,698,348,126,322,605,806,
243,590,402,105,928,594,769,838,8132
9000 DATA 211,948,827,394,569,481,964,
993,947,764,950,942,221,964,986,11161
9018 DATA 479,180,832,187,581,222,550,
578,747,938,406,453,362,565,475,7555
9530 DATA 484,110,918,798,704,676,785,
602,607,612,617,622,627,632,637,9431
9608 DATA 645,650,141,896,80,913,125,7
11,117,618,562,143,377,669,923,7570
9705 DATA 739,748,750,753,760,741,753,
932,709,978,571,580,589,598,607,10808
9810 DATA 597,606,615,629,638,217,869,
857,901,888,498,895,39,419,573,9241
15010 DATA 901,923,590,474,58,270,262,
196,661,376,576,687,232,728,607,7541
30055 DATA 531,531
```

•

Pretty Demo

```
10 DEG
20 GRAPHICS 24
30 COLOR 1
40 SETCOLOR 2,0,0
50 FOR I=1 TO 360 STEP 5
60 X=319*I/360
70 Y=80+80*SIN(I)
80 IF I>270 THEN 100
90 PLOT 0,0
100 DRAWTO X,Y
110 IF I<90 THEN 130
120 DRAWTO 319,159
130 NEXT I
140 IF PEEK(764)<>255 THEN END
150 GOTO 140
```

CHECKSUM DATA
(See pgs. 7-10)

```
10 DATA 217,4,724,287,58,750,133,493,4
56,967,434,363,737,152,710,6485
```

THUNDER ISLAND

32K Cassette or Disk

by Craig Patchett

One of the interesting features of the ATARI home computer is the priority register. This reserved memory location works together with the ATARI's player-missile graphics system to allow screen objects to pass behind or in front of other objects, an effect that can give the illusion of depth.

The priority register is called, appropriately, PRIOR and is found at memory location 623 ($26F hex). The following chart shows the effect of POKEing various values into it. Note that a high priority object will appear to move in front of an object with lower priority.

PRIOR=

higher priority ↑	8	4	2	1
	PFO	PFO	PO	PO
	PF1	PF1	P1	P1
	PO	PF2	PFO	P2
	P1	PF3 or P4	PF1	P3
	P2	PO	PF2	PFO
	P3	P1	PF3 or P4	PF1
	Pf2	P2	P2	PF2
	PF3 or P4	P3	P3	PF3 or P4
	BAK	BAK	BAK	BAK

Pn refers to player n
PFn refers to playfield n (as in SETCOLOR n)
PF3 or P4 refers to the fact that all missiles can be given the color of playfield 3 and used as an extra player (player 4). This is done by adding 16 to the value being POKEd into PRIOR.

When two players overlap, you can also choose to have a third color in the overlap region. This is done by adding 32 (decimal) to the value being POKEd into PRIOR.

Thunder Island uses the priority register to control which section of the maze immediately surrounding you can be seen at a given time. If you draw a maze in playfield one and set the color of playfield one to that of the background, under normal circumstances we won't be able to see the maze. But, by setting PRIOR to 2, we can have players two and three appear *between* the background and playfield one, thereby making the section of the maze "in front" of either player visible. That's all there is to it.

Playing the Game.

Thunder Island is located in the middle of the Pacific, about a thousand miles north of New Zealand. An internationally renowned playboy resort, its main attraction is a huge transparent maze. This maze can be set up to any one of an almost infinite number of floor plans, so that it is impossible to memorize the layout.

Because it is transparent, the maze is normally easy to solve. The island, however, is subject to frequent thunderstorms, and the power generator that lights the maze is often knocked out. As a precaution to this, those entering the maze carry lanterns, allowing them to at least see that part of it immediately surrounding them. It is the challenge of navigating the darkened maze, however, that has drawn you to Thunder Island. A different maze will be generated each time you play. Good Luck!

Options

Use the chart below to pick the type of game you want to play. A one-player game is good for practicing, but you'll find the two player games to be more fun. You can choose to play a daylight game, in which the whole maze is always visible, or a nighttime one, in which only part of it is visible. You can also choose from three maze difficulty levels, and each player can choose from three lantern sizes

(allowing better players to take a handicap). Once you've selected the game you want, press START and the computer will begin generating the maze. Once it's finished, your lantern(s) will light up and the game will start.

Using your joystick, you must maneuver your player to the corner of the maze diagonally opposite to the one you started at, and exit the maze. There is a timer at the bottom of the screen that keeps track of how long you've been in the maze, so you can compete for the fastest time. As soon as someone escapes, the storm will end and the maze will start reflecting a rainbow. Press START to run the program again. □

		NIGHT								DAY		
1 PLAYER		LARGE			MEDIUM			SMALL				
		1	2	3	4	5	6	7	8	9	10	11	12
		↕ PLAYER 2 ↕											
2 PLAYER		LARGE			MEDIUM			SMALL				
→ PLAYER 1 →	LARGE	1	2	3	10	11	12	13	14	15			
	MEDIUM	19	20	21	4	5	6	16	17	18			
	SMALL	22	23	24	25	26	27	7	8	9	28	29	30
		↓ E	↓ M	↓ H	↓ E	↓ M	↓ H	↓ E	↓ M	↓ H	↓ E	↓ M	↓ H

MAZE DIFFICULTY
E=EASY M=MEDIUM H=HARD
(LARGE, MEDIUM, SMALL=LANTERN SIZE)

```
100 CLR :GOTO 150
110 SOUND C0,C0,C0,C0:RETURN
120 FOR I=C1 TO 50:NEXT I:RETURN
130 D1=ASC(M$(Z,Z))-48:D2=ASC(M$(Z+C1,
Z+C1))-48
140 BYTE=HEX(D2)+C16*HEX(D1):Q2=Q2+C1:
POKE C709,PEEK(53770):RETURN
150 READ C0,C1,C2,C3,C4,C5,C6,C7,C8,C9
,C10,C11,C12,C13,C14,C16,C128,C560,C56
1,C709,C710,C711,C712
160 DIM DLI$(C13),R$(C16),M$(442),HEX(
22):GRAPHICS 18:POSITION C4,C5:? #C6;"
initializing"
170 FOR I=C1 TO C13:READ BYTE:DLI$(I)=
CHR$(BYTE):NEXT I
180 FOR I=C1 TO C16:READ BYTE:R$(I)=CH
R$(BYTE):NEXT I:FOR I=C0 TO 22:READ BY
TE:HEX(I)=BYTE:NEXT I
190 Q2=-C1:FOR I=C1 TO C3:READ M$:FOR
Z=C1 TO LEN(M$)-C1 STEP C2:GOSUB 130:P
OKE 1571+Q2,BYTE:NEXT Z:NEXT I
200 Q2=-C1:FOR I=C1 TO 21:READ M$:FOR
Z=C1 TO LEN(M$)-C1 STEP C2:GOSUB 130:P
OKE 29696+Q2,BYTE:NEXT Z:NEXT I
210 FOR X=29689 TO 29695:POKE X,C0:NEX
T X:FOR I=1536 TO 1570:POKE I,C0:NEXT
I
220 GRAPHICS C16:POKE C16,112:POKE 537
74,112
230 POKE C710,C0:ST=PEEK(C560)+256*PEE
K(C561)+C4:POKE ST+C2,C7:POKE ST+C4,C6
:POKE ST+24,65
240 POKE ST+25,PEEK(C560):POKE ST+26,P
EEK(C561)
250 POKE C708,C0:POKE C709,C0:POKE C71
1,C0
260 POKE ST+20,130:POKE 513,INT(ADR(DL
I$)/256):POKE 512,ADR(DLI$)-(PEEK(513)
*256):POKE 54286,192
270 POKE 752,C1:POSITION C3,C1:? "THUN
DER  ISLAND"
280 POSITION 22,C2:? "by craig patchet
t"
290 POSITION C3,20:? "Copyright (C)198
3 ANALOG Computing"
300 FOR X=C1 TO C3:POKE C712,C14:POKE
C710,C14:FOR Y=C0 TO 50:SOUND C0,Y,C8,
C8
310     IF Y=25 THEN POKE C710,C0:POKE
 C712,C0
320     NEXT Y:NEXT X:FOR Y=51 TO 255:
SOUND C0,Y,C8,C8:NEXT Y:POKE C712,50:P
OKE C710,50:POKE C709,C8
330 POKE 708,218:POKE C711,122:GOSUB 1
10
340 POSITION C10,C9:? "
    ":POSITION C10,C10:? " ONE PLAYER
/GAME   "
350 POSITION C10,11:? "
   "
360 SKILL=C1:LEVELP=C1
370 POSITION 27,C10:? SKILL;" ";:FOR X
=C1 TO 100:NEXT X
380 IF PEEK(53279)<>C3 THEN 440
390 LEVELP=LEVELP+C1:LEVELP=LEVELP-C2*
(LEVELP=C3):POSITION C11,C10:IF LEVELP
=C1 THEN ? "ONE";
400 SOUND C0,C10,C8,C8:GOSUB 110
410 IF LEVELP=C2 THEN ? "TWO";
420 IF LEVELP=C1 AND SKILL>C12 THEN SK
ILL=C1:GOTO 370
430 GOSUB 120:GOSUB 120
440 IF PEEK(53279)<>C5 THEN 490
450 SKILL=SKILL+C1:IF LEVELP=C1 THEN S
KILL=SKILL-C12*(SKILL=13)
460 IF LEVELP=C2 THEN SKILL=SKILL-30*(
SKILL=31)
470 SOUND C0,20,C8,C8:GOSUB 110
480 GOTO 370
490 IF PEEK(53279)<>C6 THEN 380
500 IF LEVELP=C2 THEN 540
510 LEVELD=(SKILL<C10):LEVELM=SKILL-C3
*INT((SKILL-C1)/C3):LEVELWA=C2-INT((SK
ILL-C1)/C3)
520 IF  NOT LEVELD THEN LEVELWA=C0
530 GOTO 610
540 LEVELD=(SKILL<28):LEVELM=SKILL-C3*
INT((SKILL-C1)/C3):T=INT((SKILL-C1)/C3
)
550 IF T=C0 OR T=C3 OR T=C4 THEN LEVEL
WA=C2
560 IF T=C1 OR T=C5 OR T=C6 THEN LEVEL
WA=C1
570 IF T=C2 OR T=C7 OR T=C8 THEN LEVEL
WA=C0
580 IF T=C0 OR T=C5 OR T=C7 THEN LEVEL
WB=C2
590 IF T=C1 OR T=C3 OR T=C8 THEN LEVEL
WB=C1
600 IF T=C2 OR T=C4 OR T=C6 THEN LEVEL
WB=C0
610 GRAPHICS 21:POKE C16,112:POKE 5377
4,112
620 LEVELM=80*(LEVELM=C2)+255*(LEVELM=
C3):WIDTHA=LEVELWA*C4:WIDTHB=LEVELWB*C
4:IF WIDTHA=C8 THEN WIDTHA=C12
630 IF WIDTHB=C8 THEN WIDTHB=C12
640 POKE C712,50:POKE C710,50:COLOR C3
:POKE C709,C14
650 FOR X=C0 TO 78 STEP C3:PLOT X,C0:D
RAWTO X,45:NEXT X:FOR Y=C0 TO 45 STEP
C3:PLOT C1,Y:DRAWTO 77,Y:NEXT Y
660 SOUND C0,C11,C8,C8:FOR X=C1 TO C3:
NEXT X:GOSUB 110:POKE C710,218:FOR X=C
1 TO 500:NEXT X
670 M$(C1,C1)="0":M$(442,442)="0":M$(C
2)=M$:A=INT(RND(C0)*390)+27:M$(A,A)="1
"
680 POKE 1536,LEVELM:POKE 1537,133:POK
E 1538,C1:SOUND C0,24,C4,C6:X=USR(3012
7,ADR(M$))
690 A=42*INT(RND(C0)*C2):COLOR C0:PLOT
 C0,C1+A:PLOT 78,44-A:PLOT C0,C2+A:PLO
T 78,43-A:GOSUB 110
700 M=112:POKE 1552,C1:POKE 1554,C1+A+
(A)C0):POKE 1556,WIDTHA
710 POKE 1553,77:IF LEVELP=C2 THEN POK
E 1555,44-A-(A)C0):POKE 1557,WIDTHB
720 FOR L=C0 TO C3:POKE 53248+L,C0:NEX
T L:POKE 54279,M:POKE 559,46:POKE 623,
34:POKE 53277,C3:PMB=M*256
730 POKE 53258,WIDTHA/C4:POKE 53259,WI
DTHB/C4
```

```
740 T=PMB+512:FOR L=T TO T+511:POKE L,
C0:NEXT L:L=A*C2+C2*(A>C0):POKE T+L+18
,24:POKE T+L+19,24
750 L=84-A*C2+C2*(A=C0):POKE T+L+146,2
4:POKE T+L+147,24
760 L=PMB+A*C2+C2*(A>C0)+768+15-C4*LEV
ELWA:FOR X=L TO L+C7+C8*LEVELWA:POKE X
,255:NEXT X
770 L=PMB+84-A*C2+C2*(A=C0)+896+15-C4*
LEVELWB:FOR X=L TO L+C7+C8*LEVELWB:POK
E X,255:NEXT X
780 POKE 53248,47
790 SOUND C0,C11,C8,C8:FOR X=C1 TO C3:
NEXT X:GOSUB 110:POKE 704,C14:GOSUB 12
0:POKE 53250,47-WIDTHA
800 FOR X=C16 TO C0 STEP -C1:SOUND C0,
X,C8,C8:POKE 706,C16-X:FOR Y=C1 TO C10
:NEXT Y:NEXT X:GOSUB 110:POKE 706,72
810 IF LEVELP<>C2 THEN 850
820 GOSUB 120:GOSUB 120:POKE 53249,199
:SOUND C0,C11,C8,C8:FOR X=C1 TO C3:NEX
T X:GOSUB 110:POKE 705,C14:GOSUB 120
830 POKE 53251,199-WIDTHB
840 FOR X=C16 TO C0 STEP -C1:SOUND C0,
X,C8,C8:POKE 707,C16-X:FOR Y=C1 TO C10
:NEXT Y:NEXT X:GOSUB 110:POKE 707,24
850 FOR X=C1 TO 200:NEXT X:POKE C712,C
14:FLASH=C0
860 IF LEVELD THEN POKE C710,144
870 ST=PEEK(C560)+256*PEEK(C561)+C4:PO
KE ST+47,C2:POKE ST+48,65:POKE ST+49,P
EEK(C560):POKE ST+50,PEEK(C561)
880 X=USR(1571):SOUND C3,C10,C8,C2
890 X=USR(29696)
900 IF FLASH<255 THEN 930
910 IF INT(RND(C0)*200)<>100 THEN 950
920 POKE C712,C14:FLASH=C0:SOUND C2,C0
,C8,15
930 FLASH=FLASH+C5:SOUND C2,FLASH,C8,C
8:IF FLASH=255 THEN SOUND C2,C0,C0,C0:
REM
940 IF FLASH=25 THEN SETCOLOR C4,C9-C6
*(LEVELD=C0),C2*(LEVELD=C0)
950 IF PEEK(1560) OR PEEK(1561) THEN 9
90
960 IF PEEK(1558) THEN SOUND C0,20,C8,
C8
970 IF PEEK(1559) THEN SOUND C0,40,C8,
C8
980 FOR X=C1 TO C5:NEXT X:GOSUB 110:GO
TO 890
990 X=USR(1703):SOUND C2,C0,C0,C0
1000 IF PEEK(1560) THEN POKE 53249,C0:
POKE 53251,C0
1010 IF PEEK(1561) THEN POKE 53248,C0:
POKE 53250,C0
1020 X=USR(ADR(R$)):FOR X=53248 TO 532
51:POKE X,C0:NEXT X:GOTO 210
1030 REM * CONSTANTS
1040 DATA 0,1,2,3,4,5,6,7,8,9,10,11,12
,13,14,16,128,560,561,709,710,711,712
1050 REM * DLI ROUTINE
1060 DATA 72,169,14,141,10,212,141,23,
208,169,88,104,64
1070 REM * RAINBOW ROUTINE
1080 DATA 104,169,6,232,142,10,212,142
,24,208,205,31,208,208,242,96
1090 REM * HEX DATA
1100 DATA 0,1,2,3,4,5,6,7,8,9,0,0,0,0,
0,0,0,10,11,12,13,14,15
1110 REM * TIMER ROUTINE
1120 DATA 6818A9A9655885CDA903655985CE
A93C8D2202A9068D230260A514C906B0034C5F
E4A9008514EE1F06AD1F06C90AD032A900
1130 DATA 8D1F06EE2006AD2006C90AD023A9
008D2006EE2106AD2106C906D014A9008D2106
EE2206AD2206C90AD005A9008D2206A000
1140 DATA A204C001F004C004D008A91A91CD
C81890F0BD1E06491091CDC8CAD0E54C5FE468
AD60E48D2202AD61E48D230260
1150 REM * P/M-STICK ROUTINE
1160 DATA 68A2008D1E06AE1E06A9009D1606
9D1806BD10068D1A06BD12068D1B06BD78024A
9003EE1B064A489003CE1B06205B74AD1E
1170 DATA 06AABD10068D1A06BD12068D1B06
684A9003EE1A064A9003CE1A06205B74EE1E06
AD1E06C902D0AC60A55885CBA55985CCAD
1180 DATA 1A064A4A184865CB85CBA90065CC
85CC6880A8D0E06A903186D0E0638ED1A068D
0E06A9008D1D06AD1B06A2040A2E1D06E0
1190 DATA 03D00B8D1C06AD1D0648AD1C0648
CAD0EA8D1C0668186D1C068D1C06686D1D068D
1D0618A5CB6D1C0685CBA5CC6D1D0685CC
1200 DATA A000B1CBAE0E064A4AA000B002A0
FFC8CAE0FFD0F2C001D00160A90138ED1E06A8
AD1E06AAB91206CD1B06D009B91006CD1A
1210 DATA 06D00160AD1E06A8208775AD1A06
DD1006F011FE16060A18692D9D00D038FD1406
9D02D0AD1B06DD1206F052FE1606902A2
1220 DATA 7DC001D00FBD80729D8272BD8073
9D827318900CBD00729D0272BD00739D0273CA
D0DE189026A202C001D00FBD80729D7E72
1230 DATA BD80739D7E7318900CBD00729DFE
71BD00739DFE72E8E07FD0DC98AAAD1B069D12
06AD1A069D100660AD1A06E000D010C900
1240 DATA D004EE1A0660C94ED014FE180660
C94ED004CE1A0660C900D004FE180660606868
8D0806688D0706AD0AD2290FC90FF0F78D
1250 DATA 0306AD0AD2291FC91A10F78D0406
200577B1CDC931D0DFA9008D0906AD0406C901
9005A0192062770E0906AD0406C919B005
1260 DATA A01B2062770E0906AD0306C90190
05A0002062770E0906AD0306C90EB005A03420
6277AD0906C900F09AAD04060A186D0406
1270 DATA 8D0A06AD0306A186D03068D0B06
A514F0FCA9008514A9148D00D2AD0AD229038D
0F06C900D019AD09062908F019CE0406EE
1280 DATA 0B06206C77EE0B06206C774CCE76
AD0F06C901D022AD09062904F022EE0406EE0A
06EE0A06EE0A06EE0B06206C77EE0B0620
1290 DATA 6C774CCE76AD0F06C902D019AD09
062902F012CE0306EE0A06206C77EE0A06206C
774CCE76AD09062901F097EE0306EE0B06
1300 DATA EE0B06EE0B06EE0A06206C77EE0A
06206C77A9188D00D2200577A93191CD38AD01
06E9018D0106AD0206E9008D0206C900D0
1310 DATA 08AD0106C900D00160AD0AD2CD00
0690034CD9754CB875AD03068D050648AD0406
A2008E0606A2040E05062E0606E002D008
1320 DATA AD060648AD050648CAD0EB18686D
05068D0506686D06068D0606680A186D04066D
05068D0506AD0606690B8D0606618AD0706
1330 DATA 6D050685CDAD08066D060685CEA0
1A60B1CDC930D003EE090660A55885CBA55985
CCAD0A064A4A184865CB85CBA90065CC85
1340 DATA CC680A0A8D0E0638AD0A06ED0E06
AAE8A93FCAF006386A6A4C96778D0E06A9008D
0C06AD0B06A2040A2E0C06E003D00B8D0D
1350 DATA 06AD0C0648AD0D0648CAD0EA8D0D
0668186D0D068D0D06686D0C068D0C0618A5CB
6D0D0685CBA5CC6D0C0685CCA000B1CB2D
1360 DATA 0E0691CB60
```

•

CHECKSUM DATA

(See pgs. 7-10)

```
100 DATA 120,824,516,417,482,48,674,35
4,803,435,271,537,335,77,927,6820
250 DATA 232,612,562,820,582,532,611,4
31,113,568,211,363,497,995,287,7416
400 DATA 940,908,982,918,994,566,121,1
86,738,7,422,389,847,713,37,8768
550 DATA 838,848,858,856,857,833,496,5
31,266,652,915,829,350,519,627,10275
700 DATA 594,213,595,989,905,613,15,33
1,147,142,889,360,64,498,903,7258
850 DATA 420,48,178,537,388,345,546,54
3,333,350,64,69,76,671,347,4915
1000 DATA 63,64,263,703,753,175,959,51
9,126,670,358,277,523,395,765,6613
1150 DATA 39,474,703,648,869,749,454,5
68,775,720,445,152,388,589,603,8176
1300 DATA 561,372,328,695,672,781,376,
3785
```

MANIAC!

32K Cassette or Disk

by Rick Messner

Over the past several years, many game programs have been developed for the ATARI computers. Unfortunately, most of these games cost from $30 to $40 apiece. Taking pity on those who, like myself, cannot afford to buy all those great games, I programmed an arcade-style game called **Maniac!** This fast-action Assembly language game is yours for the price of a few hours of typing. You're going to like this one!

The game.

Maniac! is set in a maze with eight levels. Each level is filled with crazed robots. These robots were once peaceful gardeners but became short-circuited by pesticides, and are now trying to destroy anything that moves with their missile-firing shovels. You are dropped into the first level of this maze, equipped only with your trusty .45 and ammunition. Your job is to stop all the robots on each level.

At first, your task is not particularly difficult, but as you enter the higher-numbered maze levels the job gets harder and harder. That's because the robots on the higher-numbered maze levels have been around longer and are covered with a protective layer of earth and bullrushes.

To start.

At the beginning of this game there is a short introduction (just to help you learn my name), and then the machine asks you if you are going to have a one or two-player game. Enter your choice. A light grey maze appears on your monitor screen, along with a green figure representing you and three red figures representing the robots. At the end of the maze opposite from where you start is a door. Your objective is to destroy all three robots and to run out the exit door. If you do this, the computer automatically advances you to the next level of the maze. If, however, you exit without destroying all the robots, you remain at your current level and must try again.

Movement.

The computer moves the robots in a one-person game. In a two-person game, one player controls the hunter, and the other player controls the robots. Moving either the hunter or the robots is very simple. Hold the joystick in the normal position and push it in the direction you wish to move.

If a robot and the hunter collide, either one or both is blown up. If either a robot or a hunter walks into a wall, it will explode.

Earth and bulrushes.

Robots on the higher-numbered maze levels become increasingly covered with a layer of earth and bulrushes. This makes it rather hard for the hunter's .45 to hit one. To kill a robot, the hunter must shoot it once for each level of the maze. Thus, on level six, the hunter must shoot each robot six times before it is destroyed.

2-player version.

In the two-player game, one person controls the hunter, and the other person controls the robots. The hunter is controlled by the joystick in port zero, and the robots are moved by the joystick in port one.

To start, the person with the robots moves only the leftmost robot. If it is killed, control switches to the center robot. When that one is killed, control moves to the rightmost robot.

As the player controlling the robots becomes more skilled he may want to switch control from robot to robot at will. This can be done by using one of three keyboard keys. To make the joystick control the leftmost robot, press the semi-colon key; to make the joystick control the center robot, press the plus sign key; to make the joystick control the rightmost robot, press the multiplication sign key.

Firing.

Both the hunter and the robot can fire in many different directions. To rotate the arm — and the

weapon — of either the hunter or the robot, do this: press the joystick button and hold it down. While the button is depressed, you may move the joystick in any direction — this will cause the arm to rotate. As soon as you let go of the red button, the weapon will fire.

If you wish to fire without moving the direction of the weapon, simply press and release the red button on your joystick.

Scoring.

Points are awarded to the hunter but not to the robots. The robots get their pleasure solely from frustrating the hunter. Scoring is as follows: the hunter gets ten points per level for each robot destroyed, with a 1,000-point bonus for making it through all eight levels.

Typing the program.

Two program listings follow this article. **Listing 1** is the main data and data checking routine, and will create the cassette version of **Maniac! Listing 2** should be added to **Listing 1** for disk users.

Cassette instructions.

1. Type **Listing 1** into your computer and check it for typing errors by using C:CHECK.

2. Type RUN and press RETURN. The program will check for errors in the DATA lines. If any error messages are displayed, correct the lines indicated and re-RUN the program until all errors are eliminated.

3. When all the DATA statements are correct, the program will ask you to "READY CASSETTE AND PRESS RETURN" and the console will BEEP twice. Place a cassette in your program recorder, press RECORD and PLAY simultaneously, then press RETURN. The computer will create a boot tape containing the **Maniac!** game. It is a good idea to CSAVE the BASIC program at this time, just in case you want to use it again later.

4. To play **Maniac!**, remove any cartridges from your system. Place the **Maniac!** boot tape in your program recorder, rewind it to the beginning and press PLAY. Turn the computer's power OFF, then turn it ON while pressing START. The computer will BEEP. Press RETURN and the game will load and run automatically.

Disk instructions.

1. Type **Listing 1** into your computer and check it for typing errors with D:CHECK. After correcting any typing errors, enter the lines with **Listing 2**. These lines will merge with **Listing 1** in order to create a **Maniac!** disk version maker program.

2. Type RUN and press RETURN. The program will check for any errors in the DATA statements and will display a message if any errors are found. Correct any DATA lines that are in error and re-RUN the program until all errors are corrected.

3. When the computer has made sure there are no errors in the DATA lines, it will ask "INSERT DISK WITH DOS, PRESS RETURN." Place a disk with DOS in drive 1 and press RETURN. The program will create an AUTORUN.SYS file containing the **Maniac!** game. When the READY prompt appears, the program is finished. It is a good idea to SAVE the **Maniac!** BASIC program just in case you need it later.

4. To play **Maniac!**, place the disk containing the AUTORUN.SYS file in drive 1. Remove any cartridges and turn the computer's power OFF, then ON. The **Maniac!** game will automatically load and start. □

