# POCKET COMPUTER NEWSLETTER

*PC-2 and PC-1500 extracts from the Pocket Computer Newsletter*

# PART I

## Issues 13 to 28

## PROGRAM YIELDS PHASES OF THE MOON

Input any date (month/day/year) and this program outputs the phase of the moon on that date. If the moon is not at any exact quarter phase (such as full, new, first quarter and so forth) then the program indicates the number of days into the lunar cycle.

The program is accurate for dates between March 1, 1900 and February 28, 2100. Starting with a lunar time of 14.60 days on March 1, 1900, this program adds 29.53059167 days for each lunar period. Need to know if there was a full moon when you were born... married... first became a computer programmer?!

Program submitted by: *Emerich Auersbacher, 41 King Street, Apt. 2, Belleville, NJ 07109.*

Program *Moon Phases*

```
 10  CLEAR:BEEP 1:PAUSE "MOON PHASE CALCULATOR"
 20  W=694098,X=29.53059167,Z=365.25:USING
 30  BEEP 1:PAUSE "DATE IN QUESTION >> —> "
 40  INPUT "MONTH:";M,"DAY:";D,"YEAR:";Y:IF Y < 100
     LET Y=1900+Y
 50  IF M <= 2 LET S=((INT(30.6*(M+13))+INT(Z*(Y-1))+
     D-W)+14.6)/X:S=S-INT S:S=INT(SX+1):GOTO 70
 60  S=((INT(30.6*(M+1))+INT(Z*Y)+D-W)+14.6)/X:S=S-
     INT S:S=INT(SX+1)
 65  BEEP 1
 70  IF (S=0)+(S=1)+(S=29)+(S=30) > 0 PRINT "PHASE:
     FULL MOON":GOTO 110
 80  IF (S=14)+(S=15)+(S=16) > 0 PRINT "PHASE: NEW
     MOON":GOTO 110
 90  IF (S=6)+(S=7)+(S=8) > 0 PRINT "PHASE: LAST
     QUARTER":GOTO 110
100  IF (S=21)+(S=22)+(S=23) > 0 PRINT "PHASE: FIRST
     QUARTER"
110  PRINT "CYCLE IS ";S;"DAYS INTO 30"
120  GOTO 30
```

## PUTTING SOFTKEYS UNDER PROGRAM CONTROL

The softkeys on the new Sharp PC-1500 make selecting a particular program or routine the simple matter of pressing a single key. These keys can also be given labels that appear on the display. Using the label recall key a user can be reminded of the function of each key.

In normal use, the softkeys execute a programmed operation only when the PC is in the "executive" mode. That is, when the unit is not in the BUSY condition. Also, in order to see the softkey labels, it is necessary to first press the RCL key. Thus, while it is possible to construct a multi-part program and use the labeled softkeys as a menu selector, it is necessary to end each segment to return to the executive, have the user press the RCL key to bring up the softkeys menu and then punch the appropriate softkey.

However, putting the softkeys under program control makes it possible to display a menu and use the softkeys to direct program operation without ever returning to the executive mode. The accompanying listing illustrates how this can be accomplished.

Line 7000 presents a menu on the display.

Line 7010 uses the INKEY$ function to examine the keyboard.

Line 7020 extracts the ASCII code for the key that has been pressed. It ignores all key codes outside the range (17 — 22) used by the softkeys.

Line 7030 subtracts 16 from the ASCII code of a softkey to leave a number in the range 1 to 6. This number then directs the ON...GOTO statement to the appropriate section of the program.

This routine can be implemented as a subroutine if desired. Just append a RETURN statement and change the directed GOTO statement to an ON...GOSUB directive.

Now a program can be kept entirely under programmed operation, menus will automatically be displayed as required and a single stroke of a softkey will take the user to the appropriate program section.

Program *Program Control of Softkeys*

```
7000  WAIT 0:PRINT "LB1  LB2  LB3  LB4  LB5  LB6"
7010  X$=INKEY$:IF X$="" GOTO 7010
7020  X=ASC(X$):IF (X < 17)+(X > 22) THEN 7100
7030  X=X-17:ON X GOTO 1000,2000,3000,4000,5000,6000
```

## FROM THE WATCH POCKET

I attended the **7th West Coast Computer Faire** in San Francisco this past month -- hoping to see the latest in PC wares. The only exhibitor in the PC department was a representative of **Quasar** showing (and selling) their HHC. I'l have more to say about the Quasar unit in a later issue. It is exactly the same machine as the Panasonic HHC which *David Motto* seems favorably impressed with as noted in this issue.

After over a month of extensive use of the Sharp PC-1500, my judgement is that it is a fine machine for the money. Fact is, I have yet to find a PC-1500 owner who is not basically pleased (or delighted!) with the unit. 4K memory expansion modules — kicking the PC to 6K — are already available in the U.S. and the 8K modules are due here by early April. The modules seem a little high priced ($150.00 for the 8K unit), but I could sure use 10K (total) in my PC!

Radio Shack is reported to be planning to start selling their PC-2 (equivalent to the Sharp PC-1500) in May.

If you don't want to wait until then, the Sharp PC-1500 is available through a number of U.S. outlets. *Bob Hall* reports good service from Tam's, Inc., *14932 Garfield Avenue, Paramount, CA 90723.* They take phone orders at 800-421-5188 and have the PC-1500 listed at $249.95, the CE-150 at $199.95 and the CE-151 (4k memory plug-in) at $64.95. On the east coast, **Atlantic NorthEast Marketing,** *PO Box 921, Marblehead, MA 01945,* phone 617-639-0285, has a few interesting offers. If you buy a new PC-1500 and the CE-150 printer/plotter/cassette interface, you can trade in a Sharp PC-1211 or Radio Shack PC-1 (in working condition) for $80.00 credit. On the other hand, if you are looking to pick up some used (but guaranteed to be in good working condition) PC-1211s and PC-1s and associated equipment, *Mort Rosenstein* says they have a good stock available. Call for prices and details. He (Mort) also says the PC-1500 Service Manual with all the technical details will be available shortly for $10.00.

Send a self-addressed stamped envelope to **Walt Moffett,** *Box 1108, Sebastopol, CA 95472,* if you are interested in obtaining printed templates for your PC. He has a printing machine with just the right size characters. In addition to having several "standardized" versions available, he says he can make custom templates too.

The April issue of **BYTE Magazine** features a new pocket-sized telecomputing terminal from a new outfit called **IXO, Incorporated.** A lot of human-factors engineering reportedly went into the terminal which contains a built-in modem, system-connect protocol software, a 1K user RAM buffer and color-coded keyboard. Things are really starting to pop in the PC world! — *Nat Wadsworth, Editor*

35 Old State Rd., Oxford, CT 06483

## PRODUCT REVIEW
## THE SHARP PC-1500

**Produced by:** *Sharp Electronics Corporation.*
**List Price:** *< $300.00*
**Availability:** *Sharp distributors.*
**Reviewer:** *David G. Motto, 3639 Roosevelt, Jackson, MI 49203.*

The new Sharp PC-1500 is not as big as the Panasonic HHC, but it is too large to fit into a shirt pocket.

The keyboard is typewriter-style with staggered keys. A numeric keypad is at the right of the alphabetical keys. The only keys that automatically repeat when held down (in the executive mode) are the cursor controls. These serve the same functions as those on the earlier PC-1211 unit.

The display consists of a 7 by 156 dot liquid-crystal matrix. In the executive or edit mode the display scrolls up to 80 characters across its 26-character width. In the "graphics" mode each dot on the display may be individually activated through a column-addressing technique. The display also has a number of annunciators that indicate the status of the machine.

The PC-1500 has several operating modes. The RUN mode is used to execute programs or perform calculations. The PRO (program) mode is used to create and edit programs. A RESERVE mode may be used to assign operations and functions to a set of six user-definable keys. Each such key may be assigned three different operations. Labels that appear on the display may be created to identify the function of each user-definable key. A key labeled RCL (recall) is used to bring up the key notations when desired. Another key is used to cycle between user-definable-key modes referred to as I, II and III that give those keys their versatility.

There is a 60-pin connector on the left side of the computer. A compartment into which additional memory may be plugged is provided in the back of the unit. Another compartment accessed from the back houses four size AA batteries that power the unit in portable operation. An a.c. adaptor (optional, not provided with the unit) may be plugged into the right side of the unit.

A special TIME function is provided by the computer. It holds the date and time in the numeric format: MMDDHH.mmss where MM represents the month, DD the day of the month, HH the hour, mm the minute and ss the second. The internal clock can be set by the user and then automatically maintains the correct time.

A programmable audio generator permits the playing of notes that may be varied in pitch and duration.

The BASIC language that is provided on this unit is a considerable upgrade from their earlier unit yet it is also upwards compatible. Programs written for the PC-1211 will run on the new PC-1500 as long as no tricks (such as implied multiplication) are attempted. However, there are some differences in the manner in which variables are handled. The PC-1500 has a separate memory area reserved for the single-character variables A through Z and the string variables A$ through Z$. You can now use both the variable A and A$ at the same time. The single character string variables A$ through Z$ are limited to sixteen characters maximum. (Other two-character string variables, such as AA$, can be dimensioned to hold up to 80 characters.) If you want to use the locations set aside for variables A — Z and A$ — Z$ as part of an array using subscripting, then the special symbol "@" is used along with the subscript. Thus, @(1) would be stored in the location assigned to variable A and @$(26) would correspond to Z$.

Two-character variables are stored in the user's memory area. One- or two-dimensional arrays (numeric or string) may be utilized.

This version of BASIC supports the string functions ASC, CHR$, LEFT$, MID$, RIGHT$, STR$, VAL and LEN. Strings can be compared on a character-by-character basis for equality or the greater than or less than (using the ASCII character code) condition.

New BASIC functions include AND, OR and NOT.

Program errors may be located by pressing the line scroll up key immediately after receipt of an error message. There is a TRACE mode that allows the operation of a program to be closely examined. For instance, the values of variables may be examined after the execution of each statement if desired. Error messages are coded as numbers. The nice feature of being able to go backwards in a program listing has been retained.

An INKEY$ statement allows keyboard entries (under program control) to be made without having to use the ENTER key.

Programmers will also like the built-in hexadecimal (up to 4 digits) conversion routine that translates to decimal values.

The PC-1500 is supplied with a black vinyl, padded carrying case and two manuals. The Instruction Manual is better written than previous ones, but it sometimes tends to be over-cute. The programs supplied in the Applications Manual are sometimes re-hashes of earlier ones. Many of the programs also require the use of the accessory printer/plotter and cassette interface and/or the 4K optional memory module.

An accompanying program presents the time of day each time the PC-1500 is turned on. It demonstrates some of the new string handling statements available on this PC.

