A 6502 Version Of The Winter Consumer Electronics Show

The 6502 Resource Magazine
PET • Apple • Atari • OSI • KIM • SYM • AIM

Clearing Apple II's Low-Resolution Graphics Screen

COMPUTE!

$2.50
March, 1981
Issue 10
Vol. 3, No. 3
63379

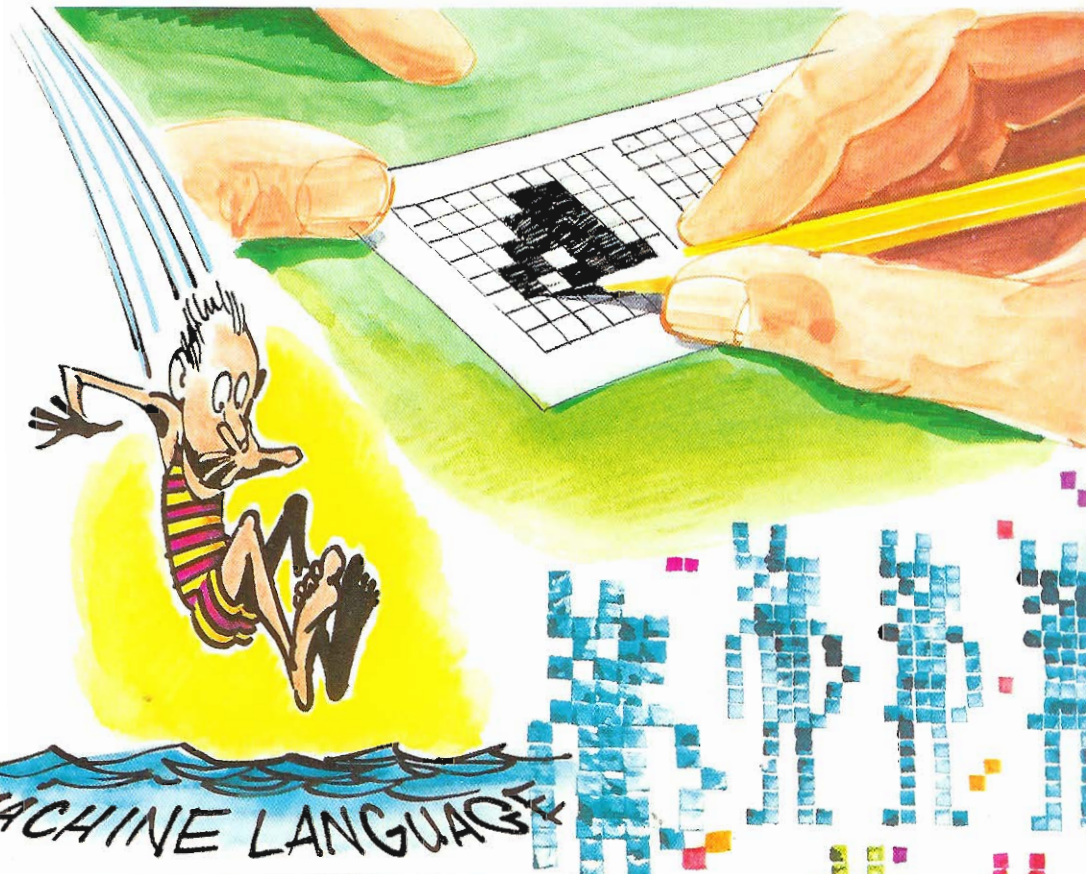
The Journal For Progressive Computing™

Designing Your Own Atari Character Set

Machine Language Taking The Plunge

Keyprint (For The Pet) Revisited

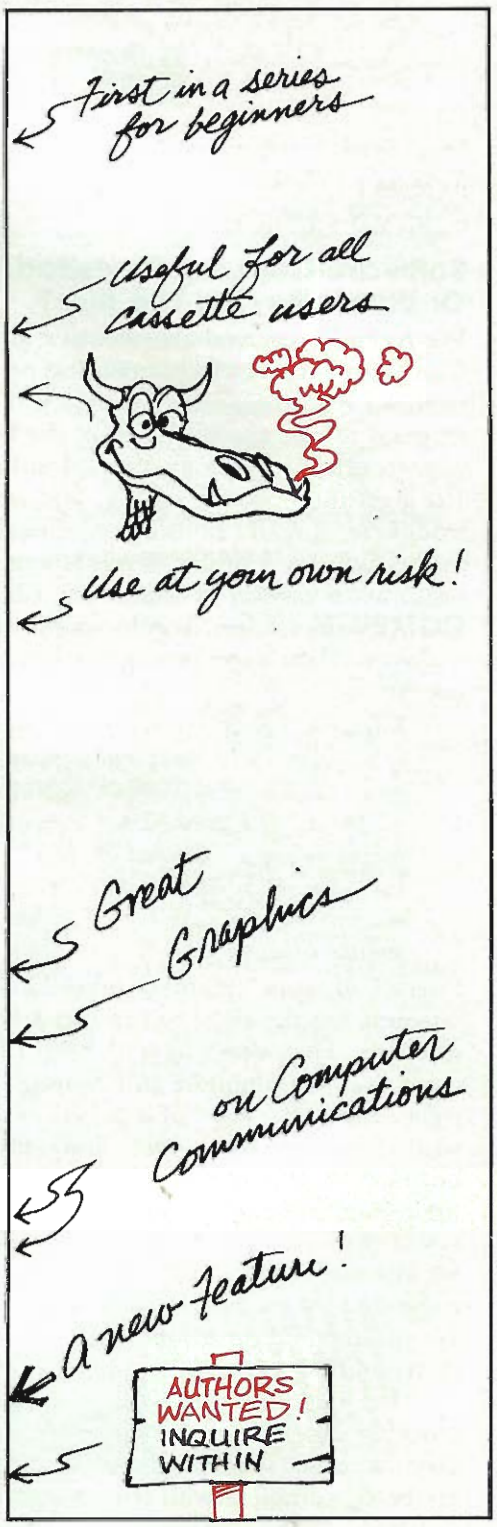
Six-Gun Shootout Game For The OSI C1P



0 74470 63379 03

Table of Contents

March, 1981, Vol. 3. No. 3



COMPUTE! The Journal for Progressive Computing (USPS: 537250) is published 12 times each year by Small System Services, Inc., P.O. Box 5406, Greensboro, NC 27403 USA. Phone: (919) 275-9809. Editorial Offices are located at 200 East Bessemer Ave., Greensboro, NC 27401.

Domestic Subscriptions: 12 issues, $20.00. Send subscription orders or change of address (P.O. Form 3579) to Circulation Dept., **COMPUTE!** Magazine, P.O. Box 5406, Greensboro, NC 27403. Controlled circulation postage paid at Greensboro, NC 27403. Application to mail at controlled circulation rates pending at Hickory, NC 28601.

ISSN 0194-357X.

A Small System Services, Inc. Publication

COMPUTE!

ATARI Memory Dump and Dissassembler

Robert W. Baker
Atoo, New Jersey

Here's a handy little utility program for the Atari 400/800 systems. It lets you examine any area of memory, either RAM or ROM, in one of two formats. You can select whether you want a straight memory dump or a dissassembly listing. In both formats, the memory locations are given as both decimal and hexadecimal values. The data can be displayed on the television/monitor screen or printed on a printer if available.