```
10 REM MANIAC CASSETTE MAKER PROGRAM
20 REM
30 CLR :DIM X$(3984):Q=0:LINE=4990:RES
TORE 5000
40 P=0
50 LINE=LINE+10:? "CHECKING LINE ";LIN
E
60 FOR I=1 TO 16
70 Q=Q+1
80 TRAP 140:READ J:IF I=1 THEN IF LINE
<>PEEK(183)+PEEK(184)*256 THEN ? "LINE
 ";LINE;" MISSING!":END
90 IF J=999 THEN 150
100 X$(Q)=CHR$(J)
110 P=P+J
120 NEXT I
130 TRAP 140:READ J:IF P=J THEN 40
140 ? "ERROR IN LINE ";LINE:STOP
150 ? "READY CASSETTE AND PRESS RETURN
":OPEN #1,8,0,"C:"
160 ? #1;X$
170 CLOSE #1
5000 DATA 0,31,0,64,35,64,169,60,141,2
,211,169,119,141,231,2,1439
5010 DATA 133,14,169,79,141,232,2,133,
15,169,38,133,10,169,64,133,1634
5020 DATA 11,24,96,96,83,58,32,28,76,3
2,55,72,162,255,141,30,1251
5030 DATA 208,232,224,4,240,246,142,18
4,79,169,2,141,187,79,32,148,2317
5040 DATA 75,172,185,79,185,232,76,133
,176,133,180,185,233,76,133,177,2430
5050 DATA 133,181,189,127,79,24,201,3,
176,18,32,210,64,32,184,65,1718
5060 DATA 32,38,68,32,173,70,32,223,68
,32,17,71,189,159,79,201,1484
5070 DATA 0,240,6,32,108,70,32,179,69,
32,108,65,169,11,24,237,1382
5080 DATA 191,79,141,188,79,169,2,141,
190,79,32,189,75,173,188,79,1995
5090 DATA 141,181,79,32,61,65,32,253,7
3,32,178,64,32,180,71,32,1506
5100 DATA 248,71,173,127,79,201,7,240,
3,76,49,64,32,127,72,76,1645
5110 DATA 44,64,173,252,2,141,0,104,17
3,200,79,201,1,208,1,96,1739
5120 DATA 173,252,2,201,7,240,46,201,6
,240,34,201,2,240,22,76,1943
5130 DATA 250,64,189,135,79,24,233,47,
157,173,79,189,139,79,24,233,2094
5140 DATA 16,157,177,79,96,169,1,141,1
95,79,76,250,64,169,2,141,1812
5150 DATA 195,79,76,250,64,169,3,141,1
95,79,172,195,79,185,127,79,2088
5160 DATA 24,201,2,176,1,96,173,207,79
,24,201,0,176,5,169,2,1536
5170 DATA 141,207,79,206,207,79,173,20
7,79,201,1,240,1,96,169,0,2086
5180 DATA 141,207,79,160,1,185,127,79,
24,201,2,144,7,200,152,201,1910
5190 DATA 4,208,242,96,140,195,79,169,
255,141,252,2,96,142,182,79,2282
```

```
5200 DATA 140,183,79,162,255,232,236,1
81,79,240,11,160,255,200,152,201,2766
5210 DATA 255,208,250,76,69,65,174,182
,79,172,183,79,96,142,182,79,2291
5220 DATA 162,255,232,236,181,79,208,2
50,174,182,79,96,189,127,79,24,2553
5230 DATA 201,2,176,1,96,24,201,7,144,
1,96,24,105,10,141,187,1416
5240 DATA 79,169,15,141,184,79,32,148,
75,173,185,79,24,109,230,76,1798
5250 DATA 133,178,173,186,79,109,231,7
6,133,179,165,176,24,125,139,79,2185
5260 DATA 133,176,165,177,105,0,133,17
7,160,0,177,178,145,176,200,152,2254
5270 DATA 201,15,208,246,254,127,79,96
,189,131,79,201,1,208,1,96,2132
5280 DATA 224,0,240,14,236,195,79,240,
3,76,216,65,173,121,2,76,1960
5290 DATA 38,67,189,120,2,76,38,67,169
,1,157,131,79,173,191,79,1577
5300 DATA 141,187,79,169,10,141,184,79
,32,148,75,173,10,210,24,205,1867
5310 DATA 185,79,144,1,96,189,139,79,2
4,205,139,79,144,50,24,233,1810
5320 DATA 1,24,205,139,79,176,35,169,1
5,141,114,3,189,135,79,24,1528
5330 DATA 205,135,79,144,15,24,205,135
,79,176,3,76,38,67,32,163,1576
5340 DATA 66,76,38,67,32,224,66,76,38,
67,32,54,66,76,9,66,1053
5350 DATA 32,110,66,76,9,66,189,173,79
,133,85,189,177,79,24,233,1720
5360 DATA 0,133,84,32,36,70,160,0,177,
178,201,0,208,29,230,85,1623
5370 DATA 32,36,70,173,181,79,201,0,20
8,17,198,85,198,85,32,36,1631
5380 DATA 70,173,181,79,201,0,208,3,16
9,14,96,169,15,96,189,173,1836
5390 DATA 79,133,85,189,177,79,24,105,
17,133,84,32,36,70,160,0,1403
5400 DATA 177,178,201,0,208,229,230,85
,32,36,70,173,181,79,201,0,2080
5410 DATA 208,217,230,85,230,85,32,36,
70,173,181,79,201,0,208,203,2238
5420 DATA 169,13,96,189,173,79,133,85,
189,177,79,133,84,32,36,70,1737
5430 DATA 169,16,141,181,79,160,0,140,
183,79,160,0,177,178,201,0,1864
5440 DATA 208,94,165,178,24,109,193,79
,133,178,165,179,105,0,133,179,2122
5450 DATA 172,183,79,200,152,205,181,7
9,208,221,173,114,3,41,11,96,2118
5460 DATA 189,173,79,24,105,8,133,85,1
89,177,79,133,84,32,36,70,1596
5470 DATA 169,16,141,181,79,160,0,140,
183,79,160,0,177,178,201,0,1864
5480 DATA 208,30,165,178,24,109,193,79
,133,178,165,179,105,0,133,179,2058
5490 DATA 172,183,79,200,152,205,181,7
9,208,221,173,114,3,41,7,96,2114
5500 DATA 173,114,3,41,15,96,201,14,24
0,29,201,6,240,59,201,7,1640
5510 DATA 240,33,201,5,240,65,201,13,2
40,19,201,9,240,71,201,11,1990
5520 DATA 240,28,201,10,240,77,96,32,2
22,67,76,161,67,32,0,68,1617
5530 DATA 76,161,67,169,1,157,127,79,2
54,135,79,76,161,67,169,0,1778
5540 DATA 157,127,79,222,135,79,76,161
,67,32,222,67,254,135,79,169,2061
5550 DATA 1,157,127,79,76,161,67,32,0,
68,254,135,79,169,1,157,1563
5560 DATA 127,79,76,161,67,32,0,68,222
,135,79,169,0,157,127,79,1578
5570 DATA 76,161,67,32,222,67,222,135,
79,169,0,157,127,79,76,161,1830
5580 DATA 67,32,189,67,189,135,79,157,
0,208,189,119,79,201,1,240,1952
5590 DATA 6,169,1,157,119,79,96,169,0,
157,119,79,96,189,135,79,1650
5600 DATA 24,201,208,176,6,24,201,48,1
44,14,96,169,208,157,135,79,1890
5610 DATA 224,0,208,3,76,228,74,96,169
,48,157,135,79,96,189,139,1921
5620 DATA 79,24,201,13,144,25,222,139,
79,24,105,16,141,181,79,188,1660
5630 DATA 139,79,177,176,136,145,176,2
00,200,152,205,181,79,208,243,96,2592
5640 DATA 189,139,79,24,201,96,176,29,
24,105,16,168,177,176,200,145,1944
5650 DATA 176,136,136,189,139,79,24,23
3,3,141,181,79,152,205,181,79,2133
5660 DATA 208,234,254,139,79,96,189,13
1,79,201,1,240,1,96,224,0,2172
5670 DATA 240,57,236,195,79,240,46,169
,0,157,131,79,189,139,79,24,2060
5680 DATA 233,14,24,205,139,79,176,14,
24,105,22,205,139,79,144,12,1614
5690 DATA 169,1,157,123,79,96,169,0,15
7,123,79,96,169,2,157,123,1700
5700 DATA 79,96,76,110,68,173,121,2,76
,110,68,189,120,2,201,6,1497
5710 DATA 201,6,240,62,201,10,240,25,2
01,7,240,65,201,11,240,28,1978
5720 DATA 201,5,240,68,201,9,240,31,20
1,14,240,71,201,13,240,73,2048
5730 DATA 96,169,0,157,123,79,169,0,15
7,127,79,96,169,1,157,123,1702
5740 DATA 79,169,0,157,127,79,96,169,2
,157,123,79,169,0,157,127,1690
5750 DATA 79,96,169,0,157,123,79,169,1
,157,127,79,96,169,1,157,1659
5760 DATA 123,79,169,1,157,127,79,96,1
69,2,157,123,79,169,1,157,1688
5770 DATA 127,79,96,169,0,157,123,79,9
6,169,2,157,123,79,96,224,1776
5780 DATA 0,240,38,236,195,79,240,27,1
73,191,79,141,187,79,169,10,2084
5790 DATA 141,184,79,32,148,75,173,10,
210,24,205,185,79,144,1,96,1786
5800 DATA 76,30,69,173,133,2,76,12,69,
189,132,2,201,0,240,8,1412
5810 DATA 189,131,79,201,1,240,7,96,16
9,1,157,131,79,96,189,159,1925
5820 DATA 79,201,0,240,1,96,169,1,141,
196,79,169,0,157,131,79,1739
5830 DATA 189,127,79,201,1,240,8,169,2
54,157,151,79,76,68,69,169,2037
5840 DATA 2,157,151,79,189,123,79,201,
0,240,9,201,1,240,13,201,1886
5850 DATA 2,240,17,96,169,255,157,155,
79,76,105,69,169,0,157,155,1901
5860 DATA 79,76,105,69,169,1,157,155,7
9,169,35,157,159,79,189,127,1805
5870 DATA 79,141,187,79,169,3,141,184,
79,32,148,75,173,185,79,24,1778
5880 DATA 125,123,79,141,184,79,169,2,
141,187,79,32,148,75,173,185,1922
5890 DATA 79,168,185,241,77,24,125,135
,79,157,143,79,185,242,77,24,2020
5900 DATA 125,139,79,157,147,79,188,14
7,79,189,110,78,25,128,105,153,1928
5910 DATA 128,105,96,222,159,79,188,14
7,79,189,110,78,89,128,105,153,2055
5920 DATA 128,105,152,24,125,155,79,15
7,147,79,189,159,79,201,0,240,2019
5930 DATA 74,189,143,79,24,125,151,79,
157,143,79,24,233,47,133,85,1765
5940 DATA 189,147,79,24,233,16,133,84,
32,36,70,173,181,79,24,201,1701
5950 DATA 0,208,31,230,85,32,36,70,173
,181,79,201,0,208,19,188,1741
5960 DATA 147,79,189,110,78,25,128,105
,153,128,105,189,143,79,157,4,1819
5970 DATA 208,96,169,0,157,159,79,157,
4,208,96,169,0,157,143,79,1881
5980 DATA 157,4,208,96,140,183,79,165,
84,141,187,79,173,193,79,141,2109
5990 DATA 184,79,32,148,75,165,85,141,
188,79,169,4,141,190,79,32,1791
6000 DATA 189,75,173,185,79,24,109,188
,79,141,185,79,173,186,79,101,2045
6010 DATA 89,133,179,173,185,79,24,101
,88,133,178,165,179,105,0,133,1944
6020 DATA 179,160,0,177,178,141,181,79
,172,183,79,96,189,143,79,24,2060
6030 DATA 201,208,176,19,24,201,44,144
,27,189,147,79,24,201,112,176,1972
6040 DATA 32,24,201,13,144,33,96,169,1
,157,159,79,169,208,157,143,1785
6050 DATA 79,76,121,70,169,1,157,159,7
9,169,44,157,143,79,76,121,1700
6060 DATA 70,169,1,157,159,79,96,169,1
,157,159,79,96,189,127,79,1787
6070 DATA 141,184,79,169,3,141,187,79,
32,148,75,173,185,79,24,125,1824
```

```
6080 DATA 123,79,141,181,79,189,119,79
,141,184,79,169,6,141,187,79,1976
6090 DATA 32,148,75,173,185,79,24,109,
181,79,141,187,79,169,15,141,1817
6100 DATA 184,79,32,148,75,173,185,79,
24,109,230,76,133,178,173,186,2064
6110 DATA 79,109,231,76,133,179,165,18
0,24,125,139,79,133,180,165,181,2178
6120 DATA 105,0,133,181,160,0,177,178,
145,180,200,152,201,15,208,246,2281
6130 DATA 96,189,4,208,24,201,0,208,73
,189,12,208,24,201,0,208,1845
6140 DATA 65,160,0,142,181,79,152,24,2
05,181,79,240,39,189,135,79,1950
6150 DATA 24,217,143,79,176,30,24,105,
8,24,217,143,79,144,21,189,1623
6160 DATA 139,79,24,217,147,79,176,12,
24,105,15,24,217,147,79,144,1628
6170 DATA 3,76,98,71,200,152,24,201,4,
208,200,96,169,1,157,159,1819
6180 DATA 79,96,222,163,79,169,1,153,1
59,79,189,163,79,24,201,0,1856
6190 DATA 240,1,96,169,1,141,197,79,16
9,2,157,127,79,152,201,0,1811
6200 DATA 240,9,138,201,0,240,31,206,1
94,79,96,173,191,79,141,184,2202
6210 DATA 79,169,10,141,187,79,32,148,
75,173,185,79,141,181,79,32,1790
6220 DATA 159,73,206,194,79,96,206,192
,79,173,192,79,201,0,240,1,2170
6230 DATA 96,76,83,74,173,196,79,201,2
,240,19,201,0,240,48,169,1897
6240 DATA 2,141,196,79,160,0,140,198,7
9,169,175,141,1,210,172,198,2061
6250 DATA 79,200,140,0,210,200,140,0,2
10,200,140,0,210,200,140,0,2069
6260 DATA 210,200,140,0,210,152,24,201
,240,176,4,140,198,79,96,169,2239
6270 DATA 0,141,196,79,141,1,210,96,14
0,183,79,173,197,79,201,0,1916
6280 DATA 240,41,201,2,240,15,169,2,14
1,197,79,169,143,141,3,210,1993
6290 DATA 169,0,141,199,79,172,199,79,
140,2,210,200,140,2,210,152,2094
6300 DATA 201,240,176,7,140,199,79,172
,183,79,96,169,0,141,197,79,2158
6310 DATA 141,3,210,172,183,79,96,169,
7,141,181,79,32,82,73,169,1817
6320 DATA 46,141,47,2,169,3,141,29,208
,169,200,141,192,2,169,56,1715
6330 DATA 141,193,2,141,194,2,141,195,
2,169,0,141,197,79,141,196,1934
6340 DATA 79,169,104,141,7,212,160,1,1
40,191,79,169,3,141,192,79,1867
6350 DATA 169,16,162,0,157,167,79,232,
224,6,208,248,32,43,74,169,1986
6360 DATA 0,141,1,210,141,0,210,141,3,
210,141,2,210,141,207,79,1837
6370 DATA 173,191,79,24,201,8,176,21,1
68,185,110,79,141,196,2,169,1923
6380 DATA 0,141,198,2,141,197,79,141,1
96,79,32,114,73,172,191,79,1835
6390 DATA 140,184,79,169,4,141,187,79,
32,148,75,169,3,141,194,79,1824
6400 DATA 173,200,79,201,2,240,8,169,4
,141,195,79,76,217,72,169,2025
6410 DATA 1,141,195,79,169,2,141,252,2
,169,40,141,193,79,173,185,1962
6420 DATA 79,24,233,3,168,24,105,4,141
,181,79,162,0,185,46,78,1512
6430 DATA 157,0,208,157,135,79,185,78,
78,157,139,79,173,191,79,157,2052
6440 DATA 163,79,169,0,157,131,79,157,
143,79,157,159,79,200,232,152,2136
6450 DATA 205,181,79,208,216,169,1,141
,163,79,160,0,169,0,153,128,2052
6460 DATA 105,153,0,106,153,128,106,15
3,0,107,153,128,107,200,152,201,1952
6470 DATA 128,208,233,169,1,141,123,79
,141,124,79,141,125,79,141,126,2038
6480 DATA 79,169,1,141,127,79,169,0,14
1,128,79,141,129,79,141,130,1733
6490 DATA 79,96,169,3,141,114,3,169,36
,141,116,3,169,64,141,117,1561
6500 DATA 3,169,28,141,122,3,173,181,7
9,141,123,3,162,48,32,86,1494
6510 DATA 228,96,160,0,185,253,77,24,2
01,255,240,34,133,85,200,185,2356
6520 DATA 253,77,133,84,169,85,32,221,
75,200,185,253,77,133,85,200,2262
6530 DATA 185,253,77,133,84,169,85,32,
2,76,200,76,116,73,96,142,1799
6540 DATA 182,79,162,0,160,0,185,167,7
9,24,105,1,9,16,153,167,1489
6550 DATA 79,201,26,208,11,169,16,153,
167,79,200,152,201,6,208,230,2106
6560 DATA 232,24,236,181,79,208,221,17
4,182,79,32,206,73,96,160,86,2269
6570 DATA 173,48,2,133,180,173,49,2,13
3,181,177,180,24,105,7,133,1700
6580 DATA 178,200,177,180,133,179,142,
182,79,162,5,160,0,189,167,79,2212
6590 DATA 145,178,200,202,152,201,6,20
8,244,174,182,79,96,160,86,173,2486
6600 DATA 48,2,133,180,173,49,2,133,18
1,177,180,24,105,45,133,178,1743
6610 DATA 200,177,180,133,179,160,0,16
9,65,145,178,200,152,201,10,240,2389
6620 DATA 5,205,192,79,208,241,169,0,1
45,178,96,160,86,173,48,2,1987
6630 DATA 133,180,173,49,2,133,181,177
,180,133,178,200,177,180,133,179,2388
6640 DATA 160,0,185,118,78,201,255,240
,9,24,233,31,145,178,200,76,2133
6650 DATA 66,74,96,32,194,76,169,0,141
,181,79,32,82,73,169,0,1464
6660 DATA 141,198,2,170,157,0,208,157,
4,208,157,1,210,157,0,210,1980
6670 DATA 232,224,4,208,239,160,0,185,
37,79,24,201,255,240,9,24,2121
6680 DATA 233,31,145,88,200,76,119,74,
160,12,162,0,189,167,79,145,1880
6690 DATA 88,136,232,224,6,208,245,32,
184,74,160,30,162,0,189,201,2171
6700 DATA 79,145,88,136,232,224,6,208,
245,169,255,141,252,2,173,252,2607
6710 DATA 2,201,255,240,249,76,38,64,1
60,5,185,167,79,24,233,0,1978
6720 DATA 24,217,201,79,176,15,24,105,
1,217,201,79,144,6,136,152,1777
6730 DATA 201,255,208,230,96,160,0,185
,167,79,153,201,79,200,152,201,2567
6740 DATA 6,208,244,96,162,0,169,0,157
,4,208,157,0,208,157,0,1776
6750 DATA 210,157,1,210,232,224,4,208,
239,32,54,75,173,194,79,24,2116
6760 DATA 201,0,208,3,238,191,79,173,1
91,79,24,201,8,144,33,169,1942
6770 DATA 1,141,191,79,160,0,169,200,1
41,181,79,140,183,79,32,159,1935
6780 DATA 73,172,183,79,200,140,183,79
,152,201,5,208,241,238,192,79,2425
6790 DATA 32,127,72,76,44,64,169,175,1
41,1,210,162,0,165,88,133,1659
6800 DATA 178,165,89,133,179,32,86,75,
232,224,16,208,240,169,0,141,2167
6810 DATA 1,210,141,0,210,96,142,182,7
9,162,0,32,126,75,165,178,1799
6820 DATA 24,109,193,79,133,178,165,17
9,105,0,133,179,238,199,79,173,2166
6830 DATA 199,79,141,0,210,232,224,79,
208,225,174,182,79,96,160,0,2288
6840 DATA 177,178,24,42,145,178,177,17
8,24,42,145,178,200,152,205,193,2238
6850 DATA 79,208,237,96,169,0,141,185,
79,141,186,79,142,182,79,162,2165
6860 DATA 8,10,46,186,79,14,187,79,144
,9,24,109,184,79,144,3,1305
6870 DATA 238,186,79,202,208,235,141,1
85,79,174,182,79,96,142,182,79,2487
6880 DATA 169,0,162,8,14,188,79,42,205
,190,79,144,6,237,190,79,1792
6890 DATA 238,188,79,202,208,238,141,1
89,79,174,182,79,96,141,181,79,2494
6900 DATA 142,182,79,140,183,79,169,11
,141,114,3,169,0,141,120,3,1676
6910 DATA 141,121,3,162,48,173,181,79,
32,86,228,174,182,79,172,183,2044
6920 DATA 79,96,141,251,2,142,182,79,1
40,183,79,169,17,141,114,3,1818
6930 DATA 162,48,32,86,228,174,182,79,
172,183,79,96,32,194,76,169,1992
6940 DATA 2,141,181,79,32,82,73,169,0,
141,198,2,165,88,24,105,1482
6950 DATA 61,133,178,165,89,105,0,133,
179,160,0,185,163,78,201,255,2085
```