A second short program, for those that may already have a PC-1500, is based on graphics produced by *Michael Lamping*. It demonstrates simple animation and sound effects

```
 10:ARUN :WAIT 50:
    A=TIME :A=A-10
    0*INT (A/100)
 20:D$="AM":B$=
    STR$ (INT A-12
    *(INT A>12)+12
    *(INT A=0)):IF
    A>=12LET D$="P
    M"
 30:C$=LEFT$ (MID$
    (STR$ (A-INT A
    ), 3, 2)+"00", 2)
 40:CURSOR 9:PRINT
    B$+":"+C$+" "+
    D$:CLEAR
```

**Time Program**
*This routine, when placed as the first program in memory, will automatically display the current time for a brief period whenever the PC-1500 is turned on. Notice the ARUN statement in line 10 which is the key to this automatic power-on operation.*

**Animated Runner Program**
*As an example of the type of animated graphics that can be executed on the PC-1500, this routine presents a surprisingly smooth animated display of a man running across the screen, complete with sound effects. If you want to study how the artist achieved the fluidity, slow the routine down by changing the WAIT statement in line 10 to read WAIT 100.*

```
 10:WAIT 03:CLS
 20:FOR A=0TO 140
    STEP 6
 40:GCURSOR A:
    GPRINT "000048
    241E0F2E1C02"
 60:GCURSOR A+1:
    GPRINT "002020
    2C1E0F0E0C32"
 80:GCURSOR A+2:
    GPRINT "000010
    241E0F0E7404"
100:GCURSOR A+3:
    GPRINT "000020
    203E0F4E3400":
    BEEP 1,5,5
120:GCURSOR A+4:
    GPRINT "000000
    002E7F3E0000"
140:GCURSOR A+5:
    GPRINT "000000
    4C3E0F2E3400"
160:NEXT A:GPRINT
    "000000000000
    00000"
180:GOTO 20
```

## PEEKING IN THE PC-1500

It didn't take users long to discover that there are a number of capabilities of the Sharp PC-1500 that are not mentioned in the instruction manual. Undocumented directives include PEEK, POKE, CALL and OPN. *George Fergus* was one of the first users to get a report in to *PCN* listing some of his discoveries using the PEEK instruction. This command permits one to examine the contents of memory by giving the directive: PEEK X, where X represents an address in decimal or hexadecimal (indicated by preceeding the value with the "&" sign) format. Some of the information George has gleaned, such as the locations (addresses) of RAM elements in the basic unit and the tokens used in the BASIC interpreter, is shown in accompanying tables.

If you want to do some exploring on your own, an accompanying routine provides a means of "dumping" portions of memory using the CE-150 printer. In addition to providing a hexadecimal dump, the program prints ASCII-equivalent characters for appropriate values underneath each line of hexadecimal code. Examining memory starting at hexadecimal address C000 (49152 decimal) reveals the coding of the BASIC interpreter. If you use the dump to examine the first few hundred bytes of ROM code you will discover the token conversion table used by the interpreter.

For those of you who may not have had experience with the PEEK directive, *Norlin Rober* provides an introduction to the subject in an article elsewhere in this issue.

Table *PC-1500 BASIC Token Codes*

| | | | | | |
|---|---|---|---|---|---|
| F1 50 | AND | F1 81 | ARUN | F1 B1 | TO |
| F1 51 | OR | F1 82 | BEEP | F1 B3 | WAIT |
| F1 58 | MEM | F1 83 | CONT | F1 B4 | ERROR |
| F1 5B | TIME | F1 86 | GRAD | F1 B5 | LOCK |
| F1 5C | INKEY$ | F1 87 | CLEAR | F1 B6 | UNLOCK |
| F1 5D | PI | F1 8A | CALL | | |
| F1 60 | ASC | F1 8B | DIM | F0 84 | CURSOR |
| F1 61 | STR$ | F1 8C | DEGREE | F0 85 | USING |
| F1 62 | VAL | F1 8D | DATA | F0 88 | CLS |
| F1 63 | CHR$ | F1 8E | END | F0 89 | CLOAD |
| F1 64 | LEN | F1 92 | GOTO | F0 8F | MERGE |
| F1 65 | DEG | F1 94 | GOSUB | F0 90 | LIST |
| F1 66 | DMS | F1 96 | IF | F0 91 | INPUT |
| F1 67 | STATUS | F1 98 | LET | F0 93 | GCURSOR |
| F1 68 | POINT | F1 99 | RETURN | F0 95 | CSAVE |
| F1 6B | SQR | F1 9A | NEXT | F0 97 | PRINT |
| F1 6D | NOT | F1 9B | NEW | F0 9F | GPRINT |
| F1 6E | PEEK# | F1 9C | ON | F0 B2 | CHAIN |
| F1 6F | PEEK | F1 9D | OPN | F0 B5 | COLOR |
| F1 70 | ABS | F1 9E | OFF | F0 B6 | LF |
| F1 71 | INT | F1 A0 | POKE# | F0 B7 | LINE |
| F1 72 | RIGHT$ | F1 A1 | POKE | F0 B8 | LLIST |
| F1 73 | ASN | F1 A2 | PAUSE | F0 B9 | LPRINT |
| F1 74 | ACS | F1 A3 | P | F0 BA | RLINE |
| F1 75 | ATN | F1 A4 | RUN | F0 BB | TAB |
| F1 76 | LN | F1 A5 | FOR | F0 BC | TEST |
| F1 77 | LOG | F1 A6 | READ | | |
| F1 78 | EXP | F1 A7 | RESTORE | E7 A9 | RMT |
| F1 79 | SGN | F1 A8 | RANDOM | | |
| F1 7A | LEFT$ | F1 AA | RADIAN | E6 80 | CSIZE |
| F1 7B | MID$ | F1 AB | REM | E6 81 | GRAPH |
| F1 7C | RND | F1 AC | STOP | E6 82 | GLCURSOR |
| F1 7D | SIN | F1 AD | STEP | E6 83 | LCURSOR |
| F1 7E | COS | F1 AE | THEN | E6 84 | SORGN |
| F1 7F | TAN | F1 AF | TRON | E6 85 | ROTATE |
| F1 80 | AREAD | F1 B0 | TROFF | E6 86 | TEXT |

Table *PC-1500 Partial Memory Map*

| | | | |
|---|---|---|---|
| &4008 | – | 40C3 | Reserve Program Area (188 bytes) |
| &40C6 | – | 47FF | User Main Program Area (1850 bytes) |
| &7050 | – | 70FF | Fixed Variables E$ – O$ (11 x 16 bytes) |
| &7150 | – | 71FF | Fixed Variables P$ – Z$ (11 x 16 bytes) |
| &78C0 | – | 78FF | Fixed Variables A$ – D$ (4 x 16 bytes) |
| &7900 | – | 79CF | Fixed Variables A – Z (26 x 8 bytes) |
| &C000 | – | FFFF | BASIC Interpreter (16K bytes in ROM) |

Program *PC-1500 Hexadecimal Memory Dump*

```
10:CSIZE 1:INPUT
   "STARTING ADDR
   ESS? ";A
20:A$="0123456789
   ABCDEF"
25:WAIT 0:W=INT (
   A/4096)
26:X=INT ((A-(W*4
   096))/256)
27:Y=INT ((A-(W*4
   096)-(X*256))/
   16)
28:Z=INT ((A-(W*4
   096)-(X*256)-(
   Y*16)))
29:LPRINT MID$ (A
   $, W+1, 1);MID$
   (A$, X+1, 1);
   MID$ (A$, Y+1, 1
   );MID$ (A$, Z+1
   , 1);"   ";
30:WAIT 0:FOR B=0
   TO 7:C=(PEEK (
   A+B))AND (&F0)
   :D=1+INT (C/16
   ):E=(PEEK (A+B
   ))AND (&0F)
40:LPRINT MID$ (A
   $, D, 1);MID$ (A
   $, E+1, 1);" ";:
   NEXT B
45:LPRINT :LPRINT
   "         ";:FOR
   B=0TO 7:LPRINT
   (CHR$ (PEEK (A
   +B)));"  ";:
   NEXT B:LPRINT
   :LF 1
50:A=A+8:GOTO 25
```

Table *PC-1500 Key Memory Locations*

| | | | |
|---|---|---|---|
| &7600 | – | 764D | Display memory |
| &7700 | – | 774D | Display Memory |
| &764E | – | 764F | RUN/PRO/RESERVE settings, trig mode, SHIFT status, SML status |
| &7800 | – | 78BF | Memory pointers, format settings |
| &79D0 | – | 79FE | Settings primarily involving the CE-150 |
| &79FF | | | LOCK/UNLOCK status |
| &7A00 | – | 7AFF | Calculation registers, data stack, FOR/NEXT stack, subroutine stack, etcetera |
| &7B00 | – | 7B07 | Random number storage |
| &7B08 | – | 7B0F | Counter, used for automatic 7-minute shutoff, possibly other purposes |
| &7B10 | – | 7B5F | String buffer |
| &7B60 | – | 7BAF | Data to be outputted to display |
| &7BB0 | – | 7BFF | Input buffer |
| &A000 | – | BFFF | ROM in the CE-150 |

**PC-1500 Version**

Implied multiplication, used extensively to fit the program in a PC-1, cannot be used in a Radio Shack PC-2/Sharp PC-1500. However, the additional memory in these models allows the program to fit with room to spare when implied directives are changed to explicit representations.

An accompanying listing illustrates how the program appears on a PC-1500/PC-1. Ordinarily, *PCN* readers will be left on their own to adapt PC-1211/PC-1 programs to PC-1500/PC-2 units. This might typically involve watching out for implied multiplication as well as dropped right hand parens, ending quotation marks, etc.

STATUS 1 at end of listing indicates the number of bytes utilized.