When first run, the program takes a minute or two to initialize but from then on it is relatively fast. The starting address for the dump/dissassembly can be entered as either a decimal or hexadecimal number. When entering it as a hexadecimal number, precede the number by a dollar sign ($). You're then asked if a dissassembly is desired. Answering N for no will cause the standard memory dump to be displayed. Answering Y for yes will generate the dissassembly listing.

Before the dump/dissassembly is displayed you are given the option to have the output printed if desired. Answering N for no causes the output to be displayed as normal, using the entire display (24 lines). At the end of each screen you are given the option to continue (C), restart (R), or stop (S). Continue will display the next screen in sequential order. Restarting will return to select the starting address and allow specifying dump/dissassembly and printer options. When printing the data output, the printer will print continuously. Just press any key on the Atari keyboard to halt the printer. When the printer stops you will see the prompt for continue, restart, or stop as mentioned above.

Memory Dump

The memory dump simply displays the contents of eight bytes of memory on each line displayed or printed. The values are given in hexadecimal to conserve display space and to correspond with the dissassembly listings. This feature is very useful for examining pointers or various values stored in memory, that do not happen to be executable machine code instructions. You might want to play around with looking at how BASIC variables or even BASIC lines themselves are stored in memory on the Atari.

For those with 80 column printers (Atari 825, etc.) you can change the FOR-NEXT loop count in line 600 to get 16 bytes per line to conserve paper. Just change line 600 to:

```
FOR X = 1 TO 16:V = PEEK(A)
```

You might even want to change the heading line in line number 302 to print the numbers 0 to 9 plus A to F.