```
6960 DATA 240,9,24,233,31,145,178,200,
76,59,76,32,205,76,32,205,1821
6970 DATA 76,32,205,76,32,205,76,32,20
5,76,169,255,141,181,79,32,1872
6980 DATA 61,65,32,61,65,32,61,65,32,6
1,65,32,61,65,32,61,851
6990 DATA 65,32,194,76,169,2,141,181,7
9,32,82,73,169,0,141,198,1634
7000 DATA 2,165,88,24,105,40,133,178,1
65,89,133,179,160,0,185,218,1864
7010 DATA 78,201,255,240,9,24,233,31,1
45,178,200,76,142,76,32,194,2114
7020 DATA 76,169,255,141,252,2,173,252
,2,24,201,30,240,14,24,201,2056
7030 DATA 31,240,3,76,166,76,169,1,141
,200,79,96,169,2,141,200,1790
7040 DATA 79,96,169,12,141,114,3,162,4
8,32,86,228,96,169,1,141,1577
7050 DATA 196,79,32,180,71,169,2,141,1
81,79,32,61,65,173,196,79,1736
7060 DATA 201,0,208,238,96,51,242,76,0
,106,128,106,0,107,128,107,1794
7070 DATA 128,105,140,92,76,44,28,12,1
2,12,12,12,4,4,4,4,689
7080 DATA 12,12,28,12,204,124,12,12,12
,12,12,4,4,4,4,12,480
7090 DATA 12,28,12,12,28,108,204,12,12
,12,4,4,4,4,12,49,517
7100 DATA 58,50,52,56,48,48,48,48,48,3
2,32,32,32,48,48,56,736
7110 DATA 48,51,62,48,48,48,48,48,32,3
2,32,32,48,48,56,48,729
7120 DATA 48,56,54,51,48,48,48,32,32,3
2,32,48,140,92,76,44,881
7130 DATA 28,12,12,12,12,12,10,10,9,9,
27,12,28,12,204,124,533
7140 DATA 12,12,12,12,12,10,10,9,9,27,
12,28,12,12,28,108,325
7150 DATA 204,12,12,12,10,10,9,9,27,49
,58,50,52,56,48,48,666
7160 DATA 48,48,48,80,80,144,144,216,4
8,56,48,51,62,48,48,48,1217
7170 DATA 48,48,80,80,144,144,216,48,5
6,48,48,56,54,51,48,48,1217
7180 DATA 48,80,80,144,144,216,128,4,2
,8,64,0,10,128,4,32,1092
7190 DATA 2,4,8,4,0,2,1,4,0,128,0,5,0,
2,0,64,224
7200 DATA 1,2,4,2,0,2,0,0,0,0,2,0,1,0,
128,0,142
7210 DATA 0,1,2,1,0,0,0,0,0,1,0,0,0,0,
0,0,5
7220 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0
7230 DATA 0,0,0,0,3,0,6,7,0,7,3,7,6,0,
0,159,198
7240 DATA 0,159,0,159,28,159,52,159,78
,159,78,0,78,0,78,0,1187
7250 DATA 50,0,30,0,0,40,0,40,40,40,40
,80,40,110,78,110,698
7260 DATA 64,100,0,100,40,100,40,130,4
0,255,192,48,12,3,51,75,1250
7270 DATA 112,176,51,96,160,192,51,58,
128,192,51,72,120,128,51,51,1689
7280 DATA 91,154,51,112,160,192,51,51,
136,192,51,24,50,72,50,32,1469
7290 DATA 38,38,50,64,64,72,50,32,64,5
6,50,18,69,64,50,18,797
7300 DATA 18,18,50,58,72,48,50,24,72,5
0,50,24,24,69,3,12,642
7310 DATA 48,192,1,2,4,8,83,67,79,82,6
9,58,32,48,48,48,869
7320 DATA 48,48,48,32,32,32,32,32,32,3
2,32,32,32,32,32,32,560
7330 DATA 32,32,32,32,32,32,32,32,32,3
2,32,32,32,32,77,69,594
7340 DATA 78,58,255,32,32,32,32,32,77,
65,78,73,65,67,33,32,1041
7350 DATA 32,32,32,32,32,32,32,32,32,3
2,32,32,32,32,66,89,603
7360 DATA 32,32,32,32,32,32,32,32,32,3
2,32,32,32,82,73,67,638
7370 DATA 75,32,77,69,83,83,78,69,82,2
55,80,82,69,83,83,32,1332
7380 DATA 49,32,79,82,32,50,32,32,32,3
2,32,32,32,32,32,32,644
7390 DATA 32,32,32,32,32,32,32,32,32,3
2,32,32,32,32,32,32,512
7400 DATA 32,32,49,32,45,32,79,78,69,3
2,80,76,65,89,69,82,941
7410 DATA 32,32,32,32,32,32,50,32,45,3
2,84,87,79,32,80,76,789
7420 DATA 65,89,69,82,255,83,67,79,82,
69,58,32,32,32,32,32,1158
7430 DATA 32,32,32,32,72,73,45,83,67,7
9,82,69,58,32,32,32,852
7440 DATA 32,32,32,32,32,32,32,32,32,3
2,32,32,32,32,32,32,512
7450 DATA 32,32,32,32,80,82,69,83,83,3
2,65,78,89,32,75,69,965
7460 DATA 89,32,84,79,32,67,79,78,84,7
3,78,85,69,255,0,6,1190
7470 DATA 52,101,199,167,54,162,176,52
,49,155,138,27,59,59,1,49,1500
7480 DATA 999
```

•

CHECKSUM DATA

(See pgs. 7-10)

```
10 DATA 48,253,903,981,241,241,100,417
,542,151,331,734,603,507,73,6125
160 DATA 308,656,680,489,744,369,665,2
16,207,936,610,222,898,866,848,8714
5130 DATA 424,93,577,899,23,198,526,81
0,418,894,649,567,704,616,248,7646
5280 DATA 190,256,370,280,65,154,421,1
00,880,242,90,244,229,455,274,4250
5430 DATA 238,844,343,169,242,846,524,
838,928,916,142,418,964,255,269,7936
5580 DATA 284,314,537,80,500,883,612,5
68,33,580,464,87,964,925,943,7774
5730 DATA 88,132,305,138,145,571,261,8
88,121,957,347,144,295,627,348,5367
5880 DATA 404,413,926,899,574,585,247,
168,595,123,408,344,612,567,385,7250
6030 DATA 514,524,278,170,560,441,410,
579,717,738,173,553,250,505,233,6645
6180 DATA 120,208,443,558,265,261,492,
58,314,27,201,453,572,88,6,4066
6330 DATA 244,261,291,114,575,265,312,
69,287,894,405,571,488,561,789,6126
6480 DATA 519,36,227,324,341,104,961,3
33,406,193,575,827,520,658,90,6114
6630 DATA 863,474,801,994,960,530,477,
549,241,257,779,745,233,41,519,8463
6780 DATA 563,46,358,19,870,539,714,45
1,803,935,227,950,229,356,281,7341
6930 DATA 580,949,380,997,270,654,265,
490,491,149,156,227,37,234,309,6188
7080 DATA 29,48,626,632,407,484,227,39
0,779,780,383,42,776,609,590,6802
7230 DATA 864,492,66,835,550,20,651,34
3,360,541,559,547,555,576,677,7636
7380 DATA 577,520,690,637,616,653,518,
701,711,74,900,6597
```

•

32K Disk revision.

```
10 REM MANIAC 32K DISK CHANGES
150 ? "INSERT DISK WITH DOS, PRESS RET
URN";:DIM IN$(1):INPUT IN$:OPEN #1,8,0
,"D:AUTORUN.SYS"
5000 DATA 255,255,6,64,127,79,169,60,1
41,2,211,169,119,141,231,2,2031
7480 DATA 10,0,11,0,38,64,224,2,225,2,
38,64,0,0,0,0,678
7490 DATA 999
```

•

HARVEY WALLBANGER

16K Cassette 24K Disk

by Charles Bachand

Machine language games are the wave of the future. They are the games that attract you with their graphics, their speed, and their playability. They are also the games that sell! Since it is inherently more difficult to write games in machine language, and since the author of a machine language game has a lot of his time invested in it, it is understandable that he would rather sell it than give it away. Cold cash usually speaks loudly. Personally, I prefer the fame.

So, for your enjoyment, here is Harvey Wallbanger with the full machine language source listing for the programers out there who like to modify games.

Here is a very quick description of the game: Harvey Wallbanger is situated in the middle of a closed-in room. He may move freely within the room (that part of the screen within the four walls) but the room is constantly getting smaller (the walls are moving in on our rabbit friend.) Harvey is allowed to touch the left and right hand walls, but he will be killed if flattened by these two walls. He is under no circumstances allowed to touch the top and bottom walls, for they are highly electrified.

All is not lost for Harvey, however. He does have his patented "Wallbanger Gun" to shoot at the walls and make them move away. His only problem is that the speed of the moving walls constantly increases with time. Numbers will appear on the screen that Harvey may collect. These numbers will be added to his score. The faster you collect the numbers, the bigger the final score. The numbers may not be within the confines of the now-receding wall and this will necessitate that Harvey shoot back a wall to access the numbers. Also, Harvey cannot shoot the numbers as this will kill off that particular number and generate a new one someplace else.

As you read the Assembler listing for the game you will notice that there are no line numbers in the source code. This is due to the fact that I am using the ATARI Macro Assembler package.

There are two BASIC programs before the machine language listing of Harvey Wallbanger. The first is a disk file maker program to get the game up and running faster than typing in the Assembler source listing. The second BASIC listing contains modifications to the first to make a cassette bootable machine language tape for the cassette-bound crowd.

To make a disk version of Harvey Wallbanger, you need a formatted disk with a version of DOS II on it. Load the disk maker program into the computer's RAM but do not run the program. Next, insert a formatted disk containing DOS II into the disk drive. Run the program. The computer will print out the line numbers of the DATA statements that the program is reading. These numbers start at 1000 and end at 2000 with increments of 10. After reading line 2000 the disk drive will turn on and an AUTORUN-.SYS' File will be written out to the disk. To play the game, simply remove all the cartridges and reboot with this new disk.

To make a cassette version of this program requires a slightly different procedure. After the disk version of the program is in memory, add the changes for the cassette maker program and run it. The program will read through all the data and then the computer will beep twice. At this time insert a blank cassette tape in the program recorder and set it up for record by pressing the record and play buttons on the recorder. We now hit the return key as if we were doing a CSAVE and wait. The computer is now recording the machine language program onto the cassette tape. Once it is through you can boot the program by rewinding the tape, powering down the computer, and turning on the computer while holding down the start key. The computer will beep once, press play on the recorder, and press return on the computer. The program will load from the cassette and run automatically. □

Basic listing.

```
100 REM HARVEY WALLBANGER
110 REM DISK MAKER PROGRAM
120 REM
130 DIM PROG$(1600):PNTR=1
140 LINE=990:TRAP 220
150 LINE=LINE+10:FOR COUNT=1 TO 15
160 READ BYTE:PROG$(PNTR)=CHR$(BYTE)
170 PNTR=PNTR+1:TOTAL=TOTAL+BYTE
180 NEXT COUNT:? "LINE:";LINE
190 READ CHECKSUM
200 IF CHECKSUM=TOTAL THEN 150
210 ? "BAD CHECKSUM: LINE ";LINE:STOP
220 IF PEEK(195)=6 THEN 240
230 ? "BAD DATA: LINE ";LINE:STOP
240 OPEN #1,8,0,"D:AUTORUN.SYS"
250 PRINT #1;PROG$;:END
999 REM
1000 DATA 255,255,0,52,216,57,76,71,52
,112,112,112,112,71,7,1560
1010 DATA 58,7,7,7,7,7,7,7,7,7,112,112
,70,27,52,2054
1020 DATA 65,3,52,242,225,226,226,233,
244,243,218,211,192,243,227,4904
1030 DATA 239,242,229,218,208,208,208,
208,103,97,109,101,0,0,111,7185
1040 DATA 118,101,114,128,112,114,101,
115,115,0,0,115,116,97,114,8645
1050 DATA 116,128,216,32,101,228,169,2
11,141,35,52,169,3,141,245,10632
1060 DATA 57,169,208,141,43,52,141,44,
52,141,45,52,141,46,52,12016
1070 DATA 169,60,141,228,57,141,26,2,3
2,220,56,160,2,32,231,13573
1080 DATA 56,136,16,250,169,3,141,48,2
,169,52,141,49,2,169,14976
1090 DATA 4,141,111,2,169,40,141,219,5
7,169,196,141,220,57,169,16812
1100 DATA 60,141,221,57,141,2,208,169,
184,141,222,57,141,3,208,18767
1110 DATA 169,122,141,217,57,169,55,14
1,218,57,169,46,141,47,2,20518
1120 DATA 169,3,141,29,208,169,48,141,
7,212,169,150,141,198,2,22305
1130 DATA 169,72,141,199,2,169,24,141,
192,2,169,152,141,193,2,24073
1140 DATA 169,52,141,194,2,169,196,141
,195,2,169,1,141,233,57,25935
1150 DATA 162,53,160,204,169,7,32,92,2
28,169,137,141,38,2,169,27698
1160 DATA 53,141,39,2,169,153,141,40,2
,169,53,141,41,2,169,29013
1170 DATA 1,141,25,2,169,0,141,30,208,
141,246,57,141,227,57,30599
1180 DATA 141,223,57,141,224,57,141,14
,210,162,3,157,229,57,157,32572
1190 DATA 247,57,157,251,57,157,255,57
,157,3,58,202,16,238,170,34654
1200 DATA 157,128,49,157,0,50,232,208,
247,169,255,157,0,51,232,36746
1210 DATA 208,250,232,138,41,1,170,189
,219,57,74,8,205,11,212,38761
1220 DATA 208,251,141,10,212,40,144,3,
141,10,212,173,10,210,41,40567
1230 DATA 246,141,26,208,160,10,169,0,
157,15,208,141,10,212,173,42443
1240 DATA 10,210,41,246,141,26,208,136
,208,237,173,200,2,141,26,44448
1250 DATA 208,173,245,57,240,8,173,246
,57,16,3,76,99,52,173,46274
1260 DATA 31,208,41,1,208,177,76,71,52
,173,228,57,201,2,240,48040
1270 DATA 3,206,228,57,169,1,141,25,2,
96,173,228,57,141,26,49593
1280 DATA 2,238,219,57,206,220,57,238,
221,57,173,221,57,141,2,51702
1290 DATA 208,206,222,57,173,222,57,14
1,3,208,238,227,57,173,227,54121
1300 DATA 57,41,1,170,189,207,57,141,0
,210,169,8,141,223,57,55792
1310 DATA 96,173,246,57,208,60,173,245
,57,208,55,32,220,56,162,57840
1320 DATA 0,142,1,210,142,3,210,142,5,
210,142,7,210,142,24,59430
1330 DATA 2,142,25,2,142,26,2,189,47,5
2,48,7,157,92,58,60421
1340 DATA 232,76,240,53,162,0,189,58,5
2,48,7,157,151,58,232,62136
1350 DATA 76,254,53,76,189,56,173,12,2
08,141,236,57,173,4,208,64052
1360 DATA 141,237,57,173,226,57,48,9,2
06,226,57,74,9,160,141,65873
1370 DATA 7,210,173,223,57,48,8,206,22
3,57,9,192,141,1,210,67638
1380 DATA 173,224,57,48,8,206,224,57,9
,128,141,3,210,173,225,69524
1390 DATA 57,238,225,57,238,225,57,238
,225,57,141,4,210,173,246,71915
1400 DATA 57,240,26,238,246,57,238,192
,2,238,192,2,173,192,2,74010
1410 DATA 10,10,10,141,2,210,169,136,1
41,3,210,76,189,56,173,75546
1420 DATA 229,57,240,13,206,229,57,173
,219,57,201,28,240,3,206,77704
1430 DATA 219,57,173,230,57,240,13,206
,230,57,173,220,57,201,204,80041
1440 DATA 240,3,238,220,57,173,231,57,
240,16,206,231,57,173,221,82404
1450 DATA 57,141,2,208,201,39,240,3,20
6,221,57,173,232,57,240,84481
1460 DATA 16,206,232,57,173,222,57,141
,3,208,201,208,240,3,238,86686
1470 DATA 222,57,169,0,133,77,141,234,
57,141,235,57,173,120,2,88504
1480 DATA 201,15,240,16,165,20,41,7,20
8,10,169,16,141,2,210,89965
1490 DATA 169,4,141,224,57,173,120,2,5
6,233,5,10,170,165,20,91514
1500 DATA 106,106,106,106,189,160,57,1
44,3,189,182,57,133,128,189,93369
1510 DATA 161,57,144,3,189,183,57,133,
129,162,3,78,120,2,176,94966
1520 DATA 16,189,209,57,240,3,141,234,
57,189,213,57,240,3,141,96955
1530 DATA 235,57,202,16,232,173,236,57
,201,12,208,9,206,35,52,98886
1540 DATA 206,245,57,238,246,57,41,4,2
40,8,238,217,57,169,0,100909
1550 DATA 141,234,57,173,236,57,41,8,2
40,8,206,217,57,169,0,102753
1560 DATA 141,234,57,24,173,219,57,105
,4,74,205,218,57,144,9,104474
1570 DATA 206,35,52,206,245,57,238,246
,57,173,218,57,105,10,10,106389
1580 DATA 205,220,57,144,9,206,35,52,2
06,245,57,238,246,57,24,108390
1590 DATA 173,217,57,109,234,57,141,21
7,57,141,0,208,24,173,218,110416
1600 DATA 57,109,235,57,141,218,57,170
,160,0,177,128,157,0,50,112132
1610 DATA 232,200,192,14,208,245,173,1
32,2,205,233,57,141,233,57,114456
1620 DATA 176,63,173,234,57,13,235,57,
208,5,238,233,57,208,50,116463
1630 DATA 169,64,141,225,57,169,4,141,
5,210,238,244,57,173,244,118604
1640 DATA 57,41,3,170,173,234,57,10,15
7,255,57,173,235,57,10,120293
1650 DATA 157,3,58,24,173,217,57,105,3
,157,247,57,173,218,57,121999
1660 DATA 105,8,157,251,57,169,0,170,1
57,128,49,232,16,250,162,123910
1670 DATA 3,189,255,57,29,3,58,240,127
,189,251,57,24,125,3,125520
1680 DATA 58,157,251,57,168,10,105,2,2
05,220,57,144,12,32,208,127206
1690 DATA 56,109,230,57,141,230,57,76,
31,56,233,12,205,219,57,128975
1700 DATA 176,9,32,208,56,109,229,57,1
41,229,57,185,128,49,29,130669
1710 DATA 34,57,153,128,49,185,129,49,
29,34,57,153,129,49,189,132093
1720 DATA 0,208,160,0,106,144,3,76,195
,56,200,192,4,208,245,133890
1730 DATA 24,189,247,57,125,255,57,157
,247,57,157,4,208,205,222,136101
1740 DATA 57,144,12,32,208,56,109,232,
57,141,232,57,76,111,56,137681
1750 DATA 233,6,205,221,57,176,9,32,20
8,56,109,231,57,141,231,139653
1760 DATA 57,202,48,3,76,232,55,162,3,
```

```
169,0,29,255,57,29,141030
1770 DATA 3,58,202,16,247,201,0,208,3,
141,5,210,160,0,78,142562
1780 DATA 237,57,144,40,32,10,57,185,2
38,57,72,32,231,56,104,144114
1790 DATA 168,240,23,162,3,254,43,52,1
89,43,52,201,218,208,8,145978
1800 DATA 169,208,157,43,52,202,16,238
,136,208,233,76,189,56,200,148161
1810 DATA 192,3,208,206,141,30,208,76,
98,228,138,72,32,10,57,149860
1820 DATA 32,231,56,104,170,76,111,56,
169,0,157,255,57,157,3,151494
1830 DATA 58,24,169,8,96,162,200,169,0
,157,6,58,202,208,250,153261
1840 DATA 96,174,10,210,224,200,176,24
9,189,7,58,208,244,173,10,155489
1850 DATA 210,41,15,201,10,176,247,153
,238,57,25,204,57,157,7,157287
1860 DATA 58,138,153,241,57,96,169,0,1
90,241,57,157,7,58,173,159082
1870 DATA 10,210,41,31,9,16,141,6,210,
169,30,141,226,57,96,160475
1880 DATA 3,12,48,192,0,0,18,10,60,116
,60,28,30,62,63,161177
1890 DATA 126,0,0,11,10,60,116,60,28,3
0,62,62,247,0,0,161989
1900 DATA 72,80,60,46,60,56,120,124,25
2,126,0,0,208,80,60,163333
1910 DATA 46,60,56,120,124,124,239,0,0
,66,36,60,20,60,24,164368
1920 DATA 60,126,126,231,0,0,66,36,60,
40,60,24,60,126,126,165509
1930 DATA 231,0,0,68,36,60,20,60,24,60
,126,254,7,0,0,166455
1940 DATA 34,36,60,40,60,24,60,126,127
,224,0,0,68,36,60,167410
1950 DATA 60,60,24,60,102,254,7,0,0,34
,36,60,60,60,24,168251
1960 DATA 60,102,127,224,0,0,62,57,62,
57,62,57,0,0,38,169159
1970 DATA 57,38,57,38,57,0,0,110,57,13
4,57,86,57,74,57,170038
1980 DATA 74,57,74,57,0,0,50,57,50,57,
50,57,0,0,122,170743
1990 DATA 57,146,57,98,57,16,80,144,38
,41,1,255,0,0,0,171733
2000 DATA 0,1,255,226,2,227,2,0,52,0,0
,0,0,0,0,172498
```

•

CHECKSUM DATA
(See pgs. 7-10)

```
100 DATA 817,614,80,233,884,712,13,669
,51,479,193,937,474,484,282,6922
250 DATA 285,126,878,689,503,189,136,4
69,900,631,894,11,31,76,166,5984
1130 DATA 174,210,183,831,739,238,97,7
4,168,918,186,183,768,726,894,6389
1280 DATA 179,511,757,70,780,373,752,1
99,13,939,969,269,952,906,17,7686
1430 DATA 434,201,157,52,186,712,689,4
07,32,226,188,945,940,196,259,5624
1580 DATA 66,458,25,309,36,207,201,236
,260,727,30,33,86,84,918,3676
1730 DATA 517,19,72,666,632,192,203,38
3,205,199,966,508,37,279,659,5537
1880 DATA 368,320,886,663,821,129,399,
358,305,643,246,341,428,5907
```

•

```
100 REM HARVEY WALLBANGER MOD5
110 REM CASSETTE MAKER PROGRAM
120 REM
240 OPEN #1,8,128,"C:"
241 PROG$(1,1)=CHR$(0)
242 PROG$(2,2)=CHR$(12)
243 PROG$(3,3)=CHR$(250)
244 PROG$(4,4)=CHR$(51)
```

Assembly listing.

```
;       ==========================================
;       HARVEY WALLBANGER by Charles Bachand
;       ==========================================

;       ==========================================
;       Copyright (C) 1982 ANALOG Magazine
;       ==========================================

;       ----------------------
;       Operating System Equates
;       ----------------------

HPOSP0  =       $D000   ;player 0 horizontal position
M0PF    =       $D000   ;missile 0/playfield collision
HPOSP2  =       $D002   ;player 2 horizontal position
HPOSP3  =       $D003   ;player 3 horizontal position
HPOSM0  =       $D004   ;missile 0 horizontal position
P0PF    =       $D004   ;player 0/playfield collisions
P0PL    =       $D00C   ;player 0 to player collisions
GRP2    =       $D00F   ;player 2 graphics register
COLBK   =       $D01A   ;background color
GRACTL  =       $D01D   ;graphics control register
HITCLR  =       $D01E   ;collision 'HIT' clear
CONSOL  =       $D01F   ;console switch port
AUDF1   =       $D200   ;audio frequency 1
AUDC1   =       $D201   ;audio volume 1
AUDF2   =       $D202   ;audio frequency 2
AUDC2   =       $D203   ;audio volume 2
AUDF3   =       $D204   ;audio frequency 3
AUDC3   =       $D205   ;audio volume 3
AUDF4   =       $D206   ;audio frequency 4
AUDC4   =       $D207   ;audio volume 4
RANDOM  =       $D20A   ;random number generator
IRQEN   =       $D20E   ;IRQ interrupt enable
PMBASE  =       $D407   ;P/M base address
WSYNC   =       $D40A   ;wait for horizontal sync
VCOUNT  =       $D40B   ;scan line counter
SETVBV  =       $E45C   ;set vertical blank vector
XITVBV  =       $E462   ;vertical blank exit vector
SIOINT  =       $E465   ;serial I/O initialization
ATRACT  =       $004D   ;atract mode counter

;       ----------------------
;       System Shadow Registers
;       ----------------------

RTCLOK  =       $0012   ;system clock
CDTMV1  =       $0218   ;system timer 1
CDTMV2  =       $021A   ;system timer 2
CDTMA1  =       $0226   ;system timer 1 vector
CDTMA2  =       $0228   ;system timer 2 vector
SDMCTL  =       $022F   ;DMA control
SDLSTL  =       $0230   ;display list pointer
GPRIOR  =       $026F   ;graphics priority
STICK0  =       $0278   ;joystick 1
STRIG0  =       $0284   ;trigger 1
PCOLR0  =       $02C0   ;player 0 color
PCOLR1  =       $02C1   ;player 1 color
PCOLR2  =       $02C2   ;player 2 color
PCOLR3  =       $02C3   ;player 3 color
COLOR2  =       $02C6   ;playfield 2 color
COLOR3  =       $02C7   ;playfield 3 color
COLOR4  =       $02C8   ;background color

;       ------------------
;       Page Zero Variables
;       ------------------

        ORG     $0080   ;area not used by system

PIC     DS      2       ;rabbit image pointer
```

```
;       -------------------------
;       Player / Missile RAM Space
;       -------------------------

        ORG     $3000   ;out of everyones way

PM      DS      $180    ;first area not used
MISL    DS      $80     ;Missile graphics area
PLR0    DS      $80     ;player 0 graphics area
PLR1    DS      $80     ;player 1 graphics area
PLR2    DS      $80     ;player 2 graphics area
PLR3    DS      $80     ;player 3 graphics area

;       ------------------
;       Program entry point
;       ------------------

        JMP     HARVEY

;       -----------------
;       Game display list
;       -----------------

DL      DB      $70,$70 ;32 blank scan lines
        DB      $70,$70
        DB      $47     ;mode 2 line w/LMS bit
        DW      DISP    ;address of game display
        DB      $07,$07 ;9 more mode 2 lines
        DB      $07,$07
        DB      $07,$07
        DB      $07,$07
        DB      $07
        DB      $70,$70 ;skip 16 lines
        DB      $46     ;mode 1 line w/LMS bit
        DW      SLINE   ;address of score line
        DB      $41     ;jump on vertical blank
        DW      DL      ;to start of display list

;       --------------
;       Score line data
;       --------------

SLINE   DB      'R'+$A0
        DB      'A'+$A0
        DB      'B'+$A0
        DB      'B'+$A0
        DB      'I'+$A0
        DB      'T'+$A0
        DB      'S'+$A0
        DB      ':'+$A0
RNUM    DB      '3'+$A0 ;number of rabbits
        DB      ' '+$A0
        DB      'S'+$A0
        DB      'C'+$A0
        DB      'O'+$A0
        DB      'R'+$A0
        DB      'E'+$A0
        DB      ':'+$A0
SNUM    DB      '0'+$A0 ;score display
        DB      '0'+$A0
        DB      '0'+$A0
        DB      '0'+$A0

;       ----------------
;       Game over message
;       ----------------

GOMSG   DB      'game'
        DB      0,0,'ov'
        DB      'er',$80

PSMSG   DB      'pres'
        DB      's',0,0,'s'
        DB      'tart',$80

;       ------------------
;       Initialization Code
;       ------------------