Program *Curve Fitting (Sharp PC-1500/Radio Shack PC-2 Version)*

```
  1:" "USING :
    PRINT "CURVE F
    IT-V4":INPUT "
    CLR (Y/N)";Z$:
    IF Z$="Y"CLEAR
  2:C=1:Z=N+1:
    PAUSE "PT#";Z:
    INPUT "X=";L:
    IF L<=0LET J=1
  3:X=L:INPUT "Y="
    ;Y:IF Y<=0LET
    T=1
  4:GOSUB 5:BEEP 2
    :PRINT "X";N;"
    =";X;" Y";N;"=
    ";Y:GOTO 2
  5:D=D+C*X:E=E+C*
    X*X:F=F+C*Y:G=
    G+C*Y*Y:H=H+C*
    X*Y:N=N+1*C:IF
    J*T=1RETURN
  6:IF J<>1LET M=M
    +C*LN X:O=O+C*
    LN X*LN X:K=K+
    C*Y*LN X
  7:IF T<>1LET P=P
    +C*LN Y:Q=Q+C*
    LN Y*LN Y:I=I+
    C*X*LN Y
  8:IF J+T=0LET S=
    S+C*LN X*LN Y
  9:RETURN
 10:"L"IF W=1THEN
    14
 11:U=1:PRINT "LIN
    . Y=B*X+A"
 12:B=(H-D*F/N)/(E
    -D*D/N):A=(F/N
    -B*D/N)
 14:R=(N*H-D*F)/√(
    (N*E-D*D)*(N*G
    -F*F)):IF W=1
    RETURN
 16:GOTO 96
 20:"A"IF W=1THEN
    24
 21:U=2:PRINT "EXP
    . Y=A*EXP(B*X)
    ":IF T=1GOTO 6
    1
 22:B=(I-P*D/N)/(E
    -D*D/N):A=EXP
    (P/N-B*D/N)
 24:R=(N*I-D*P)/√(
    (N*E-D*D)*(N*Q
    -P*P)):IF W=1
    RETURN
 26:GOTO 96
 30:"S"IF W=1THEN
    34
 31:U=3:PRINT "LOG
    . Y=B*LNX+A":
    IF J=1GOTO 61
 32:B=(K-M*F/N)/(O
    -M*M/N):A=(F-B
    *M)/N
 34:R=(N*K-F*M)/√(
    (N*O-M*M)*(N*G
    -F*F)):IF W=1
    RETURN
 36:GOTO 96
 40:"D"IF W=1THEN
    44
 41:U=4:PRINT "PWR
    . Y=A*X^B":IF
    T+J>0GOTO 61
 42:B=(S-M*P/N)/(O
    -M*M/N):A=EXP
    (P/N-(B*M/N))
 44:R=(N*S-M*P)/√(
    (N*O-M*M)*(N*Q
    -P*P)):IF W=1
    RETURN
 46:GOTO 96
 48:"F"IF J+T>0
    THEN 61
 50:W=1:U=0:V=0:
    FOR Z=1TO 4:
    GOSUB (Z*10):C
    =R*R:IF C>V
    GOSUB 99
 55:NEXT Z:USING :
    W=0:BEEP 3:
    PRINT "BEST FI
    T WAS #";U
 59:GOTO (U*10)
 60:"K"PRINT "ESTI
    MATE OF X":
    INPUT "Y=";V:
    GOTO (U+61)
 61:BEEP 1:PRINT "
    X OR Y OR BOTH
     NEG.":END
 62:L=(V-A)/B:GOTO
    97
 63:L=(LN V-LN A)/
    B:GOTO 97
 64:L=EXP ((V-A)/B
    ):GOTO 97
 65:L=EXP ((LN V-
    LN A)/B):GOTO
    97
 67:USING "#####.#
    ##":RETURN
 70:"G"BEEP 1:
    PRINT "DELETE"
    :INPUT "DELETE
     LAST PT (Y/N)
    ";Z$:IF Z$="Y"
    THEN 75
 71:PRINT "PT# ";N
    :INPUT "X=";L:
    IF L<=0LET J=1
 72:X=L:INPUT "Y="
    ;Y:GOSUB 98:
    GOTO 77
 75:PRINT "X=";X;"
     Y=";Y:INPUT "
    OK TO DEL(Y/N)
    ";Z$
 76:IF Z$="N"THEN
    71
 77:C=-1:IF Y<=0
    LET T=1
 78:GOSUB 5:END
 82:"J"PRINT "ESTI
    MATE OF Y":
    INPUT "X=";L:
    GOTO (U+85)
 86:V=B*L+A:GOTO 9
    7
 87:V=A*EXP (B*L):
    GOTO 97
 88:V=B*LN L+A:
    GOTO 97
 89:V=A*L^B:GOTO 9
    7
 96:GOSUB 67:PRINT
    "A=";A:PRINT "
    B=";B:PRINT "R
    =";R:C=R*R:
    PRINT "R^2=";C
    :END
 97:PRINT "X=";L;"
     Y=";V:END
 98:PRINT "X=";X;"
     Y=";Y:RETURN
 99:U=Z:V=C:RETURN

STATUS 1
                1677
```

## PUT THOSE SOFTKEYS TO WORK FOR YOU!

As *George Fergus* was quick to discover, you can quickly set up the Sharp PC-1500 to save yourself a lot of keying when developing programs. Just assign the most commonly used punctuation symbols that normally require two keystrokes (SHIFT and then the desired character) to a group of softkeys. For instance, using all six keys from left to right, you could define them to represent the symbols:

: ; . ? # $

Use the RCL key too, to display what you have assigned to each softkey to assist in remembering your assignments. The technique sure saves a lot of work.

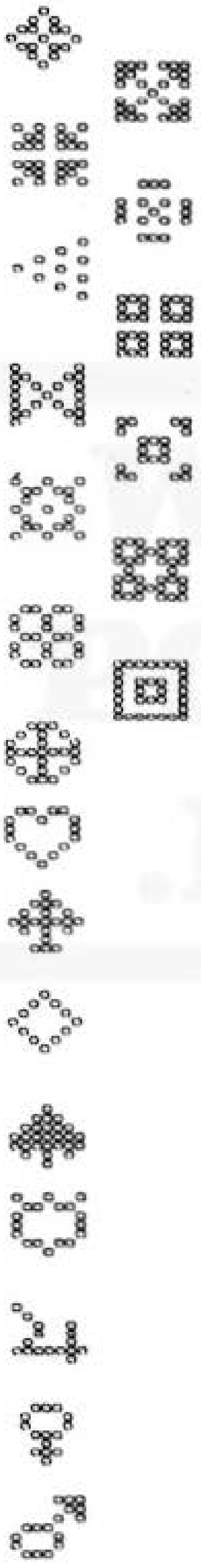
## LCD GRAPHICS FOR THE SHARP PC-1500

*David G. Motto, 3639 Roosevelt Circle, Jackson, MI 49203*, sent in the codes shown in the accompanying listing for producing a variety of graphics on the Sharp PC-1500 liquid crystal display.

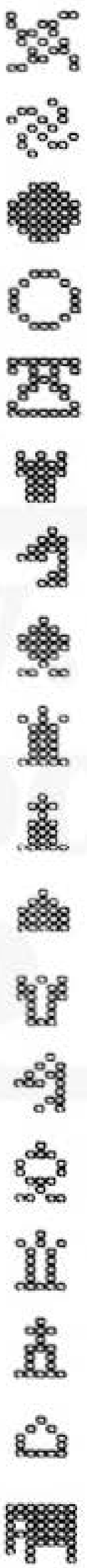
The codes are presented within DATA statements. A simple routine at the start of the listing (lines 1 – 100) may be used to access the data statements and display the various patterns. Naturally, you can extend the concepts to produce your own new patterns and incorporate them in your own programs. Note the complete set of chess pieces provided by Dave. Does that give anyone ideas? See pages 91 and 92 of the *Sharp PC-1500 Instruction Manual* for details on producing LCD graphics.

**Program** *LCD Graphics for the Sharp PC-1500 and Radio Shack PC-2.*

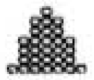
```
   1:RESTORE 1000
   5:DIM X$(0)*80
 100:CLS :WAIT 100:
     GCURSOR 0:READ
     X$(0):GPRINT X
     $(0):GOTO 100
1000:DATA "081436
     49361408"
1010:DATA "776355
     08556377"
1020:DATA "553677
     00773655"
1030:DATA "1C0055
     4955001C"
1040:DATA "080014
     002A0055"
1050:DATA "775577
     00775577"
1060:DATA "7F2214
     0814227F"
1070:DATA "63411C
     141C4163"
1080:DATA "493622
     49223649"
1090:DATA "775D77
     22775D77"
1100:DATA "364949
     36494936"
1110:DATA "7F415D
     555D417F"
1120:DATA "1C2A49
     7F492A1C"
1200:DATA "0E1121
     4221110E":
     REM   HEART
1210:DATA "081C4A
     7F4A1C08":
     REM   CLUB
1220:DATA "081422
     41221408":
     REM   DIAMON
     D
1230:DATA "183C1E
     7F1E3C18":
     REM   SPADE
1300:DATA "1D2222
     4122221D":
     REM   TAURUS
1310:DATA "21322C
     20207C20":
     REM   JUPITE
     R
1320:DATA "062979
     2906":REM
     FEMALE
1330:DATA "304848
     48350307":
     REM   MALE
```



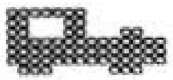
```
1340:DATA "434C34
     2A161961":
     REM   SPIRAL
1350:DATA "04324A
     14292610":
     REM   SPIRAL
1360:DATA "1C3E7F
     7F7F3E1C":
     REM   BALL
1370:DATA "1C2241
     4141221C":
     REM   CIRCLE
1380:DATA "63594F
     454F5963":
     REM   BELL
1400:DATA "0E7C7E
     7C0E":REM
     ROOK (B)
1410:DATA "080C4E
     677E":REM
     KNIGHT (B)
1420:DATA "4C5E3F
     5E4C":REM
     BISHOP (B)
1430:DATA "427C7F
     7C42":REM
     QUEEN (B)
1440:DATA "407A7F
     7A40":REM
     KING (B)
1450:DATA "1C1E1F
     1E1C":REM
     PAWN (B)
1500:DATA "0E7C42
     7C0E":REM
     ROOK (W)
1510:DATA "080C4A
     257E":REM
     KNIGHT (W)
1520:DATA "4C5233
     524C":REM
     BISHOP (W)
1530:DATA "427C43
     7C42":REM
     QUEEN (W)
1540:DATA "407A4F
     7A40":REM
     KING (W)
1550:DATA "1C1211
     121C":REM
     PAWN (W)
1600:DATA "3F0B7F
     1F1F1F7F":
     REM   ELEPHA
     NT
```



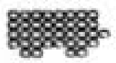
```
1610:DATA "40787E
     7F7E7840":
     REM   BUILDI
     NG
```



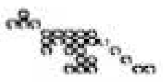
```
1620:DATA "3F7F79
     79393F3F3838
     787C7C3838":
     REM   LOCOMO
     TIVE
```



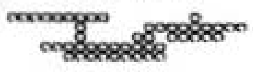
```
1630:DATA "3C7C7C
     3C3C7C7C3C10
     ":REM   BOXC
     AR
```



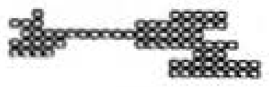
```
1640:DATA "020302
     1C0C5C7C6C08
     10204040":
     REM   KANGAR
     OO
```



```
1650:DATA "010101
     0909191F1919
     18181C1E1A06
     060706060202
     02":REM   EN
     TERPRISE
```



```
1660:DATA "1A1E1F
     1C0C04040404
     04040E0E0E6F
     6F7F7B7B6860
     60":REM   KL
     INGON
```



```
1670:DATA "081412
     122448102040
     000000402010
     102040000000
     402010102040
     0000004020"
```

## UNDERSTANDING THE SHARP PC-1500

This is the second article in a series being presented by: *Norlin Rober, 407 North 1st Avenue, Marshalltown, IA 50158.*

### An Introduction to POKE

In Issue 15 of *PCN* I explained that the PEEK function returned the byte that was stored in the address used as an argument. It is also possible to store a specified byte at a particular memory location, by using the POKE instruction.

For example, execution of POKE &79FF,0 will store a zero byte in the memory location &79FF and thus place the PC in the LOCK mode. (Try it!) As you would expect if you read the previous article, performing a POKE &79FF,&60 will cancel the LOCK mode.

A multiple POKE directive is possible on the PC-1500. When the directive POKE &7900,1,0,116,104,80,0,0,0 is executed, the byte 01 is stored into location &7900, 00 goes into &7901, and so on. This particular example will store the numerical value 78.685 into the variable A as discussed in the previous article.

### Practical Applications

It may not be immediately obvious that there are a number of ways in which the PEEK and POKE instructions can be quite useful. The following examples should give you some notion of the kinds of possibilities.

For instance, certain memory locations act as *pointers*. That is, they contain the hexadecimal addresses of various memory locations needed by the computer, such as the location of the next instruction to be executed or the location of a stored variable.