Dissassembly Listing

This feature is much more powerful and can provide a wealth of information. When a dissassembly is requested, the program displays one 6502 machine instruction per line. It indicates the hexadecimal value of one to three bytes of memory that make up the instruction. It also displays the instruction and operand in the standard assembly code forms.

Any unrecognized values are indicated by a "*?*" instead of an instruction mnemonic. You may have to try different starting locations to get the dissassembly to function properly. If you specify an address that happens to be the middle of an instruction, it may dissassemble as a different instruction and/or cause following instructions to be displayed incorrectly. This is always a problem with a dissassembly program of this kind. It is extremely difficult to "sync-up" with the machine instructions whenever there are data bytes between various routines, etc. If you should see a high number of *?*'s displayed, try another starting address, possibly one to two higher or lower than the original address. This should correct the situation.

The dissassembly gives each instruction using standard MOS Technology mnemonics and addressing nomenclatures. Operand values and addresses are shown in hexadecimal and are prefixed by a dollar sign ($) as a reminder. All immediate values are preceded by parenthesis and indexed values are suffixed by a ",X" or ",Y" as appropriate. Zero page addressing is implied by the length of the operand being only a single byte. All branch instructions show the actual computed target address in the dissassembly for added convenience. If required, the relative offset is shown in the object code.

The dissassembly function is extremely useful for examining machine language routines such as those used within the BASIC cartridge itself, or in the operating system ROMs. I'll let you know if I come across anything interesting hidden in the Atari system. Before anyone asks — if you'd like a copy on cassette tape instead of doing all the typing, send $2 to cover costs.

Just a quick note concerning the program listings. The heading lines printed by the BASIC statements in lines 302 and 305 were actually in reverse image to enhance the display. Unfortunately this does not show up in the program listings. I have