HARVEY  CLD             ;clear decimal flag
        JSR     SIOINT  ;stop cassette
        LDA     #'3'+$A0;display for '3'
        STA     RNUM    ;3 lives (display)
        LDA     #3      ;get 3 lives
        STA     LIVES   ;initialize counter
        LDA     #'0'+$A0;display for '0'
        STA     SNUM    ;store in the four
        STA     SNUM+1  ;bytes used for the
        STA     SNUM+2  ;score display
        STA     SNUM+3  ;area.
MORE    LDA     #60     ;get 1 second count
        STA     TIM2ST  ;set reset value
        STA     CDTMV2  ;set system timer #2
        JSR     CLSCRN  ;clear game playfield
        LDY     #2      ;display 3 numbers (0-2)
INUMS   JSR     PUTNUM  ;put the number on screen
        DEY             ;decrement number counter
        BPL     INUMS   ;done yet? No.
        LDA     #DL&$FF ;Yes. low byte DL address
        STA     SDLSTL  ;DL pointer (low)
        LDA     #DL/256 ;high byte DL address
        STA     SDLSTL+1;DL pointer (high)
        LDA     #$04    ;set PF over PLAYER
        STA     GPRIOR  ;graphics priority
        LDA     #40     ;high wall
        STA     BYLOC   ;starting location
        LDA     #196    ;low wall
        STA     BYLOC+1 ;starting location
        LDA     #60     ;left wall
        STA     BXLOC   ;starting location
        STA     HPOSP2  ;hardware register
        LDA     #184    ;right wall
        STA     BXLOC+1 ;starting location
        STA     HPOSP3  ;hardware register
        LDA     #122    ;center screen-4 color clocks
        STA     HARX    ;Harvey's initial X position
        LDA     #55     ;center P/M-8 bytes
        STA     HARY    ;Harvey's initial Y position
        LDA     #$2E    ;set P/M DMA on bits
        STA     SDMCTL  ;store in DMA control
        LDA     #3      ;set P/M enable bits on
        STA     GRACTL  ;store in graphics control
        LDA     #PM/256 ;get high byte of P/M addr
        STA     PMBASE  ;point hardware to it
        LDA     #$96    ;light blue color
        STA     COLOR2  ;default color too dark
        LDA     #$48    ;pink color
        STA     COLOR3  ;same here
        LDA     #$18    ;gold color
        STA     PCOLR0  ;set rabbit color
        LDA     #$98    ;blue color
        STA     PCOLR1  ;set missile 1 color
        LDA     #$34    ;red-orange color
        STA     PCOLR2  ;left wall color
        LDA     #$C4    ;green color
        STA     PCOLR3  ;right wall color
        LDA     #1      ;initialize trigger flag-
        STA     STRIGF  ;to no shot fired
        LDX     #VB/256 ;address of VB (MSB)
        LDY     #VB&$FF ;address of VB (LSB)
        LDA     #7      ;deferred vertical blank opt
        JSR     SETVBV  ;set deferred Vblank vector
        LDA     #T1&$FF ;addr of timer 1 routine LSB
        STA     CDTMA1  ;set timer 1 vector LSB
        LDA     #T1/256 ;addr of timer 1 routine MSB
        STA     CDTMA1+1;set timer 1 vector MSB
        LDA     #T2&$FF ;addr of timer 2 routine LSB
        STA     CDTMA2  ;set timer 2 vector LSB
        LDA     #T2/256 ;addr of timer 2 routine MSB
        STA     CDTMA2+1;set timer 2 vector MSB
        LDA     #1      ;get 4.25 second count
        STA     CDTMV1+1;set system timer #1
        LDA     #0      ;get a zero
        STA     HITCLR  ;reset collision registers
        STA     DIESW   ;rabbit is alive
        STA     TICTOC  ;reset tictoc counter
```

```
        STA     VOL1    ;start with no tictoc sound
        STA     VOL2    ;start with no shuffle noise
        STA     IRQEN   ;disable all IRQ interrupts
        LDX     #3      ;set index value to 3
WINCZ   STA     WINC,X  ;zero wall mover counter
        STA     SHOTX,X ;zero X missile location
        STA     SHOTY,X ;zero Y missile location
        STA     SINCX,X ;zero X missile increment
        STA     SINCY,X ;zero Y missile increment
        DEX             ;next wall mover counter
        BPL     WINCZ   ;more walls/missiles? Yes.
        TAX             ;set index to zero
IM01    STA     MISL,X  ;clear Missile area
        STA     PLR0,X  ;clear Player 0, 1 area
        INX             ;do next byte
        BNE     IM01    ;done yet? No.
        LDA     #$FF    ;turn on pixels
IM23    STA     PLR2,X  ;set Player 2, 3 area
        INX             ;do next byte
        BNE     IM23    ;done yet? No.

;       ----------------------------------------
;       Main program used to generate the display.
;       Actual game done entirely during display's
;       vertical blank processing routine.
;       ----------------------------------------

HBARS   INX             ;increment wall pointer
        TXA             ;transfer pointer to Acc
        AND     #1      ;mask off lowest bit
        TAX             ;put back in X register
        LDA     BYLOC,X ;get wall vertical position
        LSR     A       ;divide by 2, odd=carry set
        PHP             ;save carry flag
VCHECK  CMP     VCOUNT  ;compare with line counter
        BNE     VCHECK  ;not yet!
        STA     WSYNC   ;start at new line
        PLP             ;get carry flag back
        BCC     ONELIN  ;branch on even line number
        STA     WSYNC   ;wait for next line
ONELIN  LDA     RANDOM  ;random background color
        AND     #$F6    ;max lum of 6
        STA     COLBK   ;for horizontal walls
        LDY     #10     ;let's have 10 lines of this
LINES   LDA     #0      ;get a zero for overlap
        STA     GRP2,X  ;background overlaps player
        STA     WSYNC   ;wait for next line
        LDA     RANDOM  ;random background color
        AND     #$F6    ;max lum of 6
        STA     COLBK   ;for horizontal walls
        DEY             ;decrement line counter
        BNE     LINES   ;10 lines done yet? No!
        LDA     COLOR4  ;get original background
        STA     COLBK   ;store in background
        LDA     LIVES   ;more lives
        BEQ     HB1     ;No. skip code
        LDA     DIESW   ;a new life?
        BPL     HB1     ;No.
        JMP     MORE    ;Yes. more lives
HB1     LDA     CONSOL  ;check for start switch
        AND     #$01    ;mask off bit
        BNE     HBARS   ;start? No.
        JMP     HARVEY  ;restart game

;       ----------------------------------------
;       System timer #1 interrupt handler.
;       Used to speed up walls every 4.25 seconds.
;       ----------------------------------------

T1      LDA     TIM2ST  ;get wall speed
        CMP     #2      ;must stop at two
        BEQ     TIM1    ;is it two? Yes.
        DEC     TIM2ST  ;No, then decrement
TIM1    LDA     #1      ;get 4.25 second cycle time
        STA     CDTMV1+1;reset timer #1
        RTS             ;return

;       ----------------------------------------
;       System timer #2 interrupt handler.
;       Used to move walls and initiate wall noise.
;       ----------------------------------------

T2      LDA     TIM2ST  ;get timer #2 value
        STA     CDTMV2  ;reset timer #2
        INC     BYLOC   ;move top wall down
        DEC     BYLOC+1 ;move bottom wall up
        INC     BXLOC   ;change left wall location
        LDA     BXLOC   ;get new location
        STA     HPOSP2  ;change player 2 position
        DEC     BXLOC+1 ;change right wall location
        LDA     BXLOC+1 ;get new location
        STA     HPOSP3  ;change player 3 position
        INC     TICTOC  ;increment TIC-TOC counter
        LDA     TICTOC  ;get counter value
        AND     #1      ;just need 0 or 1 value
        TAX             ;use for index
        LDA     METRO,X ;get sound frequency
        STA     AUDF1   ;change frequency
        LDA     #$08    ;get volume value
        STA     VOL1    ;save in volume counter
        RTS             ;return

;       ----------------------------------------
;       Deferred vertical blank processing routine.
;       Here is where all the actual game playing
;       takes place. This could be quite long.
;       ----------------------------------------

VB      LDA     DIESW   ;rabbit dying?
        BNE     VB0     ;He sure is.
        LDA     LIVES   ;any lives left?
        BNE     VB0     ;There sure are.
        JSR     CLSCRN  ;clear screen of numbers
        LDX     #0      ;initialize X with zero
        STX     AUDC1   ;stop tictoc sound
        STX     AUDC2   ;stop dying sound
        STX     AUDC3   ;stop gun noise
        STX     AUDC4   ;stop number sound
        STX     CDTMV1  ;shut off the two timers
        STX     CDTMV1+1;ditto.
        STX     CDTMV2  ;same here.
GOPRT   LDA     GOMSG,X ;get a character
        BMI     PSINIT  ;end of scring? Yes.
        STA     DISP+85,X;put on screen
        INX             ;increment index
        JMP     GOPRT   ;continue

PSINIT  LDX     #0      ;zero the index
PSPRT   LDA     PSMSG,X ;get another character
        BMI     VBXIT   ;end of string? Yes.
        STA     DISP+144,X;put on screen
        INX             ;increment index
        JMP     PSPRT   ;continue
VBXIT   JMP     VBX     ;exit vertical blank
VB0     LDA     P0PL    ;player/player collisions
        STA     P0PLT   ;store in temp variable
        LDA     P0PF    ;player to PF collisions
        STA     P0PFT   ;store in temp variable
        LDA     NSOUND  ;treasure sound counter
        BMI     NOSND   ;end of sound? Yes.
        DEC     NSOUND  ;decrement volume
        LSR     A       ;divide volume by 2
        ORA     #$A0    ;add pure tone
        STA     AUDC4   ;change volume
NOSND   LDA     VOL1    ;get tictoc volume value
        BMI     SND2    ;if <0 we produce no sound
        DEC     VOL1    ;decrement volume value
        ORA     #$C0    ;mask on the distortion
        STA     AUDC1   ;generate the tictoc sound
SND2    LDA     VOL2    ;get shuffle volume
        BMI     SND3    ;if <0 we produce no sound
        DEC     VOL2    ;decrement volume value
        ORA     #$80    ;mask on the distortion
        STA     AUDC2   ;generate the shuffle noise
SND3    LDA     FRE03   ;get shot frequency
```

```
        INC     FREQ3   ;increment shot frequency
        INC     FREQ3   ;do it again
        INC     FREQ3   ;and one last time
        STA     AUDF3   ;change frequency (lower)
        LDA     DIESW   ;is rabbit dying
        BEQ     TMOV1   ;No. continue
        INC     DIESW   ;Yes. 2 second die period
        INC     PCOLR0  ;change rabbit colors
        INC     PCOLR0  ;again
        LDA     PCOLR0  ;get number
        ASL     A       ;*2
        ASL     A       ;*4
        ASL     A       ;*8
        STA     AUDF2   ;use as frequency
        LDA     #$88    ;get distortion
        STA     AUDC2   ;make sound
        JMP     VBX     ;exit vertical blank

TMOV1   LDA     WINC    ;check push wall up
        BEQ     TMOV2   ;push up? No.
        DEC     WINC    ;decrement push up counter
        LDA     BYLOC   ;get top wall location
        CMP     #28     ;compare with top of screen
        BEQ     TMOV2   ;at top? Yes.
        DEC     BYLOC   ;move wall up
TMOV2   LDA     WINC+1  ;check push wall down
        BEQ     TMOV3   ;push down? No.
        DEC     WINC+1  ;decrement push down counter
        LDA     BYLOC+1 ;get bottom wall location
        CMP     #204    ;compare bottom of screen
        BEQ     TMOV3   ;at bottom? Yes.
        INC     BYLOC+1 ;move wall down
TMOV3   LDA     WINC+2  ;check push wall left
        BEQ     TMOV4   ;push left? No.
        DEC     WINC+2  ;decrement push left counter
        LDA     BXLOC   ;get left wall position
        STA     HPOSP2  ;move left wall player
        CMP     #39     ;check for left wall limit
        BEQ     TMOV4   ;at limit? Yes.
        DEC     BXLOC   ;move wall left
TMOV4   LDA     WINC+3  ;check push wall right
        BEQ     TMOVX   ;push right? No.
        DEC     WINC+3  ;decrement push right counter
        LDA     BXLOC+1 ;get right wall position
        STA     HPOSP3  ;move right wall player
        CMP     #208    ;check for right wall limit
        BEQ     TMOVX   ;at limit? Yes.
        INC     BXLOC+1 ;move wall right
TMOVX   LDA     #0      ;get a zero
        STA     ATRACT  ;poke out atract mode
        STA     XTEMP   ;zero rabbit X increment
        STA     YTEMP   ;zero rabbit Y increment
        LDA     STICK0  ;get joystick value
        CMP     #$0F    ;at center position?
        BEQ     CENTER  ;Yes. skip code
        LDA     RTCLOK+2;get real time clock LSB
        AND     #$07    ;at 1/7.5 second mark?
        BNE     CENTER  ;No. skip code
        LDA     #$10    ;get shuffle frequency
        STA     AUDF2   ;set frequency register
        LDA     #$04    ;get volume value
        STA     VOL2    ;set shuffle volume
CENTER  LDA     STICK0  ;get joystick value
        SEC             ;set carry for subtract
        SBC     #5      ;values 5-15 only
        ASL     A       ;5-15 now 0,2,4,...
        TAX             ;use for index
        LDA     RTCLOK+2;get real time clock LSB
        ROR     A       ;divide by 2
        ROR     A       ;divide by 4
        ROR     A       ;divide by 8
        ROR     A       ;carry set/reset at .13 sec
        LDA     PK1,X   ;get rabbit picture LSB
        BCC     PICMVL  ;other pic at .13 sec? No.
        LDA     PK2,X   ;get alternate picture LSB
PICMVL  STA     PIC     ;store LSB of pic address
        LDA     PK1+1,X ;get rabbit picture MSB
        BCC     PICMVH  ;other pic at .13 sec? No.
        LDA     PK2+1,X ;get alternate picture MSB
PICMVH  STA     PIC+1   ;store MSB of pic address
        LDX     #3      ;count 3 downto 0
CHKSTK  LSR     STICK0  ;shift bit into carry
        BCS     CHKNXT  ;correct direction? No.
        LDA     STBLX,X ;check X movement direction
        BEQ     CHK0    ;movement allowed? No.
        STA     XTEMP   ;store X movement value
CHK0    LDA     STBLY,X ;check Y movement direction
        BEQ     CHKNXT  ;movement allowed? No.
        STA     YTEMP   ;store Y movement value
CHKNXT  DEX             ;do next stick position
        BPL     CHKSTK  ;done yet? No.
        LDA     P0PLT   ;get player 0 collision
        CMP     #$0C    ;left/right squeze?
        BNE     NOSQUE  ;No. Check indvdual walls
        DEC     RNUM    ;decrement lives display
        DEC     LIVES   ;decrement lines counter
        INC     DIESW   ;the rabbit has died switch
NOSQUE  AND     #$04    ;check left wall collision
        BEQ     BMPRT   ;hit left wall? No.
        INC     HARX    ;Yes. Move rabbit to right
        LDA     #0      ;get zero value
        STA     XTEMP   ;stop rabbit X movement
BMPRT   LDA     P0PLT   ;get player 0 collision
        AND     #$08    ;check right wall collision
        BEQ     BMPUP   ;hit right wall? No.
        DEC     HARX    ;Yes. Move rabbit to left
        LDA     #0      ;get zero value
        STA     XTEMP   ;stop rabbit X movement
BMPUP   CLC             ;clear carry for add
        LDA     BYLOC   ;top wall Y location
        ADC     #4      ;offset by 4
        LSR     A       ;divide by 2
        CMP     HARY    ;compare rabbit Y location
        BCC     BMPDN   ;hit top wall? No.
        DEC     RNUM    ;decrement lives display
        DEC     LIVES   ;decrement lines counter
        INC     DIESW   ;the rabbit has died switch
BMPDN   LDA     HARY    ;get rabbit Y location
        ADC     #10     ;offset by 10
        ASL     A       ;multiply by 2
        CMP     BYLOC+1 ;compare bottom wall Y
        BCC     NOBMP   ;hit bottom wall? No.
        DEC     RNUM    ;decrement lives display
        DEC     LIVES   ;decrement lines counter
        INC     DIESW   ;the rabbit has died switch
NOBMP   CLC             ;clear carry for add
        LDA     HARX    ;get rabbit X position
        ADC     XTEMP   ;add X increment
        STA     HARX    ;save new rabbit X position
        STA     HPOSP0  ;position rabbit player 0
        CLC             ;clear carry for add
        LDA     HARY    ;get rabbit Y position
        ADC     YTEMP   ;add Y increment
        STA     HARY    ;save new rabbit Y position
        TAX             ;use position as index
        LDY     #0      ;initialize picture counter
MOVHAR  LDA     (PIC),Y ;get rabbit picture byte
        STA     PLR0,X  ;store in player 0 area
        INX             ;increment player pointer
        INY             ;increment picture pointer
        CPY     #14     ;check for end of picture
        BNE     MOVHAR  ;at end? No.
        LDA     STRIG0  ;get trigger value
        CMP     STRIGF  ;compare with trigger flag
        STA     STRIGF  ;save new trigger flag
        BCS     NOFIRE  ;shot fired? No.
        LDA     XTEMP   ;rabbit X increment
        ORA     YTEMP   ;OR rabbit Y increment
        BNE     FIREGN  ;rabbit stationary? No.
        INC     STRIGF  ;set trigger flag to 1
        BNE     NOFIRE  ;skip fire routine
FIREGN  LDA     #$40    ;initialize frequency
        STA     FREQ3   ;zero audio freq 3
        LDA     #$04    ;shot volume + distortion
```

```
        STA     AUDC3   ;enable volume 3
        INC     SHOTS   ;increment shot pointer
        LDA     SHOTS   ;get shot pointer
        AND     #3      ;make it 0-3 only
        TAX             ;use pointer for index
        LDA     XTEMP   ;get rabbit X increment
        ASL     A       ;make shot twice as fast
        STA     SINCX,X ;set missile X increment
        LDA     YTEMP   ;get rabbit Y increment
        ASL     A       ;make shot twice as fast
        STA     SINCY,X ;set missile Y increment
        CLC             ;clear carry for add
        LDA     HARX    ;get rabbit X position
        ADC     #3      ;move to center X of rabbit
        STA     SHOTX,X ;shot initial X position
        LDA     HARY    ;get rabbit Y position
        ADC     #8      ;move to center Y of rabbit
        STA     SHOTY,X ;shot initial Y position
NOFIRE  LDA     #0      ;zero accumulator
        TAX             ;zero X index
ERASES  STA     MISL,X  ;zero all missiles
        INX             ;next missile byte
        BPL     ERASES  ;done? No.
        LDX     #3      ;count 3 downto 0
PLOTS   LDA     SINCX,X ;get missile X increment
        ORA     SINCY,X ;OR missile Y increment
        BEQ     NOPLOT  ;any movement? No.
        LDA     SHOTY,X ;missile Y position
        CLC             ;clear carry for add
        ADC     SINCY,X ;add Y increment
        STA     SHOTY,X ;store new Y position
        TAY             ;Y position now index
        ASL     A       ;multiply by 2
        ADC     #2      ;offset for compare
        CMP     BYLOC+1 ;compare with bottom wall
        BCC     HITTP   ;hit bottom wall? No.
        JSR     ZINCXY  ;zero missile increments
        ADC     WINC+1  ;add 8 to wall increment
        STA     WINC+1  ;new bottom wall increment
        JMP     PLOTNH  ;continue

HITTP   SBC     #12     ;offset for bottom side
        CMP     BYLOC   ;compare with top wall
        BCS     PLOTNH  ;hit top wall? No.
        JSR     ZINCXY  ;zero missile increments
        ADC     WINC    ;add 8 to wall increment
        STA     WINC    ;new top wall increment
PLOTNH  LDA     MISL,Y  ;get missile byte
        ORA     MISMSK,X;OR missile mask
        STA     MISL,Y  ;store new byte
        LDA     MISL+1,Y;get next missile byte
        ORA     MISMSK,X;OR missile mask
        STA     MISL+1,Y;store new next byte
        LDA     M0PF,X  ;missile/playfield collision
        LDY     #0      ;init Y register
MHPF    ROR     A       ;collision?
        BCC     MHPF0   ;No. No. No.
        JMP     MHIT    ;Yes. Yes. Yes.
MHPF0   INY             ;try next bit
        CPY     #4      ;any more bits?
        BNE     MHPF    ;Certainly! Yuk. Yuk.
        CLC             ;clear carry for add
        LDA     SHOTX,X ;get missile X position
        ADC     SINCX,X ;add X increment
        STA     SHOTX,X ;store new X position
        STA     HPOSM0,X;position missile
        CMP     BXLOC+1 ;compare missile with wall
        BCC     HITLF   ;hit right wall? No.
        JSR     ZINCXY  ;zero missile increments
        ADC     WINC+3  ;add 8 to wall increment
        STA     WINC+3  ;new wall increment
        JMP     NOPLOT  ;continue

HITLF   SBC     #6      ;offset for right side
        CMP     BXLOC   ;compare with left wall
        BCS     NOPLOT  ;hit left wall? No.
        JSR     ZINCXY  ;zero missile increments
        ADC     WINC+2  ;add 8 to wall increment
        STA     WINC+2  ;new wall increment
NOPLOT  DEX             ;next missile
        BMI     NOPL1   ;missiles done? Yes.
        JMP     PLOTS   ;continue loop

NOPL1   LDX     #3      ;set up pointer
        LDA     #0      ;zero accumulator
CHKMIS  ORA     SINCX,X ;OR in X increments
        ORA     SINCY,X ;OR in Y increments
        DEX             ;decrement pointer
        BPL     CHKMIS  ;at end? No.
        CMP     #0      ;check shot increments
        BNE     NOSSND  ;any increments? Yes.
        STA     AUDC3   ;end shot sound
NOSSND  LDY     #0      ;initialize Y index
MISHIT  LSR     P0PFT   ;shift collision to carry
        BCC     MH1     ;collision w/number? No.
        JSR     ERANUM  ;erase the number
        LDA     VTBL,Y  ;get value of number
        PHA             ;save on stack
        JSR     PUTNUM  ;put out a new number
        PLA             ;get old number
        TAY             ;use as counter value
        BEQ     SCX     ;was it zero? Yes.
SCORER  LDX     #3      ;point to score low digit
SC1     INC     SNUM,X  ;increment digit
        LDA     SNUM,X  ;get digit
        CMP     #'9'+$A1;past ATASCII '9'+color?
        BNE     SCY     ;No. continue
        LDA     #'0'+$A0;reset digit
        STA     SNUM,X  ;change score display
        DEX             ;point to next digit
        BPL     SC1     ;score rolled over? No.
SCY     DEY             ;decrement value
        BNE     SCORER  ;scoring done? No.
SCX     JMP     VBX     ;exit routine

MH1     INY             ;check next color digit
        CPY     #3      ;done 0-2 yet?
        BNE     MISHIT  ;No. continue
VBX     STA     HITCLR  ;clear collision registers
        JMP     XITVBV  ;exit deferred vertical blank

MHIT    TXA             ;save X register
        PHA             ;on stack
        JSR     ERANUM  ;erase number hit and
        JSR     PUTNUM  ;put a new one on screen
        PLA             ;pull X register
        TAX             ;from stack
        JMP     NOPLOT  ;continue on

;       ------------------------
;       Commonly used subroutines
;       ------------------------

;       Clear missile display area

ZINCXY  LDA     #0      ;get zero value
        STA     SINCX,X ;zero missile X increment
        STA     SINCY,X ;zero missile Y increment
        CLC             ;clear carry for add
        LDA     #8      ;get value for add
        RTS             ;we return to the program

;       Clear the game playfield

CLSCRN  LDX     #200    ;set 0-199 bytes
        LDA     #0      ;to zero
CL0     STA     DISP-1,X;store in display
        DEX             ;count down
        BNE     CL0     ;past zero yet? No.
        RTS             ;return to program

;       Put random number from 0-9 on screen at
;       a random location 0-199

PUTNUM  LDX     RANDOM  ;get random number
```

```
        CPX     #200    ;is number < 200?
        BCS     PUTNUM  ;No. try another
        LDA     DISP,X  ;see if space is occupied
        BNE     PUTNUM  ;Yes. try again
PN0     LDA     RANDOM  ;get another random number
        AND     #$0F    ;limit it to 0-15
        CMP     #10     ;is number < 10?
        BCS     PN0     ;No. try another
        STA     VTBL,Y  ;save number
        ORA     CTBL,Y  ;OR with color
        STA     DISP,X  ;put number on screen
        TXA             ;move screen offset to A
        STA     ATBL,Y  ;save screen offset
        RTS             ;end of routine

;       Erase number from screen

ERANUM  LDA     #0      ;get zero for blank
        LDX     ATBL,Y  ;get # position on screen
        STA     DISP,X  ;blank number on screen
        LDA     RANDOM  ;get random number
        AND     #$1F    ;mask off high bits
        ORA     #$10    ;make it $10-$1F
        STA     AUDF4   ;use as sound frequency
        LDA     #30     ;initialize-
        STA     NSOUND  ;volume counter
        RTS             ;end of routine

;       ---------------------------
;       Program tables and constants
;       ---------------------------

MISMSK  DB      $03     ;missile 0 mask
        DB      $0C     ;missile 1 mask
        DB      $30     ;missile 2 mask
        DB      $C0     ;missile 3 mask