Suppose you have executed the NEW command, effectively clearing your program from memory, and then realize you still want to use that program. Is it possible to recover the lost program? It is, provided that prior to having executed NEW you have saved the numbers obtained by PEEK &7867 and PEEK &7868. The contents of these locations act as a pointer to the memory address of the last byte in a program! The use of the NEW command does not actually erase a program from memory. It actually just resets some pointers so that the computer "thinks" that the memory is empty.

To test this, put a small program into the PC-1500 and write down the results of the two PEEKs just mentioned. Now execute NEW. To resurrect your program: (1) POKE the two previously saved PEEKed values back into their original locations and (2) POKE &40C5,0. (Note, if you have an 8K CE-155 memory module installed, step (2) should be POKE &38C5,0.) If the first line number in your program was larger than 255, you will find that it has been changed. You can edit it back to its original value.

You can also restore variables in main memory using a similar procedure. For instance, with some variables stored in main memory (created using two-character names), record the values obtained using PEEK &7899 and PEEK &789A. If these same values are POKEd back into those locations after main-memory variables have been "cleared" by CLEAR, RUN or NEW, the original variables will be restored. You see, the pointer in locations &7899 and &789A simply contains the starting address where variables are stored. (Note that this does not apply to the fixed-memory variables A through Z and A$ through Z$.)

Print formats may be set using the POKE directive instead of a USING statement, with the advantage that a calculated value may be used to determine the number of spaces formatted. Here is how the format settings are stored in memory:

| | |
|---|---|
| &7895 | The byte stored here is the sum of the following format specifications: 128 for scientific notation, 64 for asterisk fill, 32 for forced sign, 16 for comma separation. (Note: when set by USING, the computer adds 1 to these values, except when the format is simply "^".) |
| &7896 | The number of positions reserved preceding the decimal point, including one for the sign. |
| &7897 | Length specified for strings, if any. |
| &7898 | The number of positions reserved following the decimal point, including one for the decimal point itself. |

Thus, POKE &7895,32,7,0,3 would be equivalent to the statement USING "+######.##".

### Fooling the Printer

If you have the CE-150 printer connected, you can PEEK &79F4 to see what CSIZE is set. The largest size you can print is CSIZE 9, right? *Wrong!* Try POKE &79F4,50. Then LPRINT "G" and take a look at CSIZE 50! Now set the computer to GRAPH mode, execute the statement ROTATE 1, then POKE &79F4,36. Now execute LPRINT "HUGE". Be advised that if you continue this for very long, the PC will exceed its coordinate range resulting in ERROR 70. You can remedy this condition by executing SORGN so that it has a new origin. You might also discover that the computer interprets CSIZE 0 as CSIZE 256. The only thing printable in this size is an immense decimal point, obtainable by specifying LPRINT ".".

The COLOR setting is stored in location &79F3. Do *not* POKE into this location. You will only confuse the computer into thinking that the pen turrett is positioned differently from where it actually resides.

*[Note: Be careful using the POKE directive around memory locations used to control external units. It is conceivable that you might damage an external device if you inadvertently left a selenoid activated or otherwise interfered with the normal programmed operation of the device. At the very least, be prepared to immediately shut the unit off if strange sounds or activities come forth as the result of a POKE directive. – N.W.]*

### More PEEK and POKE Locations

I have located a number of memory addresses used by the PC-1500 using a process of trial and error. And, of course, there are quite a few addresses used for purposes that I have not yet figured out. The accompanying table lists those found recently. Be forewarned that if you poke values into certain locations that attempt to have the computer do something that is not possible, you may get a "crash." The PC may then completely lock up and you will have to use the ALL RESET switch to restore operation.

### A Surprise PEEK

A number of PC-1500 fans are investigating the ROM coding, obtaining the hexadecimal codes and/or interpretations of these as ASCII characters. Here is another way of looking at the ROM that you may not have considered. The program illustrated can be used to display ROM codes as graphics on the LCD.

```
10 INPUT "BEGINNING ADDRESS?";A
20 WAIT 0:FOR C=0 TO 127:GPRINT PEEK(A+C);:NEXT C:WAIT:
   PRINT
30 A=A+128:GOTO 20
```

Try using &C000 as the initial input to the beginning address prompt. When you get bored with this, take a look at the results when you use &FCA0 as a beginning address!

POINTERS

| | |
|---|---|
| &7863 | Beginning of memory. This location contains the first byte of the two-byte adress of where memory begins. Only one byte is stored because the second byte is always zero. The beginning of memory can vary depending on what expansion module, if any, is plugged into the PC. |
| &7864 | Top of memory. Again, the second byte is always zero. This is the address of the first byte following the end of "main" (user) memory. |
| &7865 – 66 | Beginning of program. This is the address following RESERVE memory. It is where the first program line of a user program is stored. |
| &7867 – 68 | End of program. The computer stores a byte with the code &FF in the address pointed to by this pointer. Thus &FF is the byte immediately after the carriage-return (ENTER) code of the user's last program line. |
| &7869 – 6A | Beginning of merged program (a program loaded from cassette using the MERGE command). |
| &7899 – 9A | Beginning of variables. |

## POINTERS USED DURING PROGRAM EXECUTION

| | |
|---|---|
| &789C – 9D | Hexadecimal value of line number being executed. |
| &789E – 9F | Beginning of the program being executed. (It is the start of a merged program if that is what is being executed.) |
| &788E – 8F | Pointer to address at which next READ statement will obtain data. If none, the value C0C8 is held in the pointer. |

## SIX-BYTE POINTERS USED IN PROGRAM EXECUTION

Each of these pointers, starting at the location shown, contains six bytes. The first two bytes specify the storage address in memory; the next two contain the program line number; and the last two hold the memory address of the first byte of the program being executed.

| | |
|---|---|
| &78A0 | Address of next program byte. It is the next byte to be executed unless a transfer (GOTO, etc.) has been specified. |
| &78A6 | Address of next byte to be executed after a transfer. |
| &78AC | Address where execution is to resume after a halt for input, etc. |
| &78B2 | Address of program location at which an error occurred. |
| &78B8 | Transfer destination specified by ON ERROR GOTO directive. When cancelled by RUN, &80 is added to the first byte of this stored address. |

## STORED MODES AND SETTINGS

| | |
|---|---|
| &764E | Status of prefix keys; specification of RESERVE group. |
| &764F | Operating mode and trigonometric mode. |
| &7863 | Beeper on/off. Set to 0 for on, 1 for off. |
| &7871 – 73 | WAIT specification. |
| &7874 – 75 | Position of display cursor (GCURSOR). |
| &788D | Trace mode. TRON = 96, TROFF = 0. |
| &7895 – 98 | Format (USING) specifications. (See text.) |
| &79FF | Lock mode. LOCK = 0, UNLOCK = 60. |
| &7B0A – 0C | Automatic 7-minute shutoff timer. Reset to value &FE 1D 1D whenever activity halts. Shuts PC off when value &FF FF FF is passed. |
| &7B0D | Timer for cursor flashing. |
| &7B0E | Type of program execution in progress. Continuous execution = 1, single-line execution (stepped by user with the line scroll-down key) = 193, and rapid execution when scroll-down key is held depressed = 225. |
| &7B0F | Keyboard matrix code for the key that initiated program execution. This would be ENTER or user-defined or a RESERVE key. |

## LOCATIONS RELATED TO PRINTER OPERATION

| | |
|---|---|
| &79E0 – E1 | The X-coordinate relative to the origin when in the GRAPH mode. Negative numbers are stored in complemented form. |
| &79E2 – E3 | The Y-coordinate. |
| &79E4 – E5 | Paper reverse counter. Used to insure that paper does not get backed up more than about 4 inches. |
| &79E6 | Location (horizontal) of pen (0 – 216). |
| &79E7 – E8 | In GRAPH mode, the amount by which the horizontal coordinate exceeds the range in which printing may take place. This is in hexadecimal notation with the low byte first. |
| &79E9 | When paper is fed by the computer the value 15 is stored here. When fed manually, the value is 0. |
| &79F2 | ROTATE setting. |
| &79F3 | COLOR setting. |
| &79F4 | CSIZE setting. |

## CONVERTING PROGRAMS FOR THE CASIO FX-702P

Several *PCN* readers have expressed dismay over the lack of programs available for the Casio FX-702P. A recent survey of PC users leads us to believe that the situation is not likely to improve substantially in the immediate future. There are simply relatively few FX-702P users compared to Radio Shack and Sharp owners, according to our findings.

But, FX-702P owners need not despair. The fact is that most of the programs published to date in *PCN* are readily adapted for use on the Casio PC by making some editing changes as programs are keyed in.

The following suggestions and tips may be helpful to Casio FX-702P users who wish to make such program adaptations.

1.) Substitute appropriate statement/function mnemonics where applicable. Most of these types of alterations are pretty obvious. Thus, for instance, GOSUB is changed to GSB, PRINT to PRT, RETURN to RET, and so forth. A few may not be quite so apparent. The PAUSE statement becomes WAIT (x) where (x) specifies the length of the pause. CLEAR must be changed to VAC on the 702P. You need to type in SQR in place of the square root (√) symbol. Also, remember that there is a special inequality symbol (≠) on the 702P that takes the place of having to use the "less than" and "greater than" ( < > ) symbols to express the inequality operation on a Sharp or Radio Shack PC.

2.) Forget about functions that do not exist on the FX-702P – such as BEEP.

3.) Shorten/abbreviate display messages to fit in the 20 character FX-702P display (versus the 24 characters on the PC-1).

4.) If subscripted variables are used in a program, remember the following:

A.) Subscripted variables having subscripts in the range 1 – 26 are really just another way of declaring the use of the variables A – Z on the Sharp/Radio Shack original PCs. Thus A(1) is the equivalent of specifying variable A on a FX-702P, A(2) = B,.... ..., A(26) = Z. Make the appropriate substitutions!

B.) Subscripted variables with subscripts above 26 can be converted to array elements on a FX-702P using the formula (I-26) where "I" represents the subscript on the Radio Shack/Sharp unit. I.e., A(27) = A(27-26) = A(1), A(28) = A(28-26) = A(2) on the 702P, etc. Simple, isn't it?

5.) Use separate IF statements in place of multiple logic expressions. An expression such as "IF (A < 52) + (B > 60) THEN 400" on a Radio Shack TRS-80 PC means "if A less than 52 *OR* B greater than 60 then go to line 400." On the FX-702P this must be precisely defined using two separate (consecutive) IF . . . THEN statements, such as:

```
IF A < 52 THEN 400
IF A > 60 THEN 400
```

The opposite logic operation (*AND*) would appear in a TRS-80 listing as "IF (A < 52) * (B > 60) THEN 400" which indicates that "if A is less than 52 *AND* B is greater than 60 then go to line 400." Such a statement can be re-arranged so that it can be stated using two consecutive IF statements on a 702P in this manner:

```
IF A >= 52 THEN ***        (*** = condition not met)
IF B > 60 THEN 400
*** (line to which program goes when condition is not met!)
```

Note that the first IF statement in this example reverses the original test so that the program can "fall" directly into the second IF statement if the first part of the AND operation is met.

6.) Explicitly state all mathematical operations using their appropriate symbols. This means eliminating all instances of "implied" multiplication and parenthesis which sometimes are inferred in TRS-80 and Sharp PC programs. That is, an expression such as:

```
ATN((ABC + (D - E
```

should be completely specified by inserting appropriate multiplication signs and adding closing parentheses, thus:

```
ATN((A*B*C) + (D - E))
```

Use the six steps outlines in this article as a guide and you will find you can readily adapt most PC programs written for other machines so that they will execute on your FX-702P. In the process, you may even find you *understand* (and can thus adapt and customize) the program better than those who simply load the programs "to see what happens."