tried to use CHR$(xx) functions in the print statements for clearing the display, etc. to make things easier to read. The Atari printers do not print the graphics and/or control characters that can be included in PRINT statements. Actually they can cause problems if a program is LISTed with these control characters imbedded in PRINT statements. The control characters will be decoded and acted upon by the printer.

```
10 REM ********************************
20 REM
25 REM MEMORY DUMP/DISASSEMBLER
30 REM
35 REM BY: ROBERT W. BAKER
40 REM 15 WINDSOR DR, ATCO NJ 08004
50 REM
60 REM ***** V1.0 ******* 1/4/81 ****
65 REM
70 GRAPHICS 0:POKE 752,1
80 PRINT CHR$(125);"        M E M O R Y
  D U M P   ":? :?
90 PRINT "INITIALIZING ....."
100 DIM H$(16),A$(6),S$(6),M$(1536)
110 H$="0123456789ABCDEF"
120 S$="      "
150 OPEN #1,4,0,"K"
160 FOR X=1 TO 1531 STEP 6
170 READ A$
175 IF A$(2,2)="*" THEN A$(2,4)="*?*"
180 N=LEN(A$):IF N<6 THEN A$(N+1)=S$
185 M$(X,X+5)=A$:NEXT X
200 PRINT CHR$(125);"        M E M O R Y
   D U M P   ":? :?
201 PRINT "ENTER DECIMAL STARTING ADDRES
S":PRINT
202 PRINT "OR HEX ADDRESS PRECEDED BY '$
'":PRINT
203 POKE 752,0
204 INPUT A$:IF A$="" THEN 800
205 IF A$(1,1)="$" THEN 209
206 FOR X=1 TO LEN(A$)
207 IF A$(X,X)<"0" OR A$(X,X)>"9" THEN 2
00
208 NEXT X:A=INT(VAL(A$)/8)*8:GOTO 240
209 A=0:IF LEN(A$)<2 THEN 200
210 FOR X=2 TO LEN(A$)
211 IF A$(X,X)<"0" THEN 200
212 IF A$(X,X)<="9" THEN A=A*16+VAL(A$(X
,X)):GOTO 220
215 IF A$(X,X)<"A" OR A$(X,X)>"F" THEN 2
00
218 A=A*16+ASC(A$(X,X))-55
220 NEXT X
240 PRINT :PRINT "WANT DISSASSEMBLY (Y/N
) ?";
242 GET #1,X:D=0:IF X=78 THEN 245
244 D=1:IF X<>89 THEN 240
245 PRINT CHR$(X)
250 PRINT :PRINT "WANT PRINTED COPY (Y/N
) ?";
252 CLOSE #2
255 P=0:GET #1,X
260 IF X=78 THEN OPEN #2,8,0,"E":GOTO 29
0
270 IF X<>89 THEN 255
280 P=1:OPEN #2,8,0,"P"
290 IF P=0 THEN PRINT CHR$(125);:GOTO 30
0
295 PRINT CHR$(125);"DEPRESS ANY KEY TO
HALT PRINTER":PRINT #2
300 PRINT #2;"LOC-DEC/HEX  ";
302 IF D=0 THEN PRINT #2;"0  1  2  3  4
 5  6  7  ":GOTO 310
305 PRINT #2;" OBJECT   DISSASSEMBLY  "
310 PRINT #2
320 POKE 764,255
330 IF P=0 THEN FOR N=1 TO 20
340 IF A>65535 THEN A=A-65536
350 A$=STR$(A):L=LEN(A$)
360 PRINT #2;S$(1,6-L);A$;" ";
370 Y=A:GOSUB 950
380 PRINT #2;": ";
400 IF D=0 THEN 600
410 V=PEEK(A)
411 GOSUB 1000:PRINT #2;" ";
415 A=A+1:X=(6*V)+1:A$=M$(X,X+5)
420 IF A$(1,1)="0" THEN PRINT #2;"
 ";A$(2,4):GOTO 630
430 V=PEEK(A):GOSUB 1000
432 PRINT #2;" ";:A=A+1
435 IF A$(1,1)="2" THEN 500
440 PRINT #2;"    ";A$(2,4);" ";
445 IF A$(5,5)<>"R" THEN 470
450 IF V>127 THEN V=V-256
460 Y=A+V:GOSUB 900:GOTO 590
470 IF A$(5,5)="#" THEN PRINT #2;"#$";:G
OSUB 1000:GOTO 590
475 IF A$(6,6)=")" THEN PRINT #2;"(";
480 PRINT #2;"$";:GOSUB 1000
482 IF A$(5,5)=" " THEN 590
485 IF A$(5,6)="Y)" THEN PRINT #2;"),Y":
GOTO 630
490 PRINT #2;",";A$(5,6):GOTO 630
500 V1=V:V=PEEK(A):GOSUB 1000:A=A+1
510 PRINT #2;"  ";A$(2,4);" ";
515 Y=V1+(256*V)
520 IF A$(5,5)=")" THEN PRINT #2;"(";:GO
SUB 900:PRINT #2;")":GOTO 630
525 GOSUB 900
530 IF A$(5,5)=" " THEN 590
540 PRINT #2;",";A$(5,5):GOTO 630
590 PRINT #2:GOTO 630
600 FOR X=1 TO 8:V=PEEK(A)
610 GOSUB 1000:PRINT #2;" ";
620 A=A+1:NEXT X:PRINT #2
630 IF P=0 THEN NEXT N:GOTO 700
640 IF PEEK(764)=255 THEN 340
```

```
650 GET #1,X
700 POKE 752,1:PRINT
705 PRINT "CONTINUE, RESTART, OR STOP (C
,R,S) ?";
710 GET #1,X:IF X=67 THEN 290
730 IF X=82 THEN 200
740 IF X<>83 THEN 710
800 POKE 752,0:CLOSE #1:CLOSE #2:END
900 PRINT #2;"$";