HARLF1  DB      0,0     ;left view #1
        DB      $12,$0A
        DB      $3C,$74
        DB      $3C,$1C
        DB      $1E,$3E
        DB      $3F,$7E
HARLF2  DB      0,0     ;left view #2
        DB      $0B,$0A
        DB      $3C,$74
        DB      $3C,$1C
        DB      $1E,$3E
        DB      $3E,$F7
HARRT1  DB      0,0     ;right view #1
        DB      $48,$50
        DB      $3C,$2E
        DB      $3C,$38
        DB      $78,$7C
        DB      $FC,$7E
HARRT2  DB      0,0     ;right view #2
        DB      $D0,$50
        DB      $3C,$2E
        DB      $3C,$38
        DB      $78,$7C
        DB      $7C,$EF
HARFR1  DB      0,0     ;front view #1
        DB      $42,$24
        DB      $3C,$14
        DB      $3C,$18
        DB      $3C,$7E
        DB      $7E,$E7
HARFR2  DB      0,0     ;front view #2
        DB      $42,$24
        DB      $3C,$28
        DB      $3C,$18
        DB      $3C,$7E
        DB      $7E,$E7
HARDN1  DB      0,0     ;down view #1
        DB      $44,$24
        DB      $3C,$14
        DB      $3C,$18
        DB      $3C,$7E
        DB      $FE,$07
HARDN2  DB      0,0     ;down view #2
        DB      $22,$24
        DB      $3C,$28
        DB      $3C,$18
        DB      $3C,$7E
        DB      $7F,$E0
HARUP1  DB      0,0     ;up view #1
        DB      $44,$24
        DB      $3C,$3C
        DB      $3C,$18
        DB      $3C,$66
        DB      $FE,$07
HARUP2  DB      0,0     ;up view #2
        DB      $22,$24
        DB      $3C,$3C
        DB      $3C,$18
        DB      $3C,$66
        DB      $7F,$E0
        DB      0,0

PK1     DW      HARRT1  ;rabbit pictures set 1
        DW      HARRT1
        DW      HARRT1
        DW      0
        DW      HARLF1
        DW      HARLF1
        DW      HARLF1
        DW      0
        DW      HARDN1
        DW      HARUP1
        DW      HARFR1

PK2     DW      HARRT2  ;rabbit pictures set 2
        DW      HARRT2
        DW      HARRT2
        DW      0
        DW      HARLF2
        DW      HARLF2
        DW      HARLF2
        DW      0
        DW      HARDN2
        DW      HARUP2
        DW      HARFR2

CTBL    DB      $10,$50 ;color offset table
        DB      $90
METRO   DB      38,41   ;tictoc tones
STBLX   DB      $01,$FF ;joystick X increments
        DB      $00,$00
STBLY   DB      $00,$00 ;joystick Y increments
        DB      $01,$FF

;       --------------------
;       Variable Storage Area
;       --------------------

HARX    DS      1       ;Harvey's X locatin
HARY    DS      1       ;Harvey's Y location
BYLOC   DS      2       ;horizontal wall Y locations
BXLOC   DS      2       ;vertical wall X locations
VOL1    DS      1       ;tictoc volume
VOL2    DS      1       ;shuffle volume
FREQ3   DS      1       ;shot frequency
NSOUND  DS      1       ;pick number up sound
TICTOC  DS      1       ;tictoc sound counter
TIM2ST  DS      1       ;wall speed timer
WINC    DS      4       ;wall mover counters
STRIGF  DS      1       ;trigger compare register
XTEMP   DS      1       ;temporary variable
YTEMP   DS      1       ;temporary variable
P0PLT   DS      1       ;player 0 collision shadow
P0PF    DS      1       ;PL to PF collision shadow
VTBL    DS      3       ;value of #'s on screen
ATBL    DS      3       ;screen offset to #'s
SHOTS   DS      1       ;shot enable counter
```

```
LIVES   DS      1       ;number of lives left
DIESW   DS      1       ;rabbit dying switch
SHOTX   DS      4       ;missile X location
SHOTY   DS      4       ;missile Y location
SINCX   DS      4       ;missile X increment
SINCY   DS      4       ;missile Y increment
DISP    DS      200     ;screen display area

        END     HARVEY
```

•

Cube Demo

```
5 REM *** CUBE 'FILL' GRAPHICS DEMO **
*
10 GRAPHICS 7+16:SETCOLOR 0,0,12:SETCO
LOR 1,3,2:SETCOLOR 2,7,4:HUE=1
20 FOR CUBE=1 TO 15:RAND=RND(0):MAX=15
+15*RAND:MIN=5+5*RAND:PX=2+RND(0)*116:
PY=2+RND(0)*52:REM *** 15 CUBES ***
30 X1=PX+MIN:X2=PX+MAX:X3=X2+MIN:Y1=PY
+MIN:Y2=PY+MAX:Y3=Y2+MIN:REM *** CUBE
COORDS ***
35 COLOR 0:PLOT X3+1,Y3+1:DRAWTO X3+1,
Y1:DRAWTO X2,PY-1:DRAWTO PX-1,PY-1
36 DRAWTO PX-1,Y2:DRAWTO X1,Y3+1:DRAWT
O X3+1,Y3+1
40 FOR N=1 TO MIN:PLOT PX+N,PY+N:DRAWT
O X2+N,PY+N:PLOT PX+N,PY+N:DRAWTO PX+N
,Y2+N:NEXT N
50 FOR N=1 TO MAX+1:PLOT X1,Y1+N:DRAWT
O X3,Y1+N:NEXT N:REM *** 35-50 ERASE C
UBE AREA ***
55 REM *** NOW DRAW & FILL CUBE SIDES
***
60 COLOR HUE:PLOT X3,Y3:DRAWTO X3,Y1:D
RAWTO X1,Y1:POSITION X1,Y3:POKE 765,HU
E:XIO 18,#6,0,0,"S:":GOSUB 200
70 COLOR HUE:PLOT X3,Y1:DRAWTO X2,PY:D
RAWTO PX,PY:POSITION X1,Y1:POKE 765,HU
E:XIO 18,#6,0,0,"S:":GOSUB 200
80 COLOR HUE:PLOT X1,Y3:DRAWTO X1,Y1:D
RAWTO PX,PY:POSITION PX,Y2:POKE 765,HU
E:XIO 18,#6,0,0,"S:"
90 PLOT X1,Y3:DRAWTO X1,Y2:DRAWTO PX,Y
2:POSITION X1,Y3:XIO 18,#6,0,0,"S:":NE
XT CUBE
100 REM *** ROTATE COLORS A WHILE ***
110 FOR ROT=1 TO 500:T=PEEK(708):POKE
708,PEEK(709):POKE 709,PEEK(710):POKE
710,T
120 FOR DELAY=1 TO 20:NEXT DELAY:NEXT
ROT:RUN :REM *** DO IT AGAIN! ***
200 HUE=HUE+1:IF HUE=4 THEN HUE=1
210 RETURN
```

CHECKSUM DATA
(See pgs. 7-10)

```
5 DATA 991,794,593,376,682,878,339,414
,60,955,100,752,929,902,845,9610
120 DATA 3,319,586,908
```

FILL 'ER UP II

16K Cassette 24K Disk

by Tom Hudson

If you've ever typed in a game program from a computer magazine hoping for an arcade-quality masterpiece, you've probably been disappointed. Games written in BASIC are usually too slow to handle the complex graphics and game logic necessary for an entertaining arcade-style game. In an effort to satisfy those avid video-gamers out there, I have written **Fill 'Er Up**!, a public-domain assembly-language game.

Typing the program.

Before tackling the program listings accompanying this article, let's look at them and see what they do.

Listing 1 is the main data and data checking routine. This listing is used to create both tape and disk versions of **Fill 'Er Up**. The data that makes up the **Fill 'Er Up** program listed in hexadecimal (base 16). The program is listed this way so that it will run with 16K cassette systems. I realize that those DATA statements aren't fun to type in, but they are a necessary evil.

Listing 2 should be added to **Listing 1** if you are using an ATARI cassette recorder.

Listing 3 should be added to **Listing 1** if you are using a disk drive.

Listing 4 is the assembly-language source code for **Fill 'Er Up**, created with the ATARI **Macro Assembler** editor. You DO NOT have to type in this listing to play the game! It is provided so that readers interested in assembly language can see how the program works.

Follow the instructions to make either a cassette or disk version of **Fill 'Er Up**.

Cassette instructions.

1. Type **Listing 1** into your computer. Use the C:CHECK program to check the accuracy of your typing.
2. With **Listing 1** in your computer, type in **Listing 2.** This operation will merge the two listings. Make sure the lines were entered correctly, then CSAVE the new program.
3. Type RUN and press RETURN. The program will begin printing the line numbers of the DATA statements as it reads and checks each one. It will alert you if it finds any problems in the DATA. Correct any problems in the data lines and re-RUN the program until all the DATA is checked and correct.
4. When all the data lines are correct. the program will ask you to "READY CASSETTE AND PRESS RETURN." Place a blank tape in your recorder, press RECORD and PLAY simultaneously and press ready. When finished, the BASIC "READY" prompt will appear. If you have not CSAVEd the BASIC program, do so at this point. You may not need this program again, but it's good to have if you ever need another copy of the game.
5. To play **Fill 'Er Up**, rewind the tape created by the BASIC program to the beginning. Turn your computer OFF and remove any cartridges. Press computer OFF and remove any cartridges. Press PLAY on the recorder, then turn your computer ON while pressing the START key. The computer will BEEP once. Press RETURN, and Fill 'Er Up will load and run automatically.

Disk instructions.

1. Type **Listing 1** into your computer. Use D:CHECK to verify your typing.
2. After **Listing 1** is correctly typed into your computer, type in **Listing 3**. The lines will automatically merge with **Listing 1**. It's a good idea to SAVE the whole BASIC program at this time.

3. Type RUN and press RETURN. The program will begin verifying the DATA lines, printing the line numbers as it checks each one. It will alert you if any errors are located in the data. Fix any incorrect lines and re-RUN the program until all errors are eliminated.
4. When all the DATA lines are checked, the program will tell you to "INSERT DISK WITH DOS, PRESS RETURN." Place a disk with DOS in drive 1 and press RETURN. The program will write an AUTORUN.SYS file to your disk. This file contains the Fill 'Er Up game. When finished, the BASIC "READY" prompt will appear. Make sure the BASIC program has been SAVED before continuing.
5. To play **Fill 'Er Up**, place the disk containing the AUTORUN.SYS file in drive 1. Turn the computer OFF, remove any cartridges and turn the computer back ON. **Fill 'Er Up** will load and run automatically.

Game description.

You have been assigned to build a series of water reservoirs in uncharted territory. Unfortunately, an electrified starfish (don't boggle; read on) is patrolling the area. Using your joystick, you must manuever yourself around on the screen, building walls to hold the water, while avoiding the starfish.

You start out on the white border surrounding the planned resevoir area. You may move around on these white walls by moving your joystick in the desired direction. You can build a reservoir wall by moving into the black "uncharted" area while pressing your joystick button. The walls you make can be any length, and must be terminated at a white wall. When you finish a wall by hitting a white wall, the area you have enclosed will fill with water. Do NOT run into the wall you are building or you will be destroyed. If the starfish hits you or any part of the wall you are building before you complete it, you will be destroyed. On levels 1,2,4,7,12 and 13 you will be safe from attack when standing on a white wall, but on other levels the starfish can destroy you on contact at any time!

If you do not complete the levels in a certain time period, electrified sea urchins will begin appearing on the white walls, moving along it looking for YOU! These creatures prove fatal on contact, but they can be destroyed by trappping them inside a completed resevoir. The sea urchins have no sense of fair play, and will "gang up" against you whenever possible.

At the bottom of the screen are several information displays. "TGT" indicates the TARGET area you must fill with water before you complete the level. "CUR" indicates the CURRENT area you have filled. Once CUR reaches TGT, you have completed the level and are awarded points. SCORE indicates the number of points you have gained. At the end of each level, the computer will give you 2 points for each unit over the target you have filled. If the TARGET amount is 8000 and you fill 9000 units, you receive 2000 points. "Fill 'Er Up" may be paused at any time by pressing the space bar.

This game contains 16 levels of difficulty. The level number is shown in the lower left corner of the screen.

You have three lives, shown in the lower right corner of the screen. Good luck! □

```
1 REM *** FILL 'ER UP! II ***
10 DATA 0,1,2,3,4,5,6,7,8,9,0,0,0,0,0,
0,0,10,11,12,13,14,15
20 DIM DAT$(91),HEX(22):FOR X=0 TO 22:
READ N:HEX(X)=N:NEXT X:LINE=990:RESTOR
E 1000:TRAP 60:? "CHECKING DATA"
25 LINE=LINE+10:? "LINE:";LINE:READ DA
T$:IF LEN(DAT$)<>90 THEN 110
28 DATLIN=PEEK(183)+PEEK(184)*256:IF D
ATLIN<>LINE THEN ? "LINE ";LINE;" MISS
ING!":END
30 FOR X=1 TO 89 STEP 2:D1=ASC(DAT$(X,
X))-48:D2=ASC(DAT$(X+1,X+1))-48:BYTE=H
EX(D1)*16+HEX(D2)
35 IF PASS=2 THEN PUT #1,BYTE:NEXT X:R
EAD CHKSUM:GOTO 25
40 TOTAL=TOTAL+BYTE:NEXT X:READ CHKSUM
:IF TOTAL=CHKSUM THEN 25
50 GOTO 110
60 IF PEEK(195)<>6 THEN 110
100 ? "WRITING FILE":PASS=2:LINE=990:R
ESTORE 1000:TRAP 60:GOTO 25
110 ? "BAD DATA: LINE ";LINE:END
1000 DATA A9258581A9148592A90085808591
A000B1809191E680E691D004E681E692A592C9
23D0ECA591C909D0E64C2D14,6169
1010 DATA A9008D2F028D1DD08DC802A9748D
C402A9C48DC502A90A8DC602A9348DC702A9F7
8D3002A9208D3102A92E8D2F,11064
1020 DATA 02A9038D1DD04C28192065E4A900
A27F9580CA10FB8D14228D15228D16228D1722
A9118D6F02A9018DAC228DC4,15402
1030 DATA 22202F15A9408D6722A9808D6822
A9D0A2049D19219D2421CA10F7A2059D3521CA
10FAA9008DC42285898DC622,20133
1040 DATA 8D1ED08D2F028D0ED48D28228580
8D08D2A2059D1D22CA10FA8D2622A9038D2C22
09908D3C21A90A8DC402A924,24483
1050 DATA 8DC502A9948DC602A9C48DC702A9
008DC802A9768DC102A93F8DC2028DC302A934
8DC002A9928D3002A9208D31,29683
1060 DATA 02A00AA21FA906205CE4A9108D07
D4A92E8D2F02A9038D1DD0A9408D0ED44C9615
A900A27F9D80119D00129D80,34142
1070 DATA 129D00139D8013CAD0EE60A5830A
8580A900858106802681068A5808584268105
818585068026810680268105,38869
1080 DATA 8018658485805816585858581A900
18658085805A9306581858105822903AAA5824A
4A18658085805A5816900858,43824
1090 DATA 60A2008682A2008683204615A683
A900A02791808810FBE8E056D0EBA9038DA622
AEA622BD92228582BD962285,49247
1100 DATA 83BD9A228DA722BD9E228DA822BD
A2228DA922204615BD7E22A000118091808A582
186DA7228582A583186DA822,54123
1110 DATA 8583CEA922D0DFCEA62210BBA950
858BA954858CAD26221869018D2922A9008D14
228D15228D16228D17228D24,58606
1120 DATA 228D25228D2A22A9FF8D23222049
1AADFA2209908D2C21ADF92209908D2D21AE26
22BDA1218D2922BDB1218D2A,62917
1130 DATA 22BDC1218D7D22BDE12185958594
A9048D232220491AA9008DAC22AA9D09239D09
24CAD0F78DD0228DCF22AD27,67913
1140 DATA 22D0FBA9FD8D00D2A9FE8D02D2A9
FF8D04D2A9A38D01D28D03D28D05D2A9008540
A58AD023A589F022AE2622BD,74064
1150 DATA D121D017A58B8582A58C85832046
15A000BD8A223180DD7E22F0034CD118A587F0
064C9C1A4C7416A9048587AD,79059
```

```
1160 DATA 78028DC822AABDD122187DD12285
8DBDE122187DE122858E058DF0DCA58B18658D
8DAD22C99FB0D0858238FDD1,85004
1170 DATA 228DAA22A58C18658E8DAE22C955
B0BB858338FDE1228DAB22204615A000BD8A22
31808DCD228ECE22ADAA2285,90224
1180 DATA 82ADAB22858320461 5A000BD8A22
318048AD8402D00668D01E4C641768DD7E22D0
15ADCD22AECE22DD7E22D00A,95090
1190 DATA ADAD22858BADAE22858C4C7416AD
CF22D0278DD022ADC8228D0923A9018DCF228D
2822A58B8DAF228DB1228DB3,100339
1200 DATA 22A58C8DB0228DB2228DB422ADCD
22AECE22DD8222D0034CD118AED022ADC822DD
0923F00CEED022E89D0923A9,105890
1210 DATA 009D0924FE0924A9038DA922A58B
8582A58C8583204615A000B1803D8E221D8222
9180CEA922F019ACD022BE09,110621
1220 DATA 23BDD1221865828582BDE1221865
8385834CC517A582858BCDB3229 0068DB3224C
0918CDB122B0038DB122A583,115711
1230 DATA 858CCDB42290068DB4224C2018CD
B222B0038DB222AECE22ADCD22DD7E22F0034C
7416A9008DCF2220F81BAD24,120720
1240 DATA 228D2922AD25228D2A22A90F8D23
2220491AA9018DC92220B219AE2622AD242238
FDA1218D2922AD2522FDB121,124652
1250 DATA 8D2A2210034C5F16AD2622C90FF0
03EE26220E29222E2A22A9FF8D232220491AA2
05A000B9F922187D1D22C90A,128289
1260 DATA 300C38E90A9D1D22FE1C224CA318
9D1D22C8CA10E2A205BD1D2209D09D3521CA10
F5A9018DC4228DAC22202F15,132641
1270 DATA A9408D6722A9808D6822A9008DC4
224C9615A9008D01D28D03D28D05D2A9018DC6
22A90F8DC722A9058588ADC7,137662
1280 DATA 228D01D2AD0AD2291F8D00D2AD0A
D229F00DC7228DC5028DC102A588D0E0CEC722
10D7CE2C22AD2C2209908D3C,142752
1290 DATA 21C990D025A93D8DEF20A9218DF0
20AD1FD02901D0F9AD1FD02901F0F9A9158DEF
20A9218DF0204C6314A9018D,148209
1300 DATA AC22AD0AD229FEC99FB0F78582AD
0AD2297EC955B0F7858320461 5A000BD8A2231
80DD7E22D0DB202F15A58285,153839
1310 DATA 8BA583858CA9008DC922AD2822F0
1C20B219ADAF228582ADB0228583204615A000
BD8E2231801D7E229180A924,158626
1320 DATA 8DC502A9008DC6228D1ED0858985
8A4C5F16ADAF228DB522ADB0228DB622A9008D
B722AEB722BD09238DCA22BD,163816
1330 DATA 09248DCB22A9018DB822A9038DCC
22ADB5228582ADB6228583204615A000ADC922
D010BD8E2231809180 4C0A1A,168465
1340 DATA A9008DCF2260BD8E2231801D7E22
9180CECC22F01AAECA22ADB522187DD1228DB5
22ADB622187DE1228DB6224C,173602
1350 DATA DC19EEB822ADB822CDCB22F002B0
034CD719EEB722ADB722CDD022F002B0B44CC3
19A204A9009DF922CA10FAA2,179464
1360 DATA 04AD2A22DDFF22F004B0109008AD
2922DD0423B006CA10E94C881AAD292238FD04
238D2922AD2A22FDFF228D2A,183980
1370 DATA 22FEF9224C551AA204AC2322300C
BDF92209D0991521C8CA10F460A586F0034C9F
1BAD7D228586AD672238E90D,188874
1380 DATA 8D6A22AD682238E92C8D6922AD0A
D2C9F09008AD0AD229074CCC1AAD6C22AA8D6B
22AD6922187D6D228582AD6A,193725
1390 DATA 22187D7522858320461 5A000BD8A
223180F007A90F8DC122D0CCA58218692C8D68
22A58318690D8D6722AD6B22,198132
1400 DATA 8D6C22A9038596A696BD1422F020
BD0C228582BD1022858320461 5A000BD8A2231
80DD7E22F00EA696A9009D14,202820
1410 DATA 22C69618D54C9F1BA9038DC322A6
96BD18228597A696A497BD0C2218790422C99F
B01E8582BD1022187908 22C9,207587
1420 DATA 55B0118583204615A000BD8A2231
80DD7E22F015CEC32230B4AEC322A597187D01
22290385974C491BA696A582,212273
1430 DATA 9D0C22A5839D1022A5979D18224C
361BA594F0034C7416A5958594A203BD1422F0
06CA10F84C74168698AD0AD2,217045
1440 DATA C9A0B0F98582AD0AD2297FC955B0
F7858320461 5A000BD8A223180DD7E22D0DDA6
98A5829D0C22A5839D1022AD,222751
1450 DATA 0AD229039D1822A9019D14224C74
16A9018DC422A9008DC022AD682238E92C8DB9
22AD672238E90D8DBA22AEC0,227245
1460 DATA 22ADB922187DF1228DB9228582AD
BA22187DF5228DBA2285832046 15A000B1803D
8A22DD7E22F00DDD8222D0D1,232421
1470 DATA A9008DBF224C731CADC0228DBF22
20521D20FC1CDD7E22D00620331D4C4B1CDD82
22D00620331D4CB91C20461D,236412
1480 DATA 4C541CA9008DC32220521D20FC1C
DD8222D00620331D4C731CADC3221869018DC3
22C903F00620461D4C7B1CAD,240407
1490 DATA C0228DBF2220521D20FC1CDD7E22
D00620331D4C4B1C20461D4CA51C205E1DA900
8DC32220FC1CDD7E22D00620,244373
1500 DATA 331D4CB91CADC3221869018DC322
C904F00620461D4CC11C20171DDD8222D00920
9E1DA9008DC4226020461D20,248227
1510 DATA FC1C4CE51CAEBF22ADB922187DF1
228DBB228582ADBA22187DF5228DBC228583A5
82C99FB011A583C955B00B20,253777
1520 DATA 4615A000B1803D8A2260A900A200
60ADBF228DC022ADBB228DB922ADBC228DBA22
60ADBF2218690129038DBF22,258398
1530 DATA 60ADBF2238E90129038DBF2260AD
B9228582CDB3229 0068DB3224C761DCDB122B0
038DB122ADBA228583CDB422,263511
1540 DATA 90068DB4224C8E1DCDB222B0038D
B2222046 15A000BD8E2231801D822291806 0A9
008D03D28D05D2ADB42238ED,268158
1550 DATA B2228DC222ADB12238E9018DB122
8DBD22ADB2228DBE2238E9018DB222ADB32218
69018DB322ADB4221869018D,273050
1560 DATA B422A9008D2B22A9008DC5222054
1EC902D0F9EEC52220541EC902F0F62901D0E6
ADC522C901F00520E81EF0DA,278385
1570 DATA ADBD228582ADBE2285832046 15A0
00B1801D86229180EE2B2220541EC900F0E029
01D0B6A9018DC52220541EC9,283210
1580 DATA 00F00A2901D0A6EEC5224C2E1EAD
C522C901D0034CDF1D20E81EDD8A22F0B44CDF
1DADBD221869018DBD22CDB3,288351
1590 DATA 22D050AD2422186D2B228D2422AD
252269008D2522A9008D2B22ADB1228DBD22A9
008DC522A9868D01D2ADC222,292630
1600 DATA 8D00D2F003CEC222ADBE22186901
8DBE22CDB422F00DADBD22CDB122D00868684C
DF1D686860ADBD228582ADBE,298112
1610 DATA 2285832046 15A000BD8A223180DD
8222D00DBD8A2211809180EE2B22A90260DD7E
22D003A90160C900D00160A9,302731
1620 DATA 0360ADBD228582ADBE2238E901CD
B222F00D8583204615A000BD8A22318060A900
A20060ADFC02C921D00DA9FF,307760
1630 DATA 8DFC02AD272249FF8D2722AD2722
F0034C5FE4ADC122300D09A08D07D2A9808D06
D2CEC122A588F002C688A594,313096
1640 DATA F006C693D002C694ADC422F0034C
5FE4A900858985 8AAD0DD0290CF002E68AAD0D
D02901F002E689AD04D02902,318528
1650 DATA F002E6898D1ED0A587F002C687A5
86F002C686AD6522F006CE65224C9F1FA9018D
6522AD6622186901C907D002,323728
1660 DATA A9008D6622AC6622AE6722A9009D
FF119D0812B92D229D0012B9342 29D0112B93B
229D0212B942229D0312B949,327636
1670 DATA 229D0412B950229D0512B957229D
0612B95E229D0712AD68228D00D0ADAC22D031
A58B18692F8D01D0A58C1869,331870
1680 DATA 10AAA9009D7D129D7E129D82129D
8312A9409D7F129D8112A9A09D8012ADC622D0
03EEC102E690A5904A2901A8,336924
1690 DATA B9FF21859AA5902901A8A9138592
A9808591A901858FB9F1218599A900A68FBCF3
21A2049191C8CA10FAA699BD,343010
1700 DATA 1422F027BD0C22BC102218692EA6
8F9D02D09818690E9DF321A8A69AA904859BBD
F5219191C8CAC69B10F5C68F,348458
1710 DATA 3012A59138E9808591A592E90085
92E6994C3C204C5FE47070704D00300D0D0D0D
0D0D0D0D0D0D0D0D0D0D0D0D,351971
1720 DATA 0D0D0D0D0D0D0D0D0D0D0D0D0D0D
0D0D0D0D0D0D0D0D0D0D0D0D0D0D0D0D0D0D0D
0D0D0D0D0D0D0D0D0D0D0D0D,352556
1730 DATA 0D0D0D0D0D0D0D0D0D0D0D0D0D0D
0D0D0D0D0D0D0D0D0D0D0D0D2046152146292141 92
20707070707070707070 3046,354537
```