## A COMPARISON OF PC MEMORIES

The Radio Shack TRS-80 and the Sharp PC-1211 pocket computers have 204 registers available for the storage of data and programs. Each register can hold eight bytes. The first 26 registers can only hold *data* (not program steps). Registers can hold numbers of up to 12 digits, a two-digit exponent (powers of ten) and the signs of the mantissa and exponent. Each digit or sign takes up 4 bits of space or a *nibble*. Alternately, a register may store up to seven alphanumeric characters. Each characters is represented as 8 bits or a *byte*. The eighth byte in the register is used to indicate that the other bytes are representing *string* (alphanumeric) information.

The remaining 178 registers can hold data or program steps. Program steps begin at the last register (number 204) and proceed to be stored in descending (numbered) registers. Each program step uses one byte of storage. The *curtain* between program memory and data memory is moved automatically as program steps are created or deleted.

Registers may be cleared in two ways, depending on which type of register is involved. The NEW command effectively clears all program and data registers. The CLEAR command only affects the data registers.

If a data register contains a numerical value, such as 123, then trying to refer to that register as a string will result in an error condition.

### A Look at the Casio FX-702P

This unit has 236 data and program registers. Twenty-six of these are reserved strictly for data purposes and 10 are reserved for program steps. The remaining 200 may be used for either data or program storage.

The Casio uses the same method of storage as the TRS-80 and 1211. The curtain, however, must be specifically specified by using the DEFM command. This directive allocates up to 200 registers (by groups of 10) for data storage. It will not permit allocation of memory that is already in use for programs. But, when the curtain is moved to create more data storage, that memory is cleared.

The VAC command clears data memory in the Casio unit and the CLR command clears program memory.

If a data register contains a number and an attempt is made to refer to it as a string, there is no error condition. Instead, the computer returns the "null" string (as though there was nothing there).

### The Sharp PC-1500 & Radio Shack PC-2

These units are different than the earlier PC-1211 and PC-1. They have 26 fixed numeric registers, 26 separate fixed string registers, and 1850 bytes of program storage. The numeric data registers behave the same as described for the other units. However, the string registers are twice as long, holding 16 characters.

The 1850 bytes of storage memory may also be used for data when needed. To do so, a two-character variable name is created . The first character of the name may be any letter from A to Z. The second character may be a letter or a digit. Adding the $ symbol creates a string variable.

Arrays of one or two dimensions may also be created in memory through use of the DIM statement. Both numerical arrays and string arrays having up to 80 characters-per-element may be defined. The assignment of an array name takes seven bytes of memory. Additional memory is then used by each element of the array.

The CLEAR and NEW commands operate similar to the earlier PCs by these firms. However, variables (including arrays) in the program workspace are not set to zero by the CLEAR directive; that memory space simply becomes available for other use.

The RUN command, used to start a program, clears all variables in the program workspace, but it does not alter those in fixed memory.

### The Panasonic HHC

The Panasonic does things differently than the other machines. It does not have any fixed registers. Each variable, including arrays, uses up program workspace as it is needed. There are three types of variables and array elements: floating-point numbers, integers and strings. Floating-point numbers have up to 9 digits of accuracy and may range from 2.93873588E-39 to 1.70141183E+38 (plus or minus). Integer values are restricted to the range -32768 to 32767. Strings may contain up to 255 characters.

Each simple variable name (versus array variables) uses up seven bytes of memory. Floating-point variables utilize the first two bytes for the name and the last five for the numeric value. Integers use the first two for the name, the next two for the value, and leave the last three unused!

String variables use two bytes for the name, one for the length of the string, and two more for the memory address of the location of the first character of the string. The last two (of the seven) bytes are not used in a string assignment.

Array names use two bytes for the name, two for the array length, one for the number of dimensions and additional bytes for the size of each dimension. Then the storage of array elements begins. Floating-point elements require five bytes, integer elements two bytes, and string elements use three bytes in the array (length and location) with the actual strings being stored separately.

### Summary

Each PC or HHC handles certain aspects of memory utilization in its own way. However, there is a clear trend towards providing substantially more memory, both in the basic unit and as add-on or plug-in modules. For instance, the PC-1500 and PC-2 can be expanded using plug-ins to (eventually) 18 kilobytes. The Panasonic, using external modules, can be equipped with up to 56 kilobytes!

An accompanying table compares the basic memory configurations of the units discussed.

Thanks for this comparative article go to: *David G. Motto, 3639 Roosevelt Circle, Jackson, MI 49203.*

**Table** *Basic Memory Supplied in Popular Units*

| Model | Fixed Data | Flexible Memory | Fixed Program |
|---|---|---|---|
| 1211/PC-1 | 208 | 1424 | 0 |
| FX-702P | 208 | 1600 | 80 |
| 1500/PC-2 | 624 | 1850 | 0 |
| RL-H1400 | 0 | 3108 | 0 |

Glossary *PC Storage Terminology*

| | |
|---|---|
| bit | The smallest unit of programmable memory, able to represent a zero or a one. |
| nibble | A group of four bits, able to hold a number from 0 to 15. |
| byte | Two nibbles or eight bits, capable of representing numbers in the range 0 to 255. |
| BCD | Binary Coded Decimal, a method of representing the digits 0 through 9 in four bits. |
| curtain | Also known as a partition. It is the hypothetical boundary between program memory and data memory. |
| fixed memory | Memory that is available for only one purpose, be it storage of data or programs. |
| flexible memory | Memory that may be used for either data or programs. |
| program memory | Memory in which program steps are stored. |
| data memory | Memory in which numbers or character strings are stored. |

## MUSIC ON THE SHARP PC-1500

Incredible! J.S. Bach's *Prelude Number II from The Well Tempered Clavier* as programmed by: *Brian Peterson, 6807 N. Sheridan Road – Apt. 520, Chicago, IL 60626.* Brian says he will write a tutorial article showing the rest of us how he makes this kind of music if there is enough interest. So, if that is the case, write and give him some encouragement!

Program *Music on the Sharp PC-1500*

```
10:DIM A(51),B(51)
15:FOR X=1TO 51
20:READ A(X):B(X)=10^(X/40)*8
25:NEXT X
30:WAIT 25:PRINT "ONE":PRINT "AND":
   PRINT "TWO":PRINT "AND":CLS
50:READ N:IF NBEEP 1,A(N),B(N)*1.2:
   GOTO 50
51:BEEP 1,A(11),B(11)*4
52:READ N:IF NBEEP 1,A(N),B(N)*2:
   GOTO 52
53:READ N:IF NBEEP 1,A(N),B(N)*1.5:
   GOTO 53
54:BEEP 1,A(11),B(11)*4
55:READ N:IF NBEEP 1,A(N),B(N)*2:
   GOTO 55
56:READ N:IF NBEEP 1,A(N),B(N)*1.5:
   GOTO 56
57:BEEP 1,A(11),B(11)*4
58:READ N:IF NBEEP 1,A(N),B(N)*2:
   GOTO 58
59:READ N:IF NBEEP 1,A(N),B(N)*1.5:
   GOTO 59
60:READ N:IF NBEEP 1,A(N),B(N):GOTO
   60
61:BEEP 1,A(16),B(16):BEEP 1,A(23),
   B(23):BEEP 1,A(26),B(26):BEEP 1,
   A(28),B(28)
62:BEEP 1,A(32),B(32)*20
63:READ N:IF NBEEP 1,A(N),B(N)*2:
   GOTO 63
64:READ N:IF NBEEP 1,A(N),B(N)*1:
   GOTO 64
65:READ N:IF NBEEP 1,A(N),B(N)*4:
   GOTO 65
66:READ N:IF NBEEP 1,A(N),B(N)*2:
   GOTO 66
67:BEEP 1,A(16),B(16):BEEP 1,A(21),
   B(21):BEEP 1,A(24),B(24):BEEP 1,
   A(28),B(28)
68:BEEP 1,A(33),B(33)*20
69:READ N:IF NBEEP 1,A(N),B(N)*2:
   GOTO 69
70:READ N:IF NBEEP 1,A(N),B(N)*1:
   GOTO 70
71:READ N:IF NBEEP 1,A(N),B(N)*1.2:
   GOTO 71
72:READ N:IF NBEEP 1,A(N),B(N)*2:
   GOTO 72
73:READ N:IF NBEEP 1,A(N),B(N)*4:
   GOTO 73
74:READ N:IF NBEEP 1,A(N),B(N)*6:
   GOTO 74
75:READ N:IF NBEEP 1,A(N),B(N)*8:
   GOTO 75
76:BEEP 1,A(32),B(32)*2:BEEP 1,A(30
   ),B(30)*2:BEEP 1,A(32),B(32)*30
77:END
100:DATA 255,244,229,215,204,193,184
    ,172,163,151
101:DATA 143,133,125,118,111,105,99,
    94,88,82,77
102:DATA 72,68,64,59,55,52,49,45,43,
    40,37,35
103:DATA 32,30,28,26,24,22,21,19,18,
    16,15,14
104:DATA 12,11,10,9,8,7
200:DATA 40,31,38,31,28,31,38,31
201:DATA 40,31,38,31,28,31,38,31
202:DATA 36,33,32,33,28,33,32,33
203:DATA 36,33,32,33,28,33,32,33
204:DATA 35,33,31,33,38,33,31,33
205:DATA 35,33,31,33,38,33,31,33
206:DATA 40,35,33,35,31,35,33,35
207:DATA 40,35,33,35,31,35,33,35
208:DATA 43,36,35,36,31,36,35,36
209:DATA 43,36,35,36,31,36,35,36
210:DATA 42,34,32,34,30,34,32,34
211:DATA 42,34,32,34,30,34,32,34
212:DATA 42,35,34,35,30,35,34,35
213:DATA 42,35,34,35,30,35,34,35
214:DATA 40,32,30,32,28,32,30,32
215:DATA 40,32,30,32,28,32,30,32
216:DATA 40,33,32,33,28,33,32,33
217:DATA 40,33,32,33,28,33,32,33
218:DATA 38,33,31,33,30,33,31,33
219:DATA 38,33,31,33,30,33,31,33
220:DATA 38,35,33,35,31,35,33,35
221:DATA 38,35,33,35,31,35,33,35
222:DATA 36,35,33,35,31,35,33,35
223:DATA 36,35,33,35,31,35,33,35
224:DATA 36,30,28,30,26,30,28,30
225:DATA 36,30,28,30,26,30,28,30
226:DATA 35,26,24,26,31,26,24,26
227:DATA 35,26,24,26,31,26,24,26
228:DATA 33,28,26,28,25,28,26,28
229:DATA 33,28,26,28,25,28,26,28
230:DATA 33,30,28,30,27,30,28,30
231:DATA 33,30,28,30,27,30,28,30
232:DATA 33,30,28,30,27,30,28,30
233:DATA 33,30,28,30,27,30,28,30
234:DATA 31,28,27,28,23,28,27,28
235:DATA 31,28,27,28,23,28,27,28
236:DATA 21,31,30,31,33,31,30,31
237:DATA 21,31,30,31,33,31,30,31
238:DATA 22,28,27,28,31,28,27,28
239:DATA 22,28,27,28,31,28,27,28
240:DATA 31,28,27,28,23,28,27,28
241:DATA 31,28,27,28,23,28,27,28
242:DATA 34,28,27,28,25,28,27,28
243:DATA 34,28,27,28,25,28,27,28
244:DATA 35,28,27,28,30,28,27,28
245:DATA 35,28,27,28,30,28,27,28
246:DATA 36,28,27,28,30,28,27,28
247:DATA 36,28,27,28,30,28,27,28
248:DATA 0,15,18,21,0,24,21,28,21,27
    ,21,30,27,24,21,28,21
249:DATA 0,16,19,23,0,28,23,22,23,31
    ,28,35,31,28,24,23,24
250:DATA 0,13,22,28,0,31,28,27,28,34
    ,28,37,34,31,28,27,28
251:DATA 11,0,42,40,42,43,40,35,40,3
    5,40,35,40,42,35,37,33
252:DATA 35,35,37,35,40,37,35,37,34,
    37,35,37,35,35,34,35
253:DATA 38,47,45,47,40,45,43,45,42,
    45,43,45,47,43,42,43
254:DATA 40,43,42,43,45,42,40,42,35,
    42,40,42,43,40,35,40
255:DATA 35,40,35,40,36,43,43,43,33,
    43,42,43,33,42,40,42
256:DATA 31,40,35,40,36,33,31,33,35,
    31,30,31,33,30,28,30,0
257:DATA 28,30,32,0,23,35,36,38,40,3
    0,36,35,0,33,0,35,32,0
258:DATA 35,33,32,33,35,36,35,0,32,3
    1,30,31,33,30,31,33,0
259:DATA 27,4,15,18,21,24,23,21,27,2
    1,30,21,27,24,23,21
260:DATA 28,25,26,23,28,24,21,24,23,
    26,23,28,24,21,18,21
261:DATA 28,23,20,16,21,18,15,18,0,4
    ,11,16,18,20,23,26,23
262:DATA 24,28,33,30,33,36,0,40,35,4
    0,0,35,33,0,30,0,0
263:DATA 0
STATUS 1
                                3403
```