950 V=INT(Y/256):GOSUB 1000
960 V=Y-(V%256)
1000 H=INT(V/16):L=V-(H%16)
1010 PRINT #2;H$(H+1,H+1);H$(L+1,L+1);
1020 RETURN
9000 DATA 0BRK,1ORA(X),0%,0%,0%,1ORA,1ASL
,0%
9010 DATA 0PHP,1ORA#,0ASL,0%,0%,2ORA,2AS
L,0%
9020 DATA 1BPLR,1ORA(Y),0%,0%,0%,1ORAX,1A
SLX,0%
9030 DATA 0CLC,2ORAY,0%,0%,0%,2ORAX,2ASL
X,0%
9040 DATA 2JSR,1ANDX),0%,0%,1BIT,1AND,1R
OL,0%
9050 DATA 0PLP,1AND#,0ROL,0%,2BIT,2AND,2
ROL,0%
9060 DATA 1BMIR,1ANDY),0%,0%,0%,1ANDX,1R
OLX,0%
9070 DATA 0SEC,2ANDY,0%,0%,0%,2ANDX,2ROL
X,0%
9080 DATA 0RTI,1EORX),0%,0%,0%,1EOR,1LSR
,0%
9090 DATA 0PHA,1EOR#,0LSR,0%,2JMP,2EOR,2
LSR,0%
9100 DATA 1BVCR,1EORY),0%,0%,0%,1EORX,1L
SRX,0%
9110 DATA 0CLI,2EORY,0%,0%,0%,2EORX,2LSR
X,0%
9120 DATA 0RTS,1ADCX),0%,0%,0%,1ADC,1ROR
,0%
9130 DATA 0PLA,1ADC#,0ROR,0%,2JMP),2ADC,
2ROR,0%
9140 DATA 1BVSR,1ADCY),0%,0%,0%,1ADCX,1R
ORX,0%
9150 DATA 0SEI,2ADCY,0%,0%,0%,2ADCX,2ROR
X,0%
9160 DATA 0%,1STAX),0%,0%,1STY,1STA,1STX
,0%
9170 DATA 0DEY,0%,0TXA,0%,2STY,2STA,2STX
,0%
9180 DATA 1BCCR,1STAY),0%,0%,1STYX,1STAX
,1STXY,0%
9190 DATA 0TYA,2STAY,0TXS,0%,0%,2STAX,0%
,0%
9200 DATA 1LDY#,1LDAX),1LDX#,0%,1LDY,1LD
A,1LDX,0%
9210 DATA 0TAY,1LDA#,0TAX,0%,2LDY,2LDA,2
LDX,0%
9220 DATA 1BCSR,1LDAY),0%,0%,1LDYX,1LDAX
,1LDXY,0%
9230 DATA 0CLV,2LDAY,0TSX,0%,2LDYX,2LDAX
,2LDXY,0%
9240 DATA 1CPY#,1CMPX),0%,0%,1CPY,1CMP,1
DEC,0%
9250 DATA 0INY,1CMP#,0DEX,0%,2CPY,2CMP,2
DEC,0%
9260 DATA 1BNER,1CMPY),0%,0%,0%,1CMPX,1D
ECX,0%
9270 DATA 0CLD,2CMPY,0%,0%,0%,2CMPX,2DEC
X,0%
9280 DATA 1CPX#,1SBCX),0%,0%,1CPX,1SBC,1
INC,0%
9290 DATA 0INX,1SBC#,0NOP,0%,2CPX,2SBC,2
INC,0%
9300 DATA 1BEQR,1SBCY),0%,0%,0%,1SBCX,1I
NCX,0%
9310 DATA 0SED,2SBCY,0%,0%,0%,2SBCX,2INC
X,0%
```

```
LOC-DEC/HEX  0  1  2  3  4  5  6  7
 40992 A020: 95 00 E8 94 00 E8 E0 92
 41000 A028: 90 F6 A2 86 A0 01 20 7F
 41008 A030: A8 A2 8C A0 03 20 7F A8
 41016 A038: A9 00 A8 91 84 91 8A C8
 41024 A040: A9 80 91 8A C8 A9 03 91
 41032 A048: 8A A9 0A 85 C9 20 F8 B8
 41040 A050: 20 41 BD 20 72 BD A5 92
 41048 A058: F0 03 20 99 BD 20 57 BD
 41056 A060: A5 CA D0 9C A2 FF 9A 20
 41064 A068: 51 DA A9 5D 85 C2 20 92
 41072 A070: BA 20 F4 A9 D0 EA A9 00
 41080 A078: 85 F2 85 9F 85 94 85 A6
 41088 A080: 85 B3 85 B0 85 B1 A5 84
 41096 A088: 85 AD A5 85 85 AE 20 A1
 41104 A090: D8 20 9F A1 20 C8 A2 A5
   ***** SAMPLE MEMORY DUMP *****
```

```
LOC-DEC/HEX   OBJECT    DISSASSEMBLY
 40992 A020: 95 00      STA $00,X
 40994 A022: E8         INX
 40995 A023: 94 00      STY $00,X
 40997 A025: E8         INX
 40998 A026: E0 92      CPX #$92
 41000 A028: 90 F6      BCC $A020
 41002 A02A: A2 86      LDX #$86
 41004 A02C: A0 01      LDY #$01
 41006 A02E: 20 7F A8   JSR $A87F
 41009 A031: A2 8C      LDX #$8C
 41011 A033: A0 03      LDY #$03
 41013 A035: 20 7F A8   JSR $A87F
 41016 A038: A9 00      LDA #$00
 41018 A03A: A8         TAY
 41019 A03B: 91 84      STA ($84),Y
 41021 A03D: 91 8A      STA ($8A),Y
 41023 A03F: C8         INY
 41024 A040: A9 80      LDA #$80
```

```
41026 A042: 91 8A     STA ($8A),Y
41028 A044: C8        INY
41029 A045: A9 03     LDA #$03
41031 A047: 91 8A     STA ($8A),Y
41033 A049: A9 0A     LDA #$0A
41035 A04B: 85 C9     STA $C9
41037 A04D: 20 F8 B8  JSR $B8F8
41040 A050: 20 41 BD  JSR $BD41
41043 A053: 20 72 BD  JSR $BD72
41046 A056: A5 92     LDA $92
41048 A058: F0 03     BEQ $A05D
41050 A05A: 20 99 BD  JSR $BD99
41053 A05D: 20 57 BD  JSR $BD57
41056 A060: A5 CA     LDA $CA
41058 A062: D0 9C     BNE $A000
41060 A064: A2 FF     LDX #$FF
41062 A066: 9A        TXS
```