```
1740 DATA 5121704765213046792170703046
8D2141F720342734 1A000000000000000233532
1A000000000002C361A000000,356557
1750 DATA 33232F32251A0000000000000000
0000000002721 2D2500002F36253200000000
000000212E212C2F2700232F,357469
1760 DATA 2D303534292E270000000006669
6C6C4765720075704100696900000000000A2
B90000B4AFAD00A8B5A4B3AF,360748
1770 DATA AE000000000000F0F2E5F3F300
00F3F4E1F2F4000000004010E028F8D41004F8
E0D4E044A870D41F272E232A,365802
1780 DATA 3027292A2E302E2F2F3030040404
0303030202020202010101010100001000101
0001010101000001010 10707,366362
1790 DATA 0706060605050504040403030201
0002000085020508820 20F820200409FF0201
000100FFFF0001000000000,368048
1800 DATA 0000000000000000000000000000
0000000000000000000000000000000081402010
080402424320100804C22424,368762
1810 DATA 131008C82418181C1FF838181818
38F81F1C182424C80810132442C20408102043
810204081020400000000000,370904
1820 DATA 000000010100FFFFFF0001000101
0001FFFFFF044010040180200802C0300C03C0
300C033FCFF3FC009E9E0000,374290
1830 DATA 0054540100FF00000100FF9F559F
55000000000000000000000000000000000000
0000000000000000000000,375458
1840 DATA 0000000000000000000000000000
0000000000001000000FF000000000000000
0000000000000000001FF00,375970
1850 DATA 000100FFFF000100000000000000
0000000327010A64E810000000000000000000
0000000000000000000000,376883
```

●

CHECKSUM DATA

(See pgs. 7-10)

```
1 DATA 338,955,686,427,745,192,617,894
,445,496,549,150,852,324,104,7774
1030 DATA 121,368,374,344,911,909,145,
258,276,448,489,547,207,532,414,6343
1180 DATA 279,391,340,36,11,311,87,57,
219,152,322,325,387,110,337,3364
1330 DATA 73,335,409,357,275,510,28,43
,154,25,103,247,143,113,166,2981
1480 DATA 95,140,70,465,277,253,135,23
3,149,39,502,34,388,935,96,3811
1630 DATA 300,427,177,66,202,175,222,3
33,51,439,886,208,233,664,863,5246
1780 DATA 80,295,856,546,765,47,848,8,
3445
```

●

```
2 REM *** CASSETTE VERSION ***
65 IF PASS=2 THEN CLOSE #1:END
70 ? "READY CASSETTE AND PRESS RETURN"
;:OPEN #1,8,128,"C:":RESTORE 200:FOR X
=1 TO 35:READ N:PUT #1,N:NEXT X
200 DATA 0,31,221,19,255,19,169,60,141
,2,211,169,0,141,231,2,133,14,169,56,1
41,232,2,133,15,169,45
210 DATA 133,10,169,20,133,11,24,96
1860 DATA 0000000000000000000000000000
0000000000000000000000000000000000000000
0000000000000000000000,376883
1870 DATA 0000000000000000000000000000
0000000000000000000000000000000000000000
0000000000000000000000,376883
```

●

```
2 REM *** DISK VERSION ***
65 IF PASS=2 THEN PUT #1,224:PUT #1,2:
PUT #1,225:PUT #1,2:PUT #1,0:PUT #1,37
:CLOSE #1:END
70 ? "INSERT DISK WITH DOS, PRESS RETU
RN";:DIM IN$(1):INPUT IN$:OPEN #1,8,0,
"D:AUTORUN.SYS"
90 PUT #1,255:PUT #1,255:PUT #1,0:PUT
#1,37:PUT #1,29:PUT #1,52
```

●

Assembly listing.

```
;
;FILL 'ER UP! II
;
;BY TOM HUDSON
;

;
;ALPHABETIC CONSTANTS
;
CA      =   'A'-$20
CB      =   'B'-$20
CC      =   'C'-$20
CD      =   'D'-$20
CE      =   'E'-$20
CF      =   'F'-$20
CG      =   'G'-$20
CH      =   'H'-$20
CI      =   'I'-$20
CJ      =   'J'-$20
CK      =   'K'-$20
CL      =   'L'-$20
CM      =   'M'-$20
CN      =   'N'-$20
CO      =   'O'-$20
CP      =   'P'-$20
CQ      =   'Q'-$20
CR      =   'R'-$20
CS      =   'S'-$20
CT      =   'T'-$20
CU      =   'U'-$20
CV      =   'V'-$20
CW      =   'W'-$20
CX      =   'X'-$20
CY      =   'Y'-$20
CZ      =   'Z'-$20
CCOL    =   ':'-$20
;
;PAGE ZERO ITEMS
;
        ORG $80

LO      DS  1
HI      DS  1
PLOTX   DS  1
PLOTY   DS  1
LOHLD   DS  1
HIHLD   DS  1
SMTIM   DS  1
MOVTIM  DS  1
TIMER   DS  1
DEADFG  DS  1
HSHORT  DS  1
PX      DS  1
PY      DS  1
XI      DS  1
YI      DS  1
```

```
DESTNM DS  1
SHFLIP DS  1
DESTLO DS  1
DESTHI DS  1
SHTIM1 DS  1
SHTIM2 DS  1
SHTIMI DS  1
SHORTN DS  1
DIRSAV DS  1
HOLDSX DS  1
VBXHLD DS  1
CPYST  DS  1
CPYCNT DS  1
;
;MISCELLANEOUS MEMORY USAGE
;
PMAREA =   $1000
MISSLS =   PMAREA+384
PL0    =   PMAREA+512
PL1    =   PMAREA+640
PL2    =   PMAREA+768
PL3    =   PMAREA+896
DISP   =   $3000
;
;SYSTEM EQUATES
;
KEY    =   $2FC
CONSOL =   $D01F
PMBASE =   $D407
RANDOM =   $D20A
SETVBV =   $E45C
XITVBV =   $E45F
COLBK  =   $2C8
COLPF0 =   $2C4
COLPF1 =   $2C5
COLPF2 =   $2C6
COLPF3 =   $2C7
AUDC1  =   $D201
AUDC2  =   $D203
AUDC3  =   $D205
AUDC4  =   $D207
AUDF1  =   $D200
AUDF2  =   $D202
AUDF3  =   $D204
AUDF4  =   $D206
AUDCTL =   $D208
PRIOR  =   $026F
ATTRAC =   $4D
DMACTL =   $22F
DLISTL =   $230
GRACTL =   $D01D
NMIEN  =   $D40E
COLPM0 =   $2C0
COLPM1 =   $2C1
COLPM2 =   $2C2
COLPM3 =   $2C3
HPOSP0 =   $D000
HPOSP1 =   $D001
HPOSP2 =   $D002
HITCLR =   $D01E
P0PF   =   $D004
P1PL   =   $D00D
STICK  =   $278
STRIG  =   $284

       ORG $6000          ;ASSEMBLE ADDR.
       LOC $1400          ;ACTUAL ADDRESS

;
;RELOCATE PROGRAM (DISK VERSION ONLY)
;

MOVEPG LDA #$25           ;FOR DISK ONLY,
       STA HI             ;THIS SECTION
       LDA #$14           ;MOVES THE
       STA DESTHI         ;PROGRAM TO
       LDA #0             ;ITS OPERATIONAL
       STA LO             ;MEMORY LOCATION
       STA DESTLO         ;OF $1400.
       LDY #0
COPYLP LDA (LO),Y
       STA (DESTLO),Y
       INC LO
       INC DESTLO
       BNE INCEND
       INC HI
       INC DESTHI
INCEND LDA DESTHI
       CMP #DIR/256
       BNE COPYLP
       LDA DESTLO
       CMP #DIR&255
       BNE COPYLP
       JMP FILLUP

;
;MAIN PROGRAM STARTS HERE
;

FILLUP LDA #$00           ;TURN OFF...
       STA DMACTL         ;DMA
       STA GRACTL         ;GRAPHICS
       STA COLBK          ;BLACK BACKGND
       LDA #$74           ;BLUE
       STA COLPF0         ;COLOR0
       LDA #$C4           ;GREEN
       STA COLPF1         ;COLOR1
       LDA #$0A           ;WHITE
       STA COLPF2         ;COLOR2
       LDA #$34           ;RED
       STA COLPF3         ;COLOR3
       LDA #TITLDL&255    ;SETUP...
       STA DLISTL         ;TITLE...
       LDA #TITLDL/256    ;DISPLAY...
       STA DLISTL+1       ;LIST'
       LDA #$2E           ;TURN ON...
       STA DMACTL         ;DMA
       LDA #3             ;TURN ON...
       STA GRACTL         ;GRAPHICS
       JMP CKSTRT         ;WAIT FOR START

START  JSR $E465          ;INIT SOUNDS
       LDA #0             ;CLEAR OUT
       LDX #127           ;ALL ZERO PAGE
CLPG0  STA $80,X          ;USER MEMORY
       DEX
       BPL CLPG0
       STA SHORTF         ;STOP SHORTS
       STA SHORTF+1
       STA SHORTF+2
       STA SHORTF+3
       LDA #$11           ;P/M PRIORITY
       STA PRIOR
       LDA #1             ;DON'T SHOW
       STA SHOOFF         ;PLAYER OR STAR
       STA FILLON         ;WE STILL MUST
       JSR PMCLR          ;CLEAR P/M AREA
       LDA #64            ;AND SET UP THE
       STA STRHGT         ;STAR'S HEIGHT
       LDA #128           ;AND
       STA STRHOR         ;HORIZONTAL POSITION
       LDA #$D0           ;NOW LET'S
       LDX #4             ;ZERO OUT
ZSCLP  STA SCOLIN+4,X     ;THE SCORE
       STA SCOLIN+15,X    ;AREAS!
       DEX
       BPL ZSCLP
       LDX #5
ZSCLP2 STA SCOLN2+12,X
       DEX
       BPL ZSCLP2
       LDA #0             ;THESE ITEMS
       STA FILLON         ;MUST BE SET
       STA DEADFG         ;TO ZERO ON
```

```
        STA NOCCHG              ;STARTUP OR
        STA HITCLR              ;ELSE WE'LL
        STA DMACTL              ;WIND UP WITH
        STA NMIEN               ;NASTY THINGS
        STA HASDRN              ;HAPPENING!
        STA HSHORT
        STA AUDCTL
        LDX #5                  ;LET'S ZERO
CMSLP   STA SCORE,X             ;OUT THE SCORE
        DEX                     ;COUNTER...
        BPL CMSLP
        STA LEVEL               ;AND LEVEL #!
        LDA #3                  ;WE START WITH
        STA LIVES               ;3 LIVES
        ORA #$90                ;AND PUT THEM IN
        STA SCOLN2+19           ;THE SCORE LINE
        LDA #$0A                ;NEXT WE SET UP
        STA COLPF0              ;THE COLORS WE
        LDA #$24                ;WANT TO USE.
        STA COLPF1
        LDA #$94
        STA COLPF2
        LDA #$C4
        STA COLPF3
        LDA #0
        STA COLBK
        LDA #$76
        STA COLPM1
        LDA #$3F
        STA COLPM2
        STA COLPM3
        LDA #$34
        STA COLPM0
        LDA #DLIST&255          ;WE'D BETTER TELL
        STA DLISTL              ;THE COMPUTER WHERE
        LDA #DLIST/256          ;OUR DISPLAY LIST
        STA DLISTL+1            ;IS LOCATED!
        LDY #INTRPT&255         ;TELL WHERE THE
        LDX #INTRPT/256         ;VERTICAL BLANK
        LDA #6                  ;INTERRUPT IS
        JSR SETVBV              ;AND SET IT!
        LDA #PMAREA/256         ;HERE'S OUR P/M
        STA PMBASE              ;GRAPHICS AREA!
        LDA #$2E                ;TURN ON THE
        STA DMACTL              ;DMA CONTROL
        LDA #$3                 ;AND
        STA GRACTL              ;GRAPHICS CONTROL!
        LDA #$40                ;ENABLE VBI
        STA NMIEN
        JMP CLRDSP
PMCLR   LDA #0                  ;CLEAR OUT
        LDX #127                ;THE P/M AREA:
PMICLR  STA MISSLS,X            ;MISSILES,
        STA PL0,X               ;PLAYER 0,
        STA PL1,X               ;PLAYER 1,
        STA PL2,X               ;PLAYER 2,
        STA PL3,X               ;AND PLAYER 3!
        DEX
        BNE PMICLR              ;LOOP UNTIL DONE
RETURN  RTS                     ;WE'RE DONE!
;
;PLOT ADDRESS CALCULATOR
;
;MULTIPLY PLOTY BY 40, THEN CALCULATE ADDRESS
;OF THE SCREEN MEMORY TO BE ALTERED.
;
PLOTCL  LDA PLOTY
        ASL A
        STA LO
        LDA #0
        STA HI                  ;*2
        ASL LO
        ROL HI                  ;*4
        ASL LO
        LDA LO
        STA LOHLD
        ROL HI                  ;*8
        LDA HI
        STA HIHLD
        ASL LO
        ROL HI                  ;*16
        ASL LO
        ROL HI                  ;*32
        LDA LO
        CLC
        ADC LOHLD
        STA LO
        LDA HI
        ADC HIHLD
        STA HI                  ;+*8=*40
        LDA #DISP&255
        CLC
        ADC LO
        STA LO
        LDA #DISP/256
        ADC HI
        STA HI                  ;+DISPLAY START
        LDA PLOTX               ;MASK X POSITION
        AND #3
        TAX
        LDA PLOTX
        LSR A
        LSR A
        CLC
        ADC LO
        STA LO
        LDA HI
        ADC #0                  ;LO & HI NOW HOLD
        STA HI                  ;THE ADDRESS!
        RTS                     ;EXIT!
;
;CLEAR THE DISPLAY MEMORY
;
CLRDSP  LDX #0                  ;THIS ROUTINE WILL
        STX PLOTX               ;CLEAR THE SCREEN RAM.
        LDX #0                  ;IT GETS THE ADDRESS
DLOOP2  STX PLOTY               ;OF THE BEGINNING OF
        JSR PLOTCL              ;EACH GR.7 LINE
        LDX PLOTY               ;THEN ZEROES OUT
        LDA #$00                ;EACH OF THE
        LDY #39                 ;40 BYTES (0-39)
DLOOP3  STA (LO),Y              ;IN THE LINE.
        DEY
        BPL DLOOP3
        INX
        CPX #86
        BNE DLOOP2
;
;DRAW THE COLOR 1 BORDER
;
        LDA #3                  ;THIS ROUTINE
        STA BORNUM              ;DRAWS THE 4 LINES
BORDER  LDX BORNUM              ;THAT MAKE UP THE
        LDA BXSTRT,X            ;WHITE GR.7 BORDER
        STA PLOTX               ;ON THE SCREEN.
        LDA BYSTRT,X
        STA PLOTY
        LDA BXINC,X
        STA BDINCX
        LDA BYINC,X
        STA BDINCY
        LDA BORCNT,X
        STA BDCNT
DRAWLN  JSR PLOTCL
        LDA COLOR1,X
        LDY #0
        ORA (LO),Y
        STA (LO),Y
        LDA PLOTX
        CLC
        ADC BDINCX
        STA PLOTX
        LDA PLOTY
        CLC
```

```
       ADC BDINCY
       STA PLOTY
       DEC BDCNT
       BNE DRAWLN
       DEC BORNUM
       BPL BORDER
;
;THIS SECTION STARTS OFF EACH LEVEL
;
       LDA #80              ;POSITION THE
       STA PX               ;PLAYER
       LDA #84
       STA PY
       LDA LEVEL            ;INCREMENT THE
       CLC                  ;LEVEL NUMBER
       ADC #1
       STA LOWK
       LDA #0               ;ZERO OUT
       STA SHORTF
       STA SHORTF+1
       STA SHORTF+2
       STA SHORTF+3
       STA CURLO            ;CURRENT TALLY
       STA CURHI            ;WORK AREA
       STA HIWK
       LDA #$FF             ;TELL DECIMAL CONVERTER
       STA SLLOC            ;NOT TO PLACE RESULT
       JSR CNVDEC           ;CONVERT LEVEL #
       LDA DECIMAL+1        ;GET DECIMAL LEVEL #
       ORA #$90             ;ADD COLOR
       STA SCOLN2+3         ;PUT IN SCORE LINE
       LDA DECIMAL          ;SAME FOR 2ND
       ORA #$90             ;LEVEL #
       STA SCOLN2+4         ;DIGIT
       LDX LEVEL            ;GET THIS LEVEL'S
       LDA TGTLO,X          ;PARAMETERS
       STA LOWK
       LDA TGTHI,X
       STA HIWK
       LDA STARSP,X
       STA STRSPD
       LDA SHTIME,X
       STA SHTIM1
       STA SHTIM2
       LDA #4
       STA SLLOC
       JSR CNVDEC           ;SHOW TARGET AMOUNT
;
;CLEAR OUT THE TRACKING TABLE THAT
;REMEMBERS WHERE THE PLAYER MOVED
;
CLRTRK LDA #0
       STA SHOOFF
       TAX
CLRTLP STA DIR,X            ;CLEAR DIRECTION
       STA LGTH,X           ;AND LENGTH ENTRIES
       DEX
       BNE CLRTLP
       STA MOVIX            ;CLEAR MOVEMENT INDEX
       STA DRAWFG           ;AND DRAW FLAG
GETSTK LDA PAUSE            ;GAME PAUSED?
       BNE GETSTK           ;YES, LOOP AND WAIT.
       LDA #$FD             ;DO 'WARBLE' SOUND
       STA AUDF1            ;USING SOUND
       LDA #$FE             ;CHANNELS 1-3
       STA AUDF2
       LDA #$FF
       STA AUDF3
       LDA #$A3
       STA AUDC1
       STA AUDC2
       STA AUDC3
       LDA #0               ;NO ATTRACT MODE!
       STA ATTRAC
       LDA HSHORT           ;DID SHORT HIT US?
       BNE JCRSH            ;YES! WE'RE DEAD!
       LDA DEADFG           ;DID STAR HIT US?
       BEQ ALIVE            ;NO!
       LDX LEVEL            ;IT HIT US--
       LDA KILLFG,X         ;UNCONDITIONAL KILL?
       BNE JCRSH            ;YES! WE'RE DEAD!!!
       LDA PX               ;NO, IF WE'RE ON A
       STA PLOTX            ;WHITE LINE (COLOR 1)
       LDA PY               ;THEN WE'RE ALIVE!
       STA PLOTY
       JSR PLOTCL
       LDY #0
       LDA BITSON,X
       AND (LO),Y
       CMP COLOR1,X         ;ON COLOR 1?
       BEQ ALIVE            ;YES (WHEW!)
JCRSH  JMP CRASH            ;GO KILL PLAYER.
ALIVE  LDA MOVTIM           ;PLAYER MOVING?
       BEQ GOTSTK           ;YES--GET STICK.
       JMP MOVSTR           ;NO, MOVE STAR.
JGSTK  JMP GETSTK           ;GO GET STICK
GOTSTK LDA #4               ;SET UP THE
       STA MOVTIM           ;MOVEMENT TIMER
       LDA STICK            ;GET THE STICK
       STA STKHLD           ;AND SAVE IT
       TAX                  ;THEN LOOK UP
       LDA XD,X             ;X DIRECTION
       CLC
       ADC XD,X
       STA XI               ;AND
       LDA YD,X             ;Y DIRECTION
       CLC
       ADC YD,X
       STA YI
       ORA XI               ;ANY MOVEMENT?
       BEQ JGSTK            ;NO, TRY AGAIN.
       LDA PX               ;INCREMENT
       CLC                  ;PLAYER X
       ADC XI               ;POSITION AND
       STA CKX              ;HOLD IT...
       CMP #159             ;OFFSCREEN?
       BCS JGSTK            ;YES!
       STA PLOTX            ;NO, SAVE IT
       SEC
       SBC XD,X
       STA PXWC
       LDA PY               ;INCREMENT
       CLC                  ;PLAYER Y
       ADC YI               ;POSITION AND
       STA CKY              ;HOLD IT...
       CMP #85              ;OFFSCREEN?
       BCS JGSTK            ;YES!
       STA PLOTY            ;NO, SAVE IT
       SEC
       SBC YD,X
       STA PYWC
       JSR PLOTCL           ;LOCATE NEW PLAYER
       LDY #0               ;POSITION.
       LDA BITSON,X
       AND (LO),Y
       STA CKV              ;SAVE THE 'LOCATE'.
       STX CKVX
       LDA PXWC             ;CHECK THE
       STA PLOTX            ;POSITION NEXT
       LDA PYWC             ;TO THE ONE WE'RE
       STA PLOTY            ;NOW IN...
       JSR PLOTCL
       LDY #0
       LDA BITSON,X
       AND (LO),Y
       PHA                  ;AND SAVE IT!
       LDA STRIG            ;TRIGGER PRESSED?
       BNE NOTDRN           ;NO!
       PLA                  ;OK TO DRAW?
       BNE JGS              ;NO!!
       JMP DRAWIN           ;YES, GO DRAW.
NOTDRN PLA                  ;NOT DRAWING--ARE WE
       CMP COLOR1,X         ;ON COLOR 1?
       BNE JGS              ;NO, TRY AGAIN
```

```
        LDA CKV             ;ARE WE MOVING
        LDX CKVX            ;ONTO ANOTHER
        CMP COLOR1,X        ;COLOR 1?
        BNE JGS             ;NO! TRY AGAIN.
        LDA CKX             ;ALL'S WELL...
        STA PX              ;UPDATE PX
        LDA CKY             ;AND
        STA PY              ;PY.
JGS     JMP GETSTK          ;GET STICK.
;
;THIS ROUTINE HANDLES THE DRAW FUNCTION.
;
DRAWIN  LDA DRAWFG          ;ALREADY DRAWING?
        BNE DRAWOK          ;YES!
        STA MOVIX           ;NO, THIS IS THE
        LDA STKHLD          ;FIRST TIME--SET UP
        STA DIR             ;INITIAL DRAWING
        LDA #1              ;VARIABLES.
        STA DRAWFG
        STA HASDRN
        LDA PX
        STA INIX
        STA MINX
        STA MAXX
        LDA PY
        STA INIY
        STA MINY
        STA MAXY
DRAWOK  LDA CKV             ;DID WE
        LDX CKVX            ;RUN INTO ANOTHER
        CMP COLOR2,X        ;COLOR 2?
        BNE NOCRSH          ;NO, WE'RE OK.
        JMP CRASH           ;CRRAAASSSHHH!
NOCRSH  LDX MOVIX           ;UPDATE THE
        LDA STKHLD          ;TRACKING
        CMP DIR,X           ;TABLES WITH
        BEQ SAMDIR          ;DIRECTION
        INC MOVIX           ;INFORMATION.
        INX
        STA DIR,X
        LDA #0
        STA LGTH,X
SAMDIR  INC LGTH,X
        LDA #3
        STA BDCNT
        LDA PX              ;NOW PLOT THE
        STA PLOTX           ;LINE WE'RE
        LDA PY              ;DRAWING...
        STA PLOTY
CCLOOP  JSR PLOTCL
        LDY #0
        LDA (LO),Y
        AND BITOFF,X
        ORA COLOR2,X        ;IN COLOR 2.
        STA (LO),Y
        DEC BDCNT
        BEQ CKCOLR
        LDY MOVIX
        LDX DIR,Y
        LDA XD,X
        CLC
        ADC PLOTX
        STA PLOTX
        LDA YD,X
        CLC
        ADC PLOTY
        STA PLOTY
        JMP CCLOOP
CKCOLR  LDA PLOTX           ;UPDATE X POS.
        STA PX
        CMP MAXX            ;CHECK MINIMUM
        BCC TMINX           ;AND MAXIMUM
        STA MAXX            ;X & Y VALUES
        JMP CHKYMM          ;AND UPDATE IF
TMINX   CMP MINX            ;NECESSARY
        BCS CHKYMM
        STA MINX