### STATUS REPORT

A number of PC-1500 users have discovered that in addition to the STATUS 0 and STATUS 1 functions available on the machine, there are several other STATUS-related capabilities. Here is a summary of STATUS functions now reported to be available:

STATUS 0 As indicated in the PC-1500 Instruction Manual, this function tells how many bytes of program memory are available for storage of programs and data. This function does not take account of memory assigned to two-character variables or array elements.

STATUS 1 As reported in the manual, this function tells how many bytes of memory a user's program *listing* has consumed. It does not include the space necessary for the storage of two-character variables or array elements.

STATUS 2 This function is not covered in the manual, but it apparently gives the actual address in memory where a user's program listing ends. If you enter

STATUS 2 - STATUS 1

you will obtain the memory address where user program storage actually begins. This address varies depending on which (if any) memory expansion modules are in use.

STATUS 3 Also not reported in the manual, this function returns the "bottom" of the two-character variables and/or array elements stack. Thus, to find out how much memory you have left when you want to include assignments for two-character variables and array elements, just enter:

STATUS 3 - STATUS 2

STATUS 4 Again is not reported in the manual, but it gives the last *complete* line number executed in a program. You can thus use this function to perform a lot of programming tricks — such as determining what line caused a jump because of an ON ERROR GOTO directive. You can also use this function to determine what part of a large program called a subroutine, etc. You just insert:

X = STATUS 4

in the first line of the subroutine and use X as desired to accomplish your programming objectives.

STATUS 5–9 Appear to duplicate the STATUS 4 function.

Thanks to readers such as *Thomas S. Cox, Brian Peterson, Norlin Rober* and others for reporting these latent capabilities.

**Error Codes** *Summary courtesy of Thomas S. Cox*

PC-1500 ERROR CODES

| CODE | MEANING |
|---|---|
| 1 | Syntax Error Ex: 10 GOTO: 10-3A=1: A=1: 10 NEW |
| 2 | No FOR to match NEXT or no GOSUB to match RETURN |
| 4 | Out of DATA |
| 5 | Double Dimension--Array already Defined |
| 6 | Trying to use an Array Variable before defining it |
| 7 | Improper Variable type (String or Numeric) |
| 8 | Attempt to Dimension array beyond 2 |
| 9 | Subscript Range Error. Subscript exceeds Dimension. |
| 10 | Out of Memory. Not enough memory to create a new variable. |
| 11 | Undefined Line Number |
| 12 | Incorrect Format in a USING command |
| 13 | Out of Memory in Program or Reserve Area |
| 14 | FOR/NEXT nested too deeply or buffer space exceeded |
| 15 | GOSUB/RETURN nested too deeply or string buffer space exceeded |
| 16 | OVERFLOW/UNDERFLOW or HEX exceeds 65535 decimal |
| 17 | Mixed String and Numeric data in mathematical expression |
| 18 | Inappropriate arguments such as MID$(";hl") or COS (45,80) |
| 19 | Dimension exceeds 255 |
| 20 | When using Fixed Memory Array Variable (@), parenthesis not used |
| 21 | Variable required in expression |
| 22 | No memory available when program is loaded |
| 23 | Improper number entered for setting TIME |
| 26 | Command is not executable in current MODE |
| 27 | No program corresponding to given label: no printer |
| 28 | Command inserted inside " " or Improper INPUT or AREAD |
| 30 | Line number exceeds 65539: Error 1 if line number 65280-65539 |
| 32 | Graphic Cursor between columns 152-155 when executing INPUT |
| 36 | Data can't be specified with format given |
| 37 | Numeric calculation overflow |
| 38 | Division by zero |
| 39 | Negative Log or Negative SQR or improper trig. argument |
| CASSETTE RELATED ERRORS: | |
| 40 | Inappropriate Specification for expression |
| 41 | SAVE or LOAD specified for ROM area |
| 42 | Cassette file too large, can't be CLOADED |
| 43 | CLOAD? data format does not match file format |
| 44 | CHECKSUM error |
| PRINTER RELATED ERRORS: | |
| 70 | Pen has exceeded -2048<=X,Y<=2047 or will exceed if executed |
| 71 | Paper has or will back up more than 10.24 cm in TEXT mode |
| 72 | Inappropriate value for TAB |
| 73 | Wrong printer mode (GRAPH or TEXT) |
| 74 | Too many commas in LINE or RLINE command |
| 76 | Calculated result can't be LPRINTED on one line in TEXT mode |
| 78 | Pens being changed or LOW BATTERY state not corrected |
| 79 | COLOR signal has not been received |
| 80 | Low battery |

## ROBER MNEMONICS UPDATE

*Norlin Rober* has some additions and corrections to the material that appeared in the recent Special Edition of *PCN*. Plus, he has gleaned a lot of new knowledge. Here is a compilation of the latest findings by the Master Sleuth (on the Sharp PC-1500/Radio Shack PC-2):

1. Since the CPU does not use true indexed addressing, registers X, Y and U should more properly be referred to as Pointer Registers, not Index Registers.

2. The description of the operation of flag V contains an error. With reference to subtractions it should read ". . . is the *same* as that of the operand subtracted" rather than "is opposite to that of the operand subtracted."

3. The instruction designated INXY (opcode F5) does more than just increment X and Y. Prior to the incrementing, it transfers the contents of the address pointed to by X into the address pointed to by Y. The mnemonic should be changed to: STI (X) (Y).

4. The ADD instructions with opcodes EF, 4F, 5F and 6F are "Add *without carry*." That is, the condition of the carry flag *prior* to the addition has no effect. (The result of the addition, however, *does* affect the flags as usual.)

5. The condensed ROM map on page 4 of the Special Edition of *PCN* lists D5BF - DCAD as containing code. It should be D6BF - DCAD. Also, there is another brief lookup table located at E4E3 - E4EA.

6. Some additional instructions have been determined:

| | |
|---|---|
| PWR DOWN | (Code FD 4E) switches off the computer. |
| DSP OFF | (Code FD C0) turns off the display. |
| DSP ON | (Code FD C1) turns on the display. |

7. New information on interrupts: A maskable interrupt (usable only when flag I is set, enabling the interrupt) pushes E and P onto the stack, then transfers control to the interrupt service routine that begins at E171 (the address stored in FFF8 & FFF9). The RTI instruction at the end of the interrupt routine pops P and E from the stack.

It appears that the BREAK key does not use the interrupt routine. When this key is pressed, bit 1 of alternate memory address F00B is set to 1. This address is checked routinely by instructions in ROM.

The manual paper advance apparently produces an interrupt.

It appears that the instruction represented by opcode FD CE sets the timer to produce a "timer interrupt" after a specified length of time. This time is related to the contents of the accumulator at the time FD CE is used, but the exact connection is not clear. If the accumulator contains zero, there is no interrupt at all. In other cases, the timer produces an interrupt after a length of time that depends on the accumulator contents. This length of time seems to be about 25 milliseconds at the longest. The timer interrupt routine begins at E22C, the address obtained from ROM locations FFFA & FFFB. It will occur only if flag I is set.

The FD CE instruction would presumably be followed by a WAI instruction to produce a programmable delay lasting until the interrupt occurred.

The FD DE opcode, not used in ROM, is similar to FD CE in effect, but the resulting delay times are different.

8. New information on inputting to the CPU from the keyboard: Connections to the 64 keys (exclusive of the ON key) form the electrical equivalent of an 8 by 8 matrix. The keyboard is polled as follows: The presence of a 1 bit in a particular position in the contents of alternate memory buffer address F00C specifies the corresponding column of the matrix. The CPU instruction LDA KB loads the accumulator with the byte having a zero bit in only the position corresponding to the row (of the specified column) in which a key is pressed. The byte loaded into A is then used to determine a location in the lookup table (FE80 - FEFF), from which the appropriate ASCII code is obtained.

The ROM routine that performs this operation begins at address E42C.

9. The power-up routine begins at E000, the address stored in bytes FFFE & FFFF of ROM.

10. Although the CPU has a non-maskable interrupt capability, it apparently is not used. The circuit diagram in the Service Manual shows the NMI pin as being grounded. The interrupt routine for NMI is located at E22B, which is the address stored in FFF8 & FFF9. This routine contains the RTI instruction.

11. According to the service manual, a third kind of interrupt, designated as a "Timer Interrupt" exists. It seems likely that the address stored in FFFA & FFFB, which is E22C, is where the Timer Interrupt begins. Note that at E22C there is a short routine ending with RTI and that it contains the code FD CE.

12. The addresses following the tokens in the list of BASIC words (ROM addresses C054 to C34D) are the starting addresses of the routines that execute the associated BASIC statements. For example the BEEP routine begins at E5C1. Note that words which may *not* begin a BASIC statement, such as THEN, TO, AND and OFF, are followed by CD89. This is the address at which the routine for ERROR 1 begins.

The routines executing BASIC statements end with CALL E2, where the stack pointer is reset and the next BASIC statement is read. If an error condition is to occur, CALL E4 (for ERROR 1) or CALL E0 (other ERROR) ends the BASIC statement routine.

The tokenized words representing functions, however, call on subroutines ending with RTS.

13. Revised versions of the ROM in the PC-1500 may differ from the ROM described above, particularly in regards to exact addresses of the various addresses.

14. Thanks to *James Stutsman* for pointing out that two separate ROMs may occupy the address space 8000 to BFFF at one time. The opcode B8, which I am giving the mnemonic ROM1, selects the ROM used by the printer/cassette interface. Opcode A8 with the mnemonic ROM2 selects the other. The "other" could be the RS-232 interface promised by Sharp.

15. You can upgrade the dissassembler program to include the items discussed in this column with the following lines:

```
345    LPRINT "STI (X) (Y)";:RETURN
476    LPRINT "PWR DOWN";:RETURN
592    LPRINT "DSP OFF";:RETURN
593    LPRINT "DSP ON";:RETURN
```

16. If you would like the printout of disassembled ROM to include the addresses of subroutines called from the base page, make the following modifications to the disassembler program:

A. End line 50 with GOTO 53 (instead of GOTO 54).

B. Add line 53 to the program as shown here:

```
53     LPRINT ",";:TAB 14:C=D+65280:D=PEEK C:GOSUB 24:
       D=PEEK(C+1):GOSUB 24:D=PEEK A
```

*[Norlin indicates that he is getting quite a bit of mail. Please remember when writing to him or any author, that it is always a nice courtesy to include a S.A.S.E. (self-addressed, stamped envelope) if you would like a reply. – N.W.]*

## PC-1500/PC-2 RENUMBERING PROGRAM

*Milt Sherwin, 8602D E. Amherst Drive, Denver, CO 80231*, provides this sought after utility. Space limitations in this issue restrict us to the presentation of brief operating instructions and the program listing. However, Milt has prepared a nice article explaining how the program works. If you are interested, let *PCN* know and we will try to present the details in an upcoming issue.

You need at least a 4K memory module in your PC to use this program. Use the program as follows:

1. Clear memory then load the program.

2. If you wish to renumber a previously developed program, then MERGE it into memory at this point, *before* trying to run the renumbering program. If you are going to develop (key in) a program that you may decide to renumber later, then immediately execute the renumbering program by going to the RUN mode and issuing a RUN command. This operation simulates a MERGE and protects the renumbering program from itself!

3. Develop your program. Assign a label to the start of the program under construction or modification so that you can test it by reference

to the label. This is necessary because the renumbering program simulates the MERGE operation. You will not be able to access your new program if you do not give it a label.

4. Anytime you want to renumber the program under development, go to the RUN mode and issue a RUN command. This activates the renumbering program. Respond to the prompt: ST,IC,EN,LN with the appropriate values as indicated here:
ST = Starting line number (defaults to first line of user's program).
IC = Line increment (defaults to an increment value of 10).
EN = Ending line number (defaults to last line of user's program).
LN = Initial line number assignment for renumbering (default is 100).
Use a comma after inputting each value or if you want to use the default value. (Thus, entering three commas [, , ,] would result in the renumber program using all its default values.)

5. At the end of each RUNning of the renumbering program, it will ask if it should be "unlinked." Answer Y for yes if you no longer wish to use the renumbering program. Answer N for no if you plan further development work. Repeat step 4 as necessary.

6. Once the renumbering program has been unlinked, you may LIST or CSAVE the user's program in a normal manner. The renumbering program is no longer accessible.

7. When you are all through, use NEW0 to normalize your PC. (The renumbering program accesses various pointers at the machine language level. NEW0 assures that all pointers are re-initialized.)

```
100:IF PEEK 30821<
    >PEEK 30825AND
    PEEK 30822<>
    PEEK 30826THEN
    150
110:IF PEEK 30824+
    1=256THEN POKE
    30826,PEEK 308
    24-255:POKE 30
    825,PEEK 30823
    +1:GOTO 130
120:POKE 30826,
    PEEK 30824+1:
    POKE 30825,
    PEEK 30823
130:POKE 30823,
    PEEK 30825:
    POKE 30824,
    PEEK 30826
140:POKE PEEK 3082
    3*256+PEEK 308
    24,255:GOTO 52
    0
150:CLEAR :DIM A$(
    0)*26:C=1:P=&7
    050:R=&7150:M=
    PEEK 30825*256
    +PEEK 30826
160:INPUT "ST,IC,E
    N,LN ";A$(0)
170:FOR J=1TO LEN
    A$(0)
180:C$=MID$ (A$(0)
    ,J,1)
190:IF C$=","THEN
    LET C=C+1:GOTO
    250
200:ON CGOTO 210,2
    20,230,240
210:A$=A$+C$:GOTO
    250
220:B$=B$+C$:GOTO
    250
230:D$=D$+C$:GOTO
    250
240:E$=E$+C$
250:NEXT J
260:S=VAL A$:F=0:N
    =100
270:I=VAL B$:IF I=
    0THEN LET I=10
280:E=VAL D$:IF E=
    0THEN LET E=65
    279
290:L=VAL E$:IF L<
    >0THEN LET F=
    INT (L/256):N=
    L-F*256
300:PAUSE "WORKING
    . . ."
310:IF S=0THEN 370
320:GOSUB 530:IF B
    =STHEN 350
330:IF PEEK (M+C+3
    )=255THEN 520
340:M=M+C+3:GOTO 3
    20
350:IF L=0THEN LET
    F=A:N=D
360:GOTO 380
370:GOSUB 530
380:POKE M,F:POKE
    (M+1),N
390:POKE P,A:POKE
    (P+1),D:P=P+2:
    POKE R,F:POKE
    (R+1),N:R=R+2
400:IF PEEK (M+C+3
    )=255THEN 460
410:IF B=ETHEN 450
420:N=N+I:IF N<256
    THEN 440
430:N=N-256:F=F+1
440:M=M+C+3:GOTO 3
    70
450:M=M+C+3:GOSUB
    530:IF PEEK (M
    +C+3)<>255THEN
    450
460:P=P-&7050:N=M:
    M=PEEK 30825*2
    56+PEEK 30826
470:C=PEEK (M+2):
    GOSUB 570
480:M=M+C+3:Z=Z-1:
    IF Z<>0THEN 47
    0
490:INPUT "UNLINK
    RENUM? ";A$
500:IF LEFT$ (A$,1
    )<>"Y"THEN 520
510:POKE 30821,
    PEEK 30825:
    POKE 30822,
    PEEK 30826
520:END
530:C=PEEK M:D=INT
    (C/16):B=D*409
    6+(C-D*16)*256
    :A=C
540:C=PEEK (M+1):D
    =INT (C/16):B=
    B+D*16+(C-D*16
    ):D=C
550:C=PEEK (M+2):Z
    =Z+1
560:RETURN
570:A$="":J=M+3:K=
    0
580:A=PEEK J:G=
    PEEK (J+1)
590:IF A=&0DTHEN 7
    80
600:IF A=&F1AND G=
    &920R G=&940R
    G=&AEOR G=&AB
    THEN LET J=J+1
    :GOTO 620
610:J=J+1:GOTO 580
620:J=J+1:A=PEEK J
630:IF A<&300R A>&
    39THEN 660
640:IF K=0THEN LET
    K=J
650:A$=A$+CHR$ A:
    GOTO 620
660:L=LEN A$:IF L=
    0THEN 580
670:D=VAL A$
680:FOR X=0TO P
    STEP 2
690:E=PEEK (&7050+
    X)*256+PEEK (&
    7050+X+1):IF E
    =DTHEN 720
700:NEXT X
710:GOTO 760
720:D=PEEK (&7150+
    X)*256+PEEK (&
    7150+X+1)
730:D$=STR$ D:R=
    LEN D$:IF R-L>
    0GOSUB 790:
    GOTO 760
740:IF R-L<0GOSUB
    830:GOTO 760
750:GOSUB 880
760:A$="":K=0:IF A
    =&2CTHEN 620
770:GOTO 580
780:RETURN
790:I=PEEK (30823)
    *256+PEEK 3082
    4:H=I
800:POKE (I+R-L),
    PEEK I
810:I=I-1:IF I>K
    THEN 800
820:GOTO 860
830:I=K:H=PEEK (30
    823)*256+PEEK
    30824
840:POKE I,PEEK (I
    -R+L)
850:I=I+1:IF I<H
    THEN 840
860:C=C+R-L:POKE (
    M+2),C:J=J+R-L
870:H=H+R-L:POKE 3
    0823,INT (H/25
    6):POKE 30824,
    H-INT (H/256)*
    256
880:FOR X=1TO R
890:A$=MID$ (D$,X,
    1):POKE K,ASC
    A$:K=K+1
900:NEXT X
910:RETURN
STATUS 1
                1935
```

## RENUMBERING PROGRAM REVISITED

It soon became apparent from the mail received after Issue 18 of *PCN*, that many readers could use an explanation of the renumbering program provided by *Milt Sherwin, 8602D East Amherst Drive, Denver, CO 80231.* Here is how Milt describes its development and operation:

### Program Concepts and Development

In developing a renumbering program, it is necessary to know the format of the BASIC lines (line number storage, linkage arrangements, BASIC text representation, etc.). Once this information is obtained, the choices revolve around how sophisticated the program should be, e.g. (1) how much flexibility will the user be provided in choosing the parameters, (2) does the program renumber line numbers following GOTOs, GOSUBs, THENs, and REMs, and (3) how easy the program is to use in general. By far the most difficult item is (2), the renumbering of GOTOs, GOSUBs, THENs and REMs. Of course, this directly influences item (3), how easy the renumbering program is to use!

BASIC program storage begins at hexadecimal address &38C5 (which is decimal 14533) in a PC-1500 equipped with an 8K model CE-155 RAM module. (The address is &40C5 if there is no memory module or only a 4K module is installed.)

BASIC lines are stored in memory with the following format:

| 1 byte | 1 byte | 1 byte | N bytes | 1 byte | 1 byte |
|---|---|---|---|---|---|
| Line Number (high) | Line Number (low) | Pointer | BASIC text | Carriage Return "&0D" | * |

The byte denoted by an asterisk here contains the high byte of the next line number or &FF if this is the last line of the program. This works OK since the highest line number allowed by the interpreter is 65279 (hexadecimal $FEFF).

Line numbers at the beginning of statement lines are stored as two bytes (high, low). Thus, line 10 decimal equals bytes 00 0A in hexadecimal. Likewise, 65279 converts to FE FF.

The value stored in the byte marked "pointer" indicates how many bytes to the line-ending carriage return (taken from the pointer location itself).

BASIC lines are made up of BASIC tokens and ASCII coded characters. The code for carriage return indicates the end of a BASIC line. Thus, the line "300 GOTO 90" would be stored in the PC-1500 as:

01 2C 05 F1 92 39 30 0D

Knowing this format enables a programmer to "step" through a program, line-by-line, using PEEKs and a few program loops.