***** SAMPLE DISSASSEMBLY ***** ©

Formatted Output for ATARI Basic

Joseph J. Wrobel

Many folks tell me that they must struggle to produce nicely formatted output when using ATARI Basic due to the lack of the TAB function and the PRINT USING command. Struggle no more. When used together, the two subroutines presented in this article can provide formatted output simply and directly in ATARI Basic. Both numerics and strings are supported. The type, arrangement and format of variables which appear on one output line are controlled on a line-by-line basis by the main program. The number of variables in one output is limited only by the character width of the output device. The output device can be the TV screen or any type of printer, ATARI or otherwise.

The approach to formatted output used here employs two subroutines. The purpose of the first is to construct a line for output in a string variable set aside for this purpose. Each time the subroutine is called, it inserts the data sent to it by the main program at the selected position in the string and in the format requested. When all the data to be printed in the current line has been positioned, the second subroutine is called. This subroutine merely prints the output line string on the appropriate device, then clears the string (fills it with spaces) to prepare it for the next line of data.

A sample program using the routines is given in Listing 1. Line 10 is required to set aside the strings to be used in the subroutines. LINE$ is the string which will ultimately contain the formatted output line. It is dimensioned to a size one less than the character width of the output device. In the example, for the 38 character wide default screen size, it is dimensioned to hold up to 37 characters by setting NC to 37. Line 20 initializes LINE$ to a string of spaces. N$ is a string used to temporarily store each data item. It should be big enough to hold the largest of your data items. To be on the safe side, its length is set equal to that of LINE$.

The actual subroutines of interest reside in lines 1000-1070 and lines 2000-2010 respectively. The latter routine, as noted above, simply prints LINE$ (line 2000) then fills it with spaces (line 2010) in preparation for the next output line. If, instead of the screen, a printer is the output device, then the PRINT of line 2000 may simply be replaced by an LPRINT or a PRINT # command, whichever is appropriate.

The routine starting at line 1000 actually does the formatting and has two different entry points depending on whether the data item is a string or a numeric. The case of a string is the simpler of the two, so let's consider it first. To position a string you must place the string in variable N$, specify the column position (RC) against which it is to be right justified, then GOSUB 1060. Three examples of this calling sequence are given in lines 100-120. At line 1060 the program first calculates LC, the leftmost character position of the data item. Then, if the column boundaries are acceptable, it inserts N$ in LINE$ and RETURNs.

To position a numeric, you must put the data item into variable N, specify ND, the number of digits to the right of the decimal point you wish printed, specify RC as defined above, then GOSUB 1000. See lines 150-170 for examples of this calling sequence. In lines 1000-1010 N is rounded to the appropriate number of decimals. In 1020 the string N$ is defined. If the number is to be printed as an integer (ND = 0), N$ is correct as is and the jump is taken to 1060 to insert N$ into LINE$. For non-integer formats, the decimal point and any trailing zeroes which were dropped in forming the string representation of the number must be restored. This is done in lines 1030-1050. N$ is then ready for insertion into LINE$.

Output from the sample program as printed on an ATARI 820 printer (using LPRINT in 2010) is shown in Figure 1. Note that the numbers are rounded for presentation with the requested precision, that the decimal points of each column neatly line up, that all trailing zeroes are present and that negative numbers are also accommodated. Also note how easy it is to line up the column headings with the data by simply specifying the appropriate value of RC when printing them (see lines 100 & 150, 110 & 160, 120 & 170).

The routines run fairly rapidly, but if you need some extra speed, the loop in line 2010 can be avoided. To do this, dimension a string, let's call it