CHKYMM  LDA PLOTY
        STA PY
        CMP MAXY
        BCC TMINY
        STA MAXY
        JMP ENDMM
TMINY   CMP MINY
        BCS ENDMM
        STA MINY
ENDMM   LDX CKVX            ;DID WE DRAW
        LDA CKV             ;INTO
        CMP COLOR1,X        ;COLOR 1?
        BEQ ENDLIN          ;YES! END OF LINE!
        JMP GETSTK          ;NO, GO GET STICK.
ENDLIN  LDA #0              ;WE AREN'T
        STA DRAWFG          ;DRAWING ANYMORE
        JSR SEARCH          ;SEARCH AND FILL!!
        LDA CURLO           ;GET CURRENT VALUE
        STA LOWK
        LDA CURHI
        STA HIWK
        LDA #15             ;PUT AT 15TH
        STA SLLOC           ;POS. IN SCOLIN
        JSR CNVDEC          ;CONVERT TO DECIMAL
        LDA #1              ;NOW REDRAW THE
        STA RDRCOL          ;PLAYER'S PATH IN
        JSR REDRAW          ;COLOR 1 (WHITE).
        LDX LEVEL           ;CHECK TO SEE
        LDA CURLO           ;IF WE'VE HIT
        SEC                 ;THE TARGET.
        SBC TGTLO,X
        STA LOWK
        LDA CURHI
        SBC TGTHI,X
        STA HIWK            ;HIT TARGET?
        BPL NEWLVL          ;YES--NEW LEVEL!
        JMP CLRTRK          ;NO, GO CLEAR TRACK
NEWLVL  LDA LEVEL           ;IF LEVEL < 15
        CMP #15             ;THEN
        BEQ NOLINC          ;INCREMENT
        INC LEVEL           ;LEVEL
;
;INCREASE SCORE HERE
;
NOLINC  ASL LOWK            ;SCORE INC =
        ROL HIWK            ;TGT-CUR * 2
        LDA #$FF            ;DON'T PLACE
        STA SLLOC           ;THE RESULT!
        JSR CNVDEC          ;CONVERT TO DECIMAL
        LDX #5              ;AND ADD TO SCORE
        LDY #0
SCOLP   LDA DECIMAL,Y
        CLC
        ADC SCORE,X
        CMP #10
        BMI NOCARY
        SEC
        SBC #10
        STA SCORE,X
        INC SCORE-1,X
        JMP NXSPOS
NOCARY  STA SCORE,X
NXSPOS  INY
        DEX
        BPL SCOLP
        LDX #5              ;NOW PLACE THE
SHSLP   LDA SCORE,X         ;SCORE IN
        ORA #$00            ;SCORE LINE #2
        STA SCOLN2+12,X
        DEX
        BPL SHSLP
        LDA #1              ;STOP VBI FOR
        STA FILLON          ;A MOMENT
        STA SHOOFF
        JSR PMCLR           ;CLEAR P/M AREA
        LDA #64             ;INITIALIZE
        STA STRHGT          ;THE
```

```
        LDA #128            ;STAR
        STA STRHOR          ;POSITION
        LDA #0              ;VBI ON AGAIN
        STA FILLON
        JMP CLRDSP          ;GO CLEAR DISPLAY!
;
;THIS SECTION HANDLES PLAYER'S DEATH
;
CRASH   LDA #0              ;NO WARBLE SOUND
        STA AUDC1
        STA AUDC2
        STA AUDC3
        LDA #1              ;NO PLAYER COLOR
        STA NOCCHG          ;CHANGE IN VBI
        LDA #15             ;SET BRIGHTNESS OF
        STA DEDBRT          ;PLAYER DEATH.
TIMRST  LDA #5              ;SET DEATH TIMER
        STA TIMER           ;TO 5 JIFFIES.
DEADCC  LDA DEDBRT          ;MOVE BRIGHTNESS
        STA AUDC1           ;TO DEATH SOUND VOLUME
        LDA RANDOM          ;GET RANDOM
        AND #$1F            ;DEATH SOUND
        STA AUDF1           ;FREQUENCY
        LDA RANDOM          ;GET RANDOM
        AND #$F0            ;DEATH COLOR
        ORA DEDBRT          ;ADD BRITE
        STA COLPF1          ;PUT IN LINE COLOR
        STA COLPM1          ;AND PLAYER COLOR
        LDA TIMER           ;TIMER DONE YET?
        BNE DEADCC          ;NO, GO CHANGE COLOR.
        DEC DEDBRT          ;DECREMENT BRIGHTNESS
        BPL TIMRST          ;IF MORE, GO DO IT.
        DEC LIVES           ;1 LESS LIFE
        LDA LIVES           ;GET # LIVES
        ORA #$90            ;ADD COLOR
        STA SCOLN2+19       ;AND DISPLAY'
        CMP #$90            ;ZERO LIVES?
        BNE NOTDED          ;NO!
        LDA #GOMSG&255      ;WE'RE COMPLETELY
        STA SCDL            ;DEAD, SHOW
        LDA #GOMSG/256      ;'GAME OVER'
        STA SCDL+1          ;MESSAGE
CKSTRT  LDA CONSOL          ;WAIT FOR START
        AND #1              ;KEY...
        BNE CKSTRT          ;NOT PRESSED--LOOP.
RELEAS  LDA CONSOL          ;KEY PRESSED, NOW
        AND #1              ;WAIT FOR RELEASE!
        BEQ RELEAS          ;NOT RELEASED YET!
        LDA #SCOLIN&255     ;PUT SCORE
        STA SCDL            ;LINE BACK
        LDA #SCOLIN/256     ;IN DISPLAY
        STA SCDL+1          ;LIST...
        JMP START           ;AND START GAME!
;
;THIS SECTION PLACES PLAYER AT A RANDOM
;LOCATION IF THERE ARE MORE LIVES LEFT.
;
NOTDED  LDA #1              ;DON'T SHOW
        STA SHOOFF          ;PLAYER
NEWLOC  LDA RANDOM          ;GET RANDOM X
        AND #$FE            ;MUST BE EVEN
        CMP #159            ;AND ON SCREEN
        BCS NEWLOC
        STA PLOTX
CSHY    LDA RANDOM          ;GET RANDOM Y
        AND #$7E            ;MUST BE EVEN
        CMP #85             ;AND ON SCREEN
        BCS CSHY
        STA PLOTY
        JSR PLOTCL
        LDY #0
        LDA BITSON,X
        AND (LO),Y          ;IS LOCATION ON
        CMP COLOR1,X        ;COLOR 1?
        BNE NEWLOC          ;NO, TRY AGAIN.
        JSR PMCLR           ;IT'S OK, CLEAR P/M
        LDA PLOTX           ;SAVE
        STA PX              ;THE PLAYER'S
        LDA PLOTY           ;NEW
        STA PY              ;COORDINATES.
        LDA #0              ;REDRAW THE
        STA RDRCOL          ;PLAYER'S TRACK
        LDA HASDRN          ;IN COLOR 0
        BEQ JCTRK
        JSR REDRAW
        LDA INIX            ;THIS PART IS
        STA PLOTX           ;NEEDED TO PLOT
        LDA INIY            ;A COLOR 1 BLOCK
        STA PLOTY           ;AT THE START OF
        JSR PLOTCL          ;THE PLAYER'S TRACK
        LDY #0              ;AFTER IT IS ERASED.
        LDA BITOFF,X        ;(NOBODY'S PERFECT!)
        AND (LO),Y
        ORA COLOR1,X
        STA (LO),Y
JCTRK   LDA #$24            ;RESTORE DRAW LINE
        STA COLPF1          ;COLOR
        LDA #0
        STA NOCCHG
        STA HITCLR
        STA DEADFG
        STA HSHORT
        JMP CLRTRK          ;AND GO START NEW TRACK.
;
;THIS ROUTINE USES THE TRACKING TABLES,
;DIR AND LGTH, TO REDRAW THE LINE THE
;PLAYER DREW.  RDRCOL INDICATES THE COLOR
;DESIRED.
;
REDRAW  LDA INIX
        STA REX
        LDA INIY
        STA REY
        LDA #0
        STA X
REDXLP  LDX X
        LDA DIR,X
        STA REDIR
        LDA LGTH,X
        STA LGTHY
        LDA #1
        STA Y
REDYLP  LDA #3
        STA TIMES
TIMES3  LDX REX
        STA PLOTX
        LDA REY
        STA PLOTY
        JSR PLOTCL
        LDY #0
        LDA RDRCOL
        BNE RDC1
        LDA BITOFF,X
        AND (LO),Y
        STA (LO),Y
        JMP SETNRP
ENDRD   LDA #0
        STA DRAWFG
        RTS
RDC1    LDA BITOFF,X
        AND (LO),Y
        ORA COLOR1,X
        STA (LO),Y
SETNRP  DEC TIMES
        BEQ NXTY
        LDX REDIR
        LDA REX
        CLC
        ADC XD,X
        STA REX
        LDA REY
        CLC
        ADC YD,X
        STA REY
```

```
       JMP TIMES3
NXTY   INC Y
       LDA Y
       CMP LGTHY
       BEQ JNRD
       BCS NXTX
JNRD   JMP REDYLP
NXTX   INC X
       LDA X
       CMP MOVIX
       BEQ JRXLP
       BCS ENDRD
JRXLP  JMP REDXLP
;
;2-BYTE DECIMAL CONVERTER.  CONVERTS
;A 2-BYTE BINARY NUMBER TO A 5-BYTE
;DECIMAL NUMBER.  WILL PLACE THE
;DECIMAL NUMBER IN SCOLIN IF DESIRED
;(SLLOC DETERMINES POSITION).
;
CNVDEC LDX #4
       LDA #0
CDLP   STA DECIMAL,X
       DEX
       BPL CDLP
       LDX #4
CKMAG  LDA HIWK
       CMP HIVALS,X
       BEQ CKM2
       BCS SUBEM
       BCC NOSUB
CKM2   LDA LOWK
       CMP LOVALS,X
       BCS SUBEM
NOSUB  DEX
       BPL CKMAG
       JMP SHOWIT
SUBEM  LDA LOWK
       SEC
       SBC LOVALS,X
       STA LOWK
       LDA HIWK
       SBC HIVALS,X
       STA HIWK
       INC DECIMAL,X
       JMP CKMAG
SHOWIT LDX #$4
       LDY SLLOC
       BMI SHEND
SHOLP  LDA DECIMAL,X
       ORA #$D0
       STA SCOLIN,Y
       INY
       DEX
       BPL SHOLP
SHEND  RTS
;
;THIS ROUTINE MOVES THE STAR AROUND ON
;THE PLAYFIELD.  THE STAR IS ROTATED AND
;PLOTTED (IN A PLAYER) IN THE VBI.
;
MOVSTR LDA SMTIM           ;TIME TO MOVE?
       BEQ MSTR            ;YES, GO DO IT
       JMP TRYSHO          ;NO, GET STICK
MSTR   LDA STRSPD          ;SET MOVEMENT TIMER
       STA SMTIM           ;WITH STAR SPEED
       LDA STRHGT          ;ADJUST P/M
       SEC                 ;COORDINATES TO
       SBC #13             ;MATCH PLAYFIELD
       STA STRLY           ;PLOTTING
       LDA STRHOR          ;COORDINATES.
       SEC
       SBC #44
       STA STRLX
       LDA RANDOM          ;WANT TO CHANGE
       CMP #240            ;THE STAR'S DIRECTION?
       BCC SAMSTD          ;NO, USE SAME.
NEWDIR LDA RANDOM          ;GET RANDOM
       AND #7              ;DIRECTION
       JMP DIRCHK
SAMSTD LDA STRDIR          ;GET OLD DIRECTION.
DIRCHK TAX                 ;CHECK TO SEE
       STA TMPDIR          ;IF STAR WILL
       LDA STRLX           ;BUMP INTO ANY
       CLC                 ;PLAYFIELD
       ADC STRDTX,X        ;OBJECT.
       STA PLOTX
       LDA STRLY
       CLC
       ADC STRDTY,X
       STA PLOTY
       JSR PLOTCL
       LDY #0
       LDA BITSON,X
       AND (LO),Y          ;ANY COLLISION?
       BEQ WAYCLR          ;NO, ALL CLEAR!
       LDA #15             ;HIT SOMETHING,
       STA BSCNT           ;START BUMP SOUND AND
       BNE NEWDIR          ;GET NEW DIRECTION.
WAYCLR LDA PLOTX           ;ADJUST STAR
       CLC                 ;COORDINATES
       ADC #44             ;BACK TO P/M
       STA STRHOR          ;COORDINATES
       LDA PLOTY           ;FROM PLAYFIELD.
       CLC
       ADC #13
       STA STRHGT
       LDA TMPDIR          ;SET DIRECTION
       STA STRDIR
MOVESH LDA #3              ;CHECK ALL
       STA SHORTN          ;4 SHORTS
SHMVLP LDX SHORTN          ;GET SHORT #
       LDA SHORTF,X        ;SHORT ALIVE?
       BEQ NXTSM           ;NO
       LDA SHORTX,X        ;GET X
       STA PLOTX           ;COORDINATE
       LDA SHORTY,X        ;AND Y
       STA PLOTY           ;COORDINATE
       JSR PLOTCL          ;IS SHORT
       LDY #0              ;ON...
       LDA BITSON,X
       AND (LO),Y
       CMP COLOR1,X        ;COLOR1?
       BEQ MOVEIT          ;YUP!
KILLSH LDX SHORTN          ;STOP THIS SHORT
       LDA #0              ;BY TURNING
       STA SHORTF,X        ;FUNCTION FLAG OFF
NXTSM  DEC SHORTN          ;MORE SHORTS?
       BPL SHMVLP          ;YES!
       JMP TRYSHO          ;NO.
MOVEIT LDA #3              ;TRY 4
       STA TRIES           ;DIRECTIONS
       LDX SHORTN          ;GET SHORT #
       LDA SHORTD,X        ;AND DIRECTION
       STA DIRSAV          ;SAVE IT
TRYMOV LDX SHORTN          ;GET SHORT #
       LDY DIRSAV          ;AND DIRECTION
       LDA SHORTX,X        ;FIND OUT
       CLC                 ;WHERE THE
       ADC DIRX,Y          ;SHORT WILL
       CMP #159            ;BE NEXT
       BCS NXTTRN          ;POSITION,
       STA PLOTX
       LDA SHORTY,X
       CLC
       ADC DIRY,Y
       CMP #85
       BCS NXTTRN
       STA PLOTY
       JSR PLOTCL
       LDY #0
       LDA BITSON,X
       AND (LO),Y          ;IS IT OVER
       CMP COLOR1,X        ;COLOR 1?
```

```
        BEQ GOTDIR              ;YES! IT'S OK!
NXTTRN  DEC TRIES               ;MORE DIRECTIONS?
        BMI KILLSH              ;NO!
        LDX TRIES               ;GET NEXT
        LDA DIRSAV              ;TRIAL DIRECTION
        CLC
        ADC DADD,X
        AND #3
        STA DIRSAV
        JMP TRYMOV              ;AND TRY TO MOVE!
GOTDIR  LDX SHORTN              ;GET SHORT #
        LDA PLOTX               ;SAVE ALL
        STA SHORTX,X            ;NEW SHORT
        LDA PLOTY               ;POSITION
        STA SHORTY,X            ;AND DIRECTION
        LDA DIRSAV              ;VALUES!
        STA SHORTD,X
        JMP NXTSM               ;DO NEXT SHORT!

;       THIS ROUTINE GENERATES NEW
;       SHORTS AT THE PROPER TIME IF
;       ANY ARE INACTIVE.

TRYSHO  LDA SHTIM2              ;READY TO START ONE?
        BEQ TRYSH2              ;YES!
        JMP GETSTK              ;NO!
TRYSH2  LDA SHTIM1              ;RESET THE
        STA SHTIM2              ;SHORT TIMER
        LDX #3                  ;SEARCH FOR
SHSCAN  LDA SHORTF,X            ;INACTIVE SHORT
        BEQ STRTSH              ;GOT ONE!!!
        DEX
        BPL SHSCAN
        JMP GETSTK              ;NONE FOUN
STRTSH  STX HOLDSX
STRTSX  LDA RANDOM              ;RANDOM SHORT X
        CMP #160
        BCS STRTSX
        STA PLOTX
STRTSY  LDA RANDOM              ;RANDOM SHORT Y
        AND #$7F
        CMP #85
        BCS STRTSY
        STA PLOTY
        JSR PLOTCL
        LDY #0
        LDA BITSON,X
        AND (LO),Y
        CMP COLOR1,X            ;ON COLOR 1?
        BNE STRTSX              ;NO, TRY AGAIN!
        LDX HOLDSX
        LDA PLOTX               ;SAVE ALL
        STA SHORTX,X            ;SHORT
        LDA PLOTY               ;PARAMETERS
        STA SHORTY,X
        LDA RANDOM
        AND #3
        STA SHORTD,X
        LDA #1                  ;TELL PROGRAM
        STA SHORTF,X            ;SHORT IS ALIVE!
        JMP GETSTK              ;LOOP BACK
;
;SEARCH FOR FILLABLE AREA
;
;THIS SECTION SEARCHES FOR THE AREA TO
;BE FILLED.  IT IS SO COMPLICATED THAT
;EXPLANATION OF ITS FINER DETAILS
;WOULD BE ALMOST IMPOSSIBLE WITHOUT
;WRITING ANOTHER COMPLETE ARTICLE.  AT
;ANY RATE, IT WORKS.  THOSE WITH ANY
;SPECIFIC QUESTIONS SHOULD WRITE ME,
;CARE OF A.N.A.L.O.G.
;
SEARCH  LDA #1
        STA FILLON
        LDA #0
        STA D
        LDA STRHOR
        SEC
        SBC #44
        STA SX
        LDA STRHGT
        SEC
        SBC #13
        STA SY
FINDCL  LDX D
        LDA SX
        CLC
        ADC SXD,X
        STA SX
        STA PLOTX
        LDA SY
        CLC
        ADC SYD,X
        STA SY
        STA PLOTY
        JSR PLOTCL
        LDY #0
        LDA (LO),Y
        AND BITSON,X
        CMP COLOR1,X
        BEQ FINDC2
        CMP COLOR2,X
        BNE FINDCL
        LDA #0
        STA TD
        JMP FOUND2
FINDC2  LDA D
        STA TD
        JSR DECD
FC2A    JSR SRCHLC
        CMP COLOR1,X
        BNE FC2B
        JSR GRABEM
        JMP FINDC2
FC2B    CMP COLOR2,X
        BNE FC2C
        JSR GRABEM
        JMP OUTLIN
FC2C    JSR INCD
        JMP FC2A
FOUND2  LDA #0
        STA TRIES
        JSR DECD
FND2A   JSR SRCHLC
        CMP COLOR2,X
        BNE FND2B
        JSR GRABEM
        JMP FOUND2
FND2B   LDA TRIES
        CLC
        ADC #1
        STA TRIES
        CMP #3
        BEQ FINDC1
        JSR INCD
        JMP FND2A
FINDC1  LDA D
        STA TD
        JSR DECD
FC1A    JSR SRCHLC
        CMP COLOR1,X
        BNE FC1B
        JSR GRABEM
        JMP FINDC2
FC1B    JSR INCD
        JMP FC1A
OUTLIN  JSR PLSXSY
        LDA #0
        STA TRIES
OUTLA   JSR SRCHLC
        CMP COLOR1,X
        BNE OUTLB
        JSR GRABEM
```

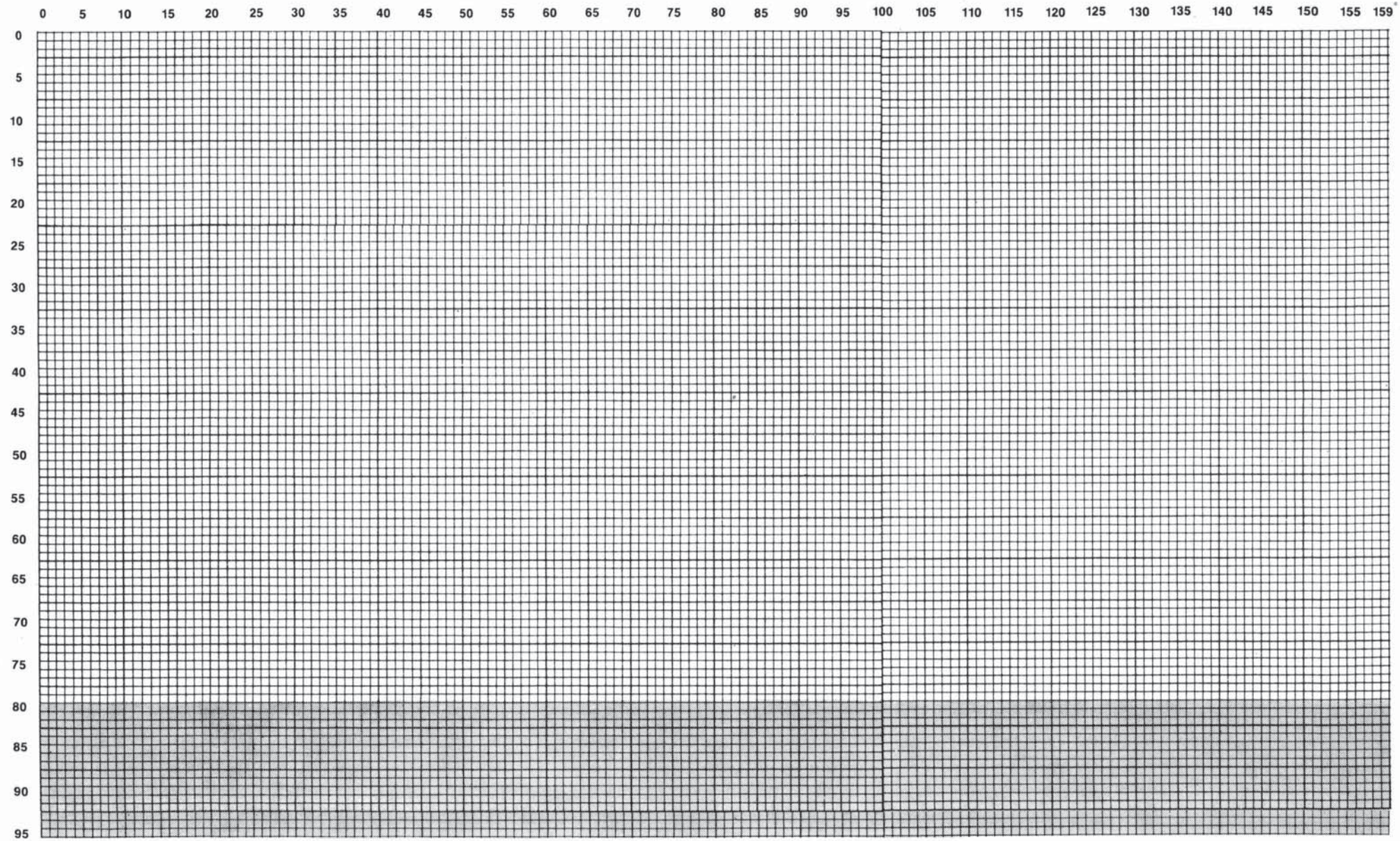
```
        JMP OUTLIN
OUTLB   LDA TRIES
        CLC
        ADC #1
        STA TRIES
        CMP #4
        BEQ OUTLD
        JSR INCD
        JMP OUTLA
OUTLD   JSR LOCTXY
OUTLD2  CMP COLOR2,X
        BNE OUTLE
        JSR FILL
        LDA #0
        STA FILLON
        RTS
OUTLE   JSR INCD
        JSR SRCHLC
        JMP OUTLD2
SRCHLC  LDX TD
        LDA SX
        CLC
        ADC SXD,X
        STA TX
        STA PLOTX
        LDA SY
        CLC
        ADC SYD,X
        STA TY
        STA PLOTY
LOCTXY  LDA PLOTX
        CMP #159
        BCS NOREAD
        LDA PLOTY
        CMP #85
        BCS NOREAD
        JSR PLOTCL
        LDY #0
        LDA (LO),Y
        AND BITSON,X
        RTS
NOREAD  LDA #0
        LDX #0
        RTS
GRABEM  LDA TD
        STA D
        LDA TX
        STA SX
        LDA TY
        STA SY
        RTS
INCD    LDA TD
        CLC
        ADC #1
        AND #3
        STA TD
        RTS
DECD    LDA TD
        SEC
        SBC #1
        AND #3
        STA TD
        RTS
PLSXSY  LDA SX
        STA PLOTX
        CMP MAXX
        BCC TMINX2
        STA MAXX
        JMP CKYMM2
TMINX2  CMP MINX
        BCS CKYMM2
        STA MINX
CKYMM2  LDA SY
        STA PLOTY
        CMP MAXY
        BCC TMINY2
        STA MAXY
        JMP ENDMM2
TMINY2  CMP MINY
        BCS ENDMM2
        STA MINY
ENDMM2  JSR PLOTCL
        LDY #0
        LDA BITOFF,X
        AND (LO),Y
        ORA COLOR2,X
        STA (LO),Y
        RTS
;
;FILL ROUTINE
;
;AS WITH THE 'SEARCH' SUBROUTINE, THE
;FILL SUBROUTINE IS FAR TOO COMPLEX TO
;EXPLAIN HERE.  THIS FILL IS ENTIRELY
;DIFFERENT FROM THE SYSTEM'S FILL
;ROUTINE, AS IT WILL FILL ANY SHAPE
;THAT IS OUTLINED IN COLOR 2.
;
FILL    LDA #0                  ;TURN OFF
        STA AUDC2               ;SOUND CHANNELS
        STA AUDC3               ;2 AND 3.
        LDA MAXY                ;INITIALIZE
        SEC                     ;THE FILL
        SBC MINY                ;SOUND
        STA FILFRQ              ;FREQUENCY.
        LDA MINX
        SEC
        SBC #1
        STA MINX
        STA FX
        LDA MINY
        STA FY
        SEC
        SBC #1
        STA MINY
        LDA MAXX
        CLC
        ADC #1
        STA MAXX
        LDA MAXY
        CLC
        ADC #1
        STA MAXY
        LDA #0
        STA SCTALY
CLRC2T  LDA #0
        STA C2TALY
LOCLP1  JSR LOCATE
        CMP #2
        BNE LOCLP1
LOCLP2  INC C2TALY
        JSR LOCATE
        CMP #2
        BEQ LOCLP2
        AND #1
        BNE CLRC2T
        LDA C2TALY
        CMP #1
        BEQ FILLIT
        JSR LOCPRV
        BEQ CLRC2T
FILLIT  LDA FX
        STA PLOTX
        LDA FY
        STA PLOTY
        JSR PLOTCL
        LDY #0
        LDA (LO),Y
        ORA COLOR3,X
        STA (LO),Y
        INC SCTALY
        JSR LOCATE
        CMP #0
        BEQ FILLIT
```