Once the program starting point and the format of BASIC lines are known, it is easy to find the line number locations and appropriate tokens (for GOTO, GOSUB, THEN or REM) that would be associated with renumbering. It is also possible to determine the line numbers that follow those tokens, using the string-manipulating functions CHR$, LEN and VAL, etc. However, the line numbers following tokens are stored as ASCII digits, and line numbers such as 5, 50, 500 and 5000 *all require a different number of bytes for storage* (1, 2, 3 and 4 respectively). A comprehensive renumbering program must take this into consideration. The plot, thus, thickens! If a program to be renumbered contains a line such as "300 GOTO 90" and renumbering causes it to become "310 GOTO 100", then an additional byte is needed because the number following the GOTO statement has now expanded to three digits! That means everything stored after that location (perhaps an entire program!) must be relocated by one byte to create room for the new digit. This is not an easy undertaking. A similar problem occurs in the opposite direction when a referenced number decreases, such as when "310 GOTO 100" becomes "300 GOTO 90". Now one byte must be removed and everything above that point moved "down" appropriately. One more little point: If the line length changes because digit(s) are added/subtracted, then the line length "pointer" must also be altered to reflect the new line length!

Note that these problems do not occur with statement line numbers. They are always stored in hexadecimal, two-byte notation. Hence, statement line numbers are easily modified without encountering expansion/contraction problems.

There is another aspect of program renumbering which must be considered during program design. The relationship of the original line numbers to the renumbered line numbers must be kept track of in order to renumber the line numbers following the GOTOs, GOSUBs, THENs and REMs. The location of this line number relationship table could be critical to the program's operation. During the operation of the renumbering program, the variables storage area could be disturbed if the program size changes due to the altering of line numbers following tokens. One way to avoid this potential problem is to POKE (store) and PEEK (read) numbers directly into "safe" areas in memory. I selected two areas which should be OK for this purpose. Namely, the fixed variable storage areas for character variables E$ through O$ (at addresses &7050 through &70FF) and P$ through Z$ (at addresses &7150 through &71FF). None of these character variables are used by the main part of the renumbering program. *[And, it makes no difference if these areas are used by the program being renumbered. After all, you cannot execute the program being renumbered at the same time that the renumbering is occurring! — N.W.]* Thus, this area is effectively protected *whenever it is being used for storage of the number relationship table.*

The first area, &7050 through &70FF is used to hold the original line numbers. The 176 bytes here can store 88 line numbers. The second area, &7150 through &71FF can also hold 88 line numbers too. So, that is where the renumbered line values are held.

If you have more than 88 lines in a program, then renumber it in sections! *[Or, use the information provided by Norlin Rober in this issue to relocate the line number relationship tables to a larger, protected area of memory. After all, now we know how to set aside as much memory as we want for special purposes! — N.W.]*

Next, there is the problem of "where does the renumbering program itself reside in memory?" If it is made a part of the main program (numbered so that it resides at the end of the program being developed), then the dynamics of the changing line lengths in the program being renumbered will effect the renumbering program as it tries to operate. That won't work.

A separate renumbering program could be written which could be MERGEd with the program under development. Would this approach work? The MERGE command is indeed interesting. It allows a user to have more than one program in memory at one time. If each program uses alphabetical labels, it can be accessed and executed. However, only the last program MERGEd can actually be modified by the system's editor. Furthermore, no selectivity is provided if a program is LLISTed or CSAVEd. Everything in memory is simply listed or saved. Furthermore, in order to develop a new program one must initially MERGE a program from tape. You cannot just start typing in line numbers and BASIC statements. And, even if the renumbering program is MERGEd after program development and is used to renumber the program already in memory, there is still a problem. While the renumbering program could then be removed line-by-line, the program under development would still not be accessible to the editor. This is because the &FF byte which terminated the MERGEd renumbering program would still be in memory following the &FF byte that terminated the original program under development. That is, there would be two consecutive &FF bytes in memory which would prohibit alterations being made by the editor. Of course, the program could be CSAVEd and then CLOADed. But, that takes a lot of time and is, frankly, cumbersome. But, is there light at the end of the MERGE tunnel?

What if the renumbering program were loaded first? Then, development work could take place in memory beyond the renumbering program. The program under development could also be expanded or contracted without disturbing the renumbering program. The renumbering program would not have to be MERGEd for each use or taken out to continue the development process. But, what about overcoming the previously mentioned problems with MERGE?

There happen to be three pointers (taking a total of six bytes, each has a high and low part) which indicate the memory address of what I call the "start of BASIC" (SOB), the "end of BASIC" (EOB) and the "start of program" (SOP) locations.

| 30821 | 30822 | 30823 | 30824 | 30825 | 30826 |
|---|---|---|---|---|---|
| Start of BASIC (SOB) | | End of BASIC (EOB) | | Start of Program (SOP) | |
| High | Low | High | Low | High | Low |
| &7865 | &7866 | &7867 | &7868 | &7869 | &786A |

In the non-MERGE mode of operation, pointers SOB and SOP are the same. When there is a MERGE, pointer SOP contains the starting address of the MERGEd program. This is the reason that additions, changes and deletions only work on the last program MERGEd. Pointer EOB points to the &FF byte of the last program in memory. A &FF byte also terminates every program that is MERGEd into memory. The SOB pointer normally remains fixed at &38C5 (14533 decimal) in a PC-1500 containing an 8K memory module. Would it be possible for a renumbering program to modify these pointers?

Yes indeed! And, that is just what my renumbering program does. The methodology is as follows:

1) The renumbering program is CLOADed into memory.

2) A RUN command causes the renumbering program to alter the SOP and EOP to simulate the performance of a MERGE.

3) Program development may then take place. The program being developed should be given a label for use when testing. (Thus, one can say LIST "A" or RUN "A" as appropriate.)

4) Subsequent RUN directives cause the renumbering program to renumber the program that is under development.

5) At the end of a renumbering pass, the renumbering program asks whether it should be "unlinked." If so, the program modifies SOB so that it equals SOP, thereby effectively removing the renumbering program from further access.

6) Once the renumbering program has been unlinked, the program under development can be LISTed, CSAVEd or executed using normal methods.

7) Execution of the command NEW0 restores all three pointers (SOB, EOB and SOP) to their pristine states.

Furthermore, if a previously developed program needs to be renumbered, it can simply be MERGEd in place of step 2 above. Doing so causes SOP and EOB to be set appropriately by the MERGE operation instead of by the renumbering program.

For reference purposes, the behavior of the SOB, EOB and SOP pointers are summarized here:

| | |
|---|---|
| DURING REGULAR OPERATIONS | |
| NEW0/START | SOB = EOB = SOP = 14533 |
| PROGRAMMING/CLOAD | SOB = SOP = 14533, EOB > 14533 |
| MERGE | SOB = 14533, SOP > 14533, EOB > SOP |
| DURING RENUMBERING OPERATIONS | |
| NEW0/START | SOB = EOB = SOP = 14533 |
| CLOAD RENUM | SOB = SOP, EOB > 14533 |
| INITIAL RUN/MERGE | SOB = 14533, SOP = EOB > SOB |
| DEVELOPMENT | SOB = 14533, SOP > SOB, EOB > SOP |
| UNLINK | SOB = SOP(Development), EOB > SOP |
| NEW0 | SOB = EOB = SOP = 14533 |

(Value of 14533 is for a PC-1500 with an 8K memory module installed.)

**Operation of the Renumbering Program**

Now that you are familiar with the concepts behind the design of the program, the actual implementation of the program can be described. Please remember that the program only renumbers statement line numbers and line numbers referenced by GOTO, GOSUB, THEN and REM statements. *It will not recalculate line numbers that are obtained by manipulating variable names or performing similar types of programming tricks!*

Lines 100 – 140: The values of SOB and SOP are compared to determine if they are equal, indicating that a MERGE has not been performed. If equal, a MERGE is simulated, program execution terminates and the user can undertake program development. If SOB and SOP are not equal, then the program assumes that program development has taken place (either through an actual tape MERGE or a previous execution of this program).

Lines 150 – 250: Variables are initialized and renumbering parameters are requested from the operator. There are four parameters used by the program:

| PARAMETER | DEFINITION | DEFAULT VALUE |
|---|---|---|
| ST | Starting line number | First line of the development program |
| IC | Line increment | 10 |
| EN | Ending line number | Last line of the development program |
| LN | Initial line number when program renumbered | 100 |

Commas must be used to separate the parameters during input *even if values are not entered for some parameters.* Thus, inputting ",,, 200" will result in the current development program being renumbered in its entirety, with line increments of 10, and the first line of the renumbered program being 200. However, if the RETURN key is pressed without entering any parameter values or commas, then default values will be used to renumber the program. The parameters are input as a single character variable, then split into four individual variables for use within the program.

Lines 260 – 300: The variables used to store the parameters are set for operation at their default values or the values inputted by the user.

Lines 310 – 360: The starting location is found.

Lines 370 – 450: Line numbers are renumbered. The original line number/renumbered line number table is built. The subroutine at line 530 is used to determine the original line numbers.

Lines 460 – 480: Variables are set for renumbering line numbers that follow GOTOs, GOSUBs, THENs and REMs. Each line is searched for the corresponding tokens using the subroutine at line 570.

Lines 490 – 520: The operator is provided the option of unlinking the renumbering program. If unlinking is to occur, SOB is set equal to SOP. If not, the program is terminated. Further program development may then take place.

Lines 530 – 560: Subroutine to determine a line number. Line numbers, stored as two bytes, are converted to single line number values.

Lines 570 – 610: First part of the subroutine that searches for GOTO, GOSUE, THEN and REM tokens. (Renumbering line numbers after a REM only works if the line number follows immediately after the REM.) If no appropriate token is found, detection of the code for carriage return in line 590 causes the subroutine to terminate (via line 780).

Lines 620 – 650: Second part of the token-searching subroutine. If an appropriate token is found, the ASCII representation of the associated line number(s) is(are) located. As they are found, they are converted so they may be stored in a character variable for later use.

Lines 660 – 710: Part three of the subroutine evaluates the line number stored as a character variable. The previously created line number table is used to determine the referenced renumbered line number.

Lines 720 – 780: The fourth part tests to determine if the renumbered line number has the same number of digits as the original line number. If the renumbered line number has more digits than the original line number, the subroutine starting at line 790 is used to create more space. If the new number has less digits, the subroutine at line 830 is used to remove the excess space. If the line numbers have the same number of digits, the renumbered line number is placed in position by the subroutine at line 860. If no match is found in the table, no renumbered line number is required (as the original line number has not changed). The variables are then reset. The search for more line numbers or tokens resumes.

Lines 790 – 820: Subroutine to create additional space in a program.

Lines 830 – 870: Subroutine to remove excess space in a program.

Lines 880 – 910: Subroutine to place a renumbered line number into memory.

*[OK, all you machine language programmers. Here is an opportunity for you to really have some fun. Milt has described in detail how his renumbering program operates. Machine-language speed would be nice in this application. Who is going to tackle the project? – N.W.]*

## MEMO PRINTER PROGRAM

Here is a program for Radio Shack PC-2 and Sharp PC-1500 users who want to keep notes or text in memory. This program was created by: *Brian Peterson, 6807 N. Sheridan Road, Chicago, IL 60626.* One of the interesting features about this program is that *it prints sideways!* Thus, it accepts and prints out full 80-character lines. Brian calls his program "Memo Printer." Here is what he has to say about using it:

### It Operates In Three Modes

These modes are referred to as: Input, Edit and Print.

The Input mode allows the user to enter characters into memory. You need at least a 4K memory expansion module to use the program. With a 4K module, it can accept approximately 33 lines of text. An 8K module allows about 88 lines. Theoretically, a 16K expansion unit would