```
        AND #1
        BNE CLRC2T
        LDA #1
        STA C2TALY
FOLLOW  JSR LOCATE
        CMP #0
        BEQ LOCLP3
        AND #1
        BNE CLRC2T
        INC C2TALY
        JMP FOLLOW
LOCLP3  LDA C2TALY
        CMP #1
        BNE LOCLP4
        JMP CLRC2T
LOCLP4  JSR LOCPRV
        CMP BITSON,X
        BEQ FILLIT
        JMP CLRC2T
LOCATE  LDA FX
        CLC
        ADC #1
        STA FX
        CMP MAXX
        BNE STOFX
        LDA CURLO
        CLC
        ADC SCTALY
        STA CURLO
        LDA CURHI
        ADC #0
        STA CURHI
        LDA #0
        STA SCTALY
        LDA MINX
        STA FX
        LDA #0
        STA C2TALY
        LDA #$86
        STA AUDC1
        LDA FILFRQ
        STA AUDF1
        BEQ NOFFDC
        DEC FILFRQ
NOFFDC  LDA FY
        CLC
        ADC #1
        STA FY
        CMP MAXY
        BEQ FILEND
        LDA FX
        CMP MINX
        BNE STOFX
        PLA
        PLA
        JMP CLRC2T
FILEND  PLA
        PLA
        RTS
STOFX   LDA FX
        STA PLOTX
        LDA FY
        STA PLOTY
        JSR PLOTCL
        LDY #0
        LDA BITSON,X
        AND (LO),Y
        CMP COLOR2,X
        BNE NOTC2
        LDA BITSON,X
        ORA (LO),Y
        STA (LO),Y
        INC SCTALY
        LDA #2
        RTS
NOTC2   CMP COLOR1,X
        BNE NOTC1
        LDA #1
        RTS
NOTC1   CMP #0
        BNE C3
        RTS
C3      LDA #3
        RTS
LOCPRV  LDA FX
        STA PLOTX
        LDA FY
        SEC
        SBC #1
        CMP MINY
        BEQ NOLOCP
        STA PLOTY
        JSR PLOTCL
        LDY #0
        LDA BITSON,X
        AND (LO),Y
        RTS
NOLOCP  LDA #0
        LDX #0
        RTS
;
;VBI ROUTINE
;
INTRPT  LDA KEY             ;IS SPACE BAR
        CMP #$21            ;PRESSED?
        BNE NOPRES          ;NO, CHECK FOR PAUSE.
        LDA #$FF            ;CLEAR OUT
        STA KEY             ;KEY CODE,
        LDA PAUSE           ;COMPLEMENT
        EOR #$FF            ;THE PAUSE
        STA PAUSE           ;FLAG.
NOPRES  LDA PAUSE           ;ARE WE PAUSED?
        BEQ NOPAUS          ;NO!
        JMP XITVBV          ;PAUSED, NO VBI!
NOPAUS  LDA BSCNT           ;MORE BUMP SOUND?
        BMI NOBS            ;NO, PROCESS TIMER.
        ORA #$A0            ;MIX VOLUME WITH
        STA AUDC4           ;PURE TONE,
        LDA #$80            ;SET UP BUMP
        STA AUDF4           ;SOUND FREQUENCY
        DEC BSCNT           ;AND DECREMENT COUNT.
NOBS    LDA TIMER           ;TIMER DOWN TO ZERO?
        BEQ NODEC           ;YES, DON'T DECREMENT.
        DEC TIMER           ;DECREMENT TIMER.
NODEC   LDA SHTIM2
        BEQ NODEC2
        DEC SHTIM1
        BNE NODEC2
        DEC SHTIM2
NODEC2  LDA FILLON          ;ARE WE FILLING?
        BEQ NOFILL          ;NO, DO REST OF VBI.
        JMP XITVBV          ;YES, EXIT VBI
NOFILL  LDA #0              ;CLEAR OUT
        STA DEADFG          ;DEAD FLAG
        STA HSHORT          ;AND SHORT HIT.
        LDA P1PL            ;HAS PLAYER 1
        AND #$0C            ;HIT.PLAYER 2/3?
        BEQ NOHITS          ;NO, IT'S OK
        INC HSHORT          ;YES!!!
NOHITS  LDA P1PL            ;HAS PLAYER 1
        AND #$01            ;HIT PLAYER 0?
        BEQ NOHITP          ;NO!
        INC DEADFG          ;YES!!!
NOHITP  LDA P0PF            ;HAS PLAYER 0
        AND #$02            ;HIT COLOR 2?
        BEQ NOHITL          ;NO!
        INC DEADFG          ;YES!!!
NOHITL  STA HITCLR          ;CLEAR COLLISION.
        LDA MOVTIM          ;MOVEMENT TIMER ZERO?
        BEQ NOMDEC          ;YES, DON'T DECREMENT.
        DEC MOVTIM          ;DECREMENT TIMER.
NOMDEC  LDA SMTIM           ;STAR MOVE TIMER ZERO?
        BEQ NMTDEC          ;YES, DON'T DECREMENT.
        DEC SMTIM           ;DECREMENT TIMER.
```

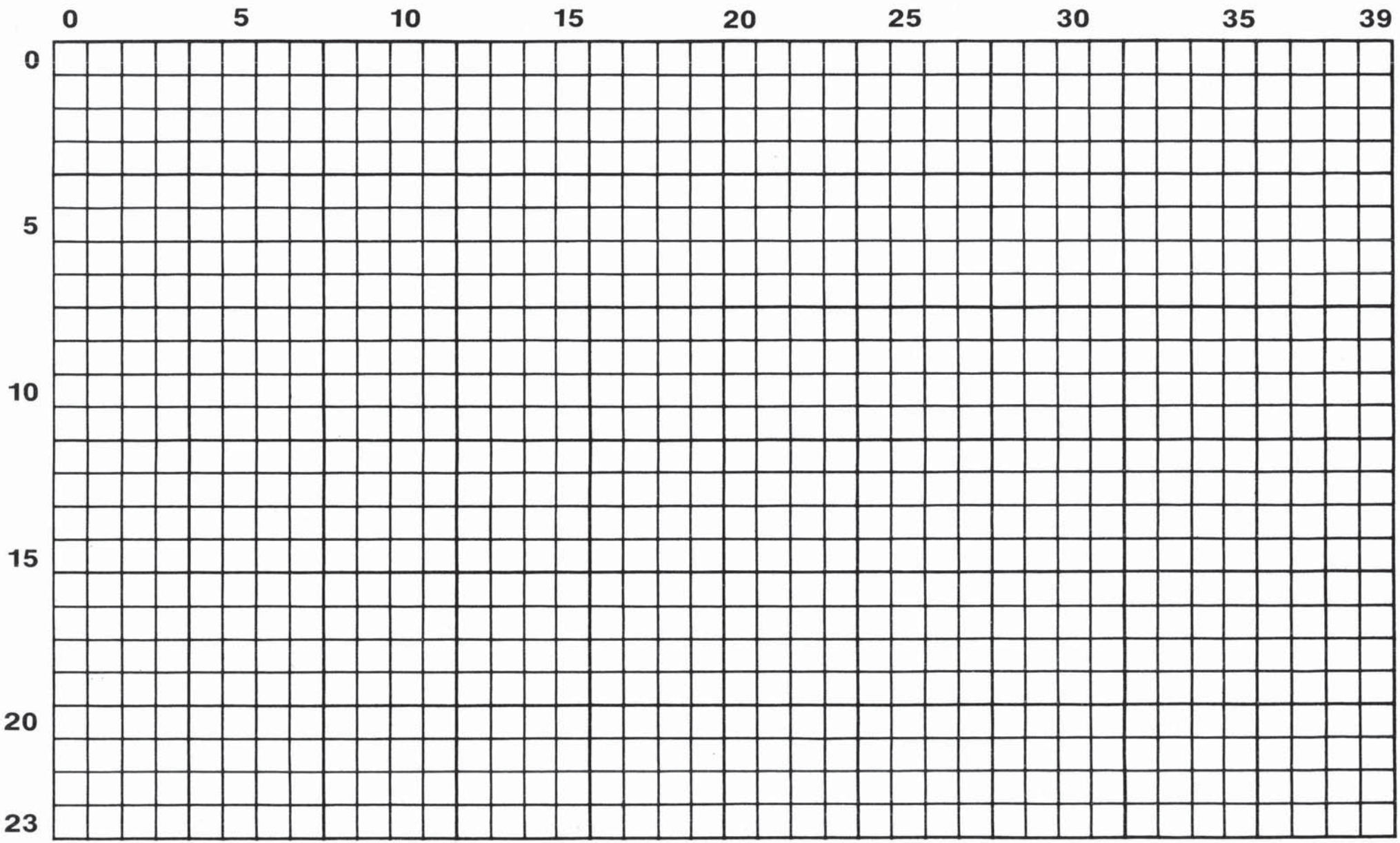
```
NMTDEC LDA STARCT          ;STAR ROT. TIMER ZERO?
       BEQ STAROT          ;YES, ROTATE STAR!
       DEC STARCT          ;DECREMENT TIMER
       JMP VBREST          ;AND SKIP ROTATION.
STAROT LDA #1              ;SET ROT. TIMER
       STA STARCT          ;TO 1
       LDA STRPOS          ;INCREMENT
       CLC                 ;STAR ROTATION
       ADC #1              ;COUNTER,
       CMP #7              ;ALLOW ONLY 0-6.
       BNE STOSTP          ;ROT. COUNT OK
       LDA #0              ;ZERO ROT. COUNTER.
STOSTP STA STRPOS          ;SAVE ROT. POS.
VBREST LDY STRPOS          ;THIS SECTION
       LDX STRHGT          ;DRAWS THE STAR
       LDA #0              ;IN PLAYER 0
       STA PL0-1,X         ;MEMORY USING
       STA PL0+8,X         ;THE TABLES
       LDA STARB1,Y        ;'STARB1' THRU
       STA PL0,X           ;'STARB8'.
       LDA STARB2,Y
       STA PL0+1,X
       LDA STARB3,Y
       STA PL0+2,X
       LDA STARB4,Y
       STA PL0+3,X
       LDA STARB5,Y
       STA PL0+4,X
       LDA STARB6,Y
       STA PL0+5,X
       LDA STARB7,Y
       STA PL0+6,X
       LDA STARB8,Y
       STA PL0+7,X
       LDA STRHOR          ;SET STAR'S
       STA HPOSP0          ;HORIZ. POS.
       LDA SHOOFF          ;OK TO SHOW PLAYER?
       BNE ENDVBI          ;NO, EXIT VBI
       LDA PX              ;SET PLAYER'S
       CLC                 ;HORIZONTAL
       ADC #47             ;POSITION
       STA HPOSP1
       LDA PY              ;DRAW PLAYER
       CLC                 ;IN PLAYER 3
       ADC #$10            ;MEMORY
       TAX
       LDA #0
       STA PL1-3,X
       STA PL1-2,X
       STA PL1+2,X
       STA PL1+3,X
       LDA #$40
       STA PL1-1,X
       STA PL1+1,X
       LDA #$A0
       STA PL1,X
       LDA NOCCHG          ;COLOR CHANGE OK?
       BNE ENDVBI          ;NO, EXIT VBI
       INC COLPM1          ;YES, CYCLE THE COLOR.

;      THE FOLLOWING ROUTINE USES TWO
;      PLAYERS (2 & 3) TO DISPLAY THE
;      "SHORTS."  EACH PLAYER IS USED
;      TO SHOW 2 SHORTS, SO SOME
;      FLICKER MAY BE OBSERVED.

ENDVBI INC SHFLIP          ;TOGGLE FLIP
       LDA SHFLIP          ;MASK FLIP
       LSR A               ;TO EITHER
       AND #1              ;0 OR 1
       TAY                 ;PUT IN Y
       LDA CPYSTN,Y        ;AND GET IMAGE
       STA CPYST           ;TO USE (+/X)
       LDA SHFLIP          ;GET FLIP.
       AND #1              ;MASK AND
       TAY                 ;PUT IN Y
       LDA #PL3/256        ;PUT PLAYER 3
       STA DESTHI          ;IN DESTINATION
       LDA #PL3&255        ;ADDRESS
       STA DESTLO          ;HI & LO
       LDA #1              ;SET DEST #
       STA DESTNM
       LDA SHSTRT,Y        ;GET START
       STA VBXHLD          ;SHORT #
SHORLP LDA #0
       LDX DESTNM
       LDY SHYHLD,X        ;GET LAST INDEX
       LDX #4              ;NOW ERASE
ERSSHO STA (DESTLO),Y      ;PREVIOUS
       INY                 ;SHORT
       DEX
       BPL ERSSHO
       LDX VBXHLD
       LDA SHORTF,X        ;SHORT ALIVE?
       BEQ NXTSHO          ;NO!
       LDA SHORTX,X        ;GET SHORT'S
       LDY SHORTY,X        ;COORDINATES,
       CLC
       ADC #46
       LDX DESTNM
       STA HPOSP2,X        ;SET HORIZ. POS.
       TYA
       CLC
       ADC #14
       STA SHYHLD,X        ;AND VERTICAL POS.
       TAY
       LDX CPYST
       LDA #4              ;NOW COPY
       STA CPYCNT          ;SHORT IMAGE
SHOCOP LDA SHOIMG,X        ;TO THE
       STA (DESTLO),Y      ;PLAYER
       INY
       DEX
       DEC CPYCNT
       BPL SHOCOP
NXTSHO DEC DESTNM          ;MORE?
       BMI VBEND           ;NO, EXIT!
       LDA DESTLO          ;POINT TO
       SEC                 ;NEXT PLAYER
       SBC #128
       STA DESTLO
       LDA DESTHI
       SBC #0
       STA DESTHI
       INC VBXHLD
       JMP SHORLP          ;AND LOOP BACK.

VBEND  JMP XITVBV          ;DONE WITH VBI!
;
;DATA
;
DLIST  DB  $70,$70,$70
       DB  $40,DISP&255,DISP/256
       DB  $D,$D,$D,$D,$D,$D,$D,$D,$D,$D
       DB  $D,$D,$D,$D,$D,$D,$D,$D,$D,$D
       DB  $D,$D,$D,$D,$D,$D,$D,$D,$D,$D
       DB  $D,$D,$D,$D,$D,$D,$D,$D,$D,$D
       DB  $D,$D,$D,$D,$D,$D,$D,$D,$D,$D
       DB  $D,$D,$D,$D,$D,$D,$D,$D,$D,$D
       DB  $D,$D,$D,$D,$D,$D,$D,$D,$D,$D
       DB  $D,$D,$D,$D,$D,$D,$D,$D,$D,$D
       DB  $D,$D,$D,$D,$D,$20,$46
SCDL   DW  SCOLIN
       DB  $46
       DW  SCOLN2
       DB  $41,DLIST&255,DLIST/256

TITLDL DB  $70,$70,$70,$70,$70,$70
       DB  $70,$70,$70,$30,$46
       DW  MAGMSG
       DB  $70,$47
       DW  TITLE
       DB  $30,$46
       DW  AUTHOR
```

```
        DB  $70,$70,$30,$46
        DW  STMSG
        DB  $41
        DW  TITLDL

SCOLIN  DB  CT,CG,CT,CCOL,0,0,0,0,0,0
        DB  0,CC,CU,CR,CCOL,0,0,0,0,0
SCOLN2  DB  CL,CV,CCOL,0,0,0,CS,CC,CO
        DB  CR,CE,CCOL,0,0,0,0,0,0,0,0
GOMSG   DB  0,0,0,0,0,CG,CA,CM,CE,0
        DB  0,CO,CV,CE,CR,0,0,0,0,0
MAGMSG  DB  0,0,$21,$2E,$21,$2C,$2F,$27
        DB  0,$23,$2F,$2D,$30,$35
        DB  $34,$29,$2E,$27,0,0
TITLE   DB  0,0,0,$66,$69,$6C,$6C,$47
        DB  $65,$72,0,$75,$70,$41,0
        DB  $69,$69,0,0,0
AUTHOR  DB  0,0,0,$A2,$B9,0,0,$B4
        DB  $AF,$AD,0,$A8,$B5,$A4,$B3,$AF
        DB  $AE,0,0,0
STMSG   DB  0,0,0,0,$F0,$FSTMSG  DB  0,0,0,0,$F0,$F2,$E5,$F3
        DB  $F3,0,0,$F3,$F4,$E1,$F2,$F4
        DB  0,0,0,0

;
;LEVEL TABLES
;
TGTLO   DB  64,16,224,40,248,212,16,4
        DB  248,224,212,224,68,168,112,212
TGTHI   DB  31,39,46,35,42,48,39,41,42
        DB  46,48,46,47,47,48,48
STARSP  DB  4,4,4,3,3,3,2,2,2,2,2,1,1
        DB  1,1,1
KILLFG  DB  0,0,1,0,1,1,0,1,1,1,1,0,0
        DB  1,1,1
SHTIME  DB  7,7,7,6,6,6,5,5,5,4,4,4,3
        DB  3,2,1
SHSTRT  DB  0,2
SHYHLD  DS  2
SHOIMG  DB  $88,$50,$20,$50,$88
        DB  $20,$20,$F8,$20,$20
CPYSTN  DB  4,9
DADD    DB  $FF,2,1
DIRX    DB  0,1,0,$FF
DIRY    DB  $FF,0,1,0
SHORTX  DS  4
SHORTY  DS  4
SHORTF  DS  4
SHORTD  DS  4

ZERO1   DB  0
SCORE   DB  0,0,0,0,0,0
SLLOC   DB  0
CURLO   DB  0
CURHI   DB  0
LEVEL   DB  0
PAUSE   DB  0
HASDRN  DB  0
LOWK    DB  0
HIWK    DB  0
SCTALY  DB  0
LIVES   DB  0
;
;STAR PLAYER-MISSILE IMAGES
;
STARB1  DB  $81,$40,$20,$10,$08,$04,$02
STARB2  DB  $42,$43,$20,$10,$08,$04,$C2
STARB3  DB  $24,$24,$13,$10,$08,$C8,$24
STARB4  DB  $18,$18,$1C,$1F,$F8,$38,$18
STARB5  DB  $18,$18,$38,$F8,$1F,$1C,$18
STARB6  DB  $24,$24,$C8,$08,$10,$13,$24
STARB7  DB  $42,$C2,$04,$08,$10,$20,$43
STARB8  DB  $81,$02,$04,$08,$10,$20,$40
STARCT  DB  0
STRPOS  DB  0
STRHGT  DB  0
STRHOR  DB  0
STRLX   DB  0
STRLY   DB  0
TMPDIR  DB  0
STRDIR  DB  0
STRDTX  DB  1,1,0,255,255,255,0,1
STRDTY  DB  0,1,1,0,1,255,255,255
STRSPD  DB  4
COLOR1  DB  $40,$10,$04,$01
COLOR2  DB  $80,$20,$08,$02
COLOR3  DB  $C0,$30,$0C,$03
BITSON  DB  $C0,$30,$0C,$03
BITOFF  DB  $3F,$CF,$F3,$FC
BXSTRT  DB  0,158,158,0
BYSTRT  DB  0,0,84,84
BXINC   DB  1,0,255,0
BYINC   DB  0,1,0,255
BORCNT  DB  159,85,159,85
BORNUM  DB  0
BDINCX  DB  0
BDINCY  DB  0
BDCNT   DB  0
PXWC    DB  0
PYWC    DB  0
SHOOFF  DB  0
CKX     DB  0
CKY     DB  0
INIX    DB  0
INIY    DB  0
MINX    DB  0
MINY    DB  0
MAXX    DB  0
MAXY    DB  0
REX     DB  0
REY     DB  0
X       DB  0
Y       DB  0
SX      DB  0
SY      DB  0
TX      DB  0
TY      DB  0
FX      DB  0
FY      DB  0
TD      DB  0
D       DB  0
BSCNT   DB  0
FILFRQ  DB  0
TRIES   DB  0
FILLON  DB  0
C2TALY  DB  0
NOCCHG  DB  0
DEDBRT  DB  0
STKHLD  DB  0
RORCOL  DB  0
REDIR   DB  0
LGTHY   DB  0
TIMES   DB  0
CKV     DB  0
CKVX    DB  0
DRAWFG  DB  0
MOVIX   DB  0
XD      DB  0,0,0,0
        DB  0,0,0,1
        DB  0,0,0,255
        DB  0,0,0,0
YD      DB  0,0,0,0
        DB  0,0,0,0
        DB  0,0,0,0
        DB  0,1,255,0
SXD     DB  0,1,0,255
SYD     DB  255,0,1,0
DECIMAL     DB                  0,0,0,0,0
ZERO2   DB  0
HIVALS  DB  0,0,0,3,39
LOVALS  DB  1,10,100,232,16
DIR     DS  256
LGTH    DS  256

        END $6000
```

Graphics Mode 6 (2 COLORS-2025 BYTES) & Mode 7 (4 COLORS-3945 BYTES)

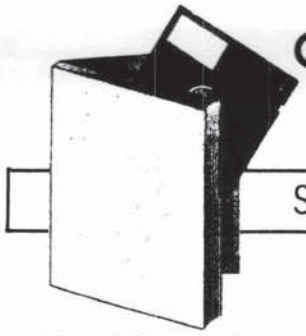


Graphics Mode 3 (4 COLORS-273 BYTES) & Mode 0 (1 COLOR - 993 BYTES / 2 LUMINANCE - TEXT)



NOTE: Graphics Mode O is a text mode full-screen display only.

Notes



Get All A.N.A.L.O.G. Compendium Programs on Disk!

Special Offer for Compendium Owners Only.

Only $35.00 per set (3 Disks)

Yes! Send ____ Disk set(s) which contains all the programs in the **A.N.A.L.O.G. Compendium** to:

Name

Address

City State Zip

☐ MasterCard ☐ Payment Enclosed ☐ VISA
Account No. _____ Expires ____/____

Foreign orders add $5.00 for Air Mail Postage.

THE MAGAZINE FOR ATARI COMPUTER OWNERS

Yes, Please enter my subscription to:

THE MAGAZINE FOR ATARI COMPUTER OWNERS
ANALOG COMPUTING

For Fastest Service,
Call Our **Toll-Free**
US Order Line
800-345-8112
In Pennsylvania call 800-662-2444

☐ 1 year .....(12 issues) $28
☐ 2 years .....(24 issues) $52
☐ 3 years .....(36 issues) $79

Additional Postage per year:
Canada & Mexico $8 Surface, $12 Air
Other Foreign $12 Surface, $48 Air

Name _____

Address _____

City _____ State _____ Zip _____

☐ Check #_____ ☐ M.O. #_____ ☐ VISA ☐ MC ☐ Bill me

Card # _____ Exp. Date ____/____/____

Your subscription will begin with the first available issue. Please allow 4-6 weeks for delivery of first issue.

SUBSCRIBE TO

ANALOG COMPUTING ON TAPE OR DISK

Includes magazine with tape or disk

Cassette .....☐ ½ year .....(6 issues) $48
☐ Full year .....(12 issues) $90
Disk .....☐ ½ year .....(6 issues) $72
☐ Full year .....(12 issues) $130

Name _____

Address _____

City _____ State _____ Zip _____

☐ Check #_____ ☐ M.O. #_____ ☐ VISA ☐ MC ☐ Bill me

Card # _____ Exp. Date ____/____/____

FOREIGN ORDERS ADD ADDITIONAL $20 TO EACH 6-ISSUE SUBSCRIPTION.

PUT STAMP HERE
THE POST OFFICE
WILL NOT DELIVER
MAIL WITHOUT
POSTAGE

THE MAGAZINE FOR ATARI® COMPUTER OWNERS™

ANALOG COMPUTING

P.O. Box 23

Worcester, MA 01603

PUT STAMP HERE
THE POST OFFICE
WILL NOT DELIVER
MAIL WITHOUT
POSTAGE

THE MAGAZINE FOR ATARI® COMPUTER OWNERS™

ANALOG COMPUTING

P.O. Box 615

Holmes, PA 19043

PUT STAMP HERE
THE POST OFFICE
WILL NOT DELIVER
MAIL WITHOUT
POSTAGE

THE MAGAZINE FOR ATARI® COMPUTER OWNERS™

ANALOG COMPUTING

P.O. Box 615

Holmes, PA 19043

THE A.N.A.L.O.G. COMPENDIUM

. . . is not just another "1001 programs" book. The **A.N.A.L.O.G. Compendium** contains the best of the first ten issues of **A.N.A.L.O.G. Computing**, the first magazine devoted exclusively to Atari home computer users.

This volume is a must for anyone with an Atari computer. Not only does it contain games that are equal in quality to many commercially available, but it is packed with tutorials, business, educational, and useful home applications as well.

The **A.N.A.L.O.G. Compendium** is one of the best single sources of high quality Atari programs you can buy. In fact, you may find that just one program could be worth the price of the entire book!

ISBN 0-914177-00-1

$14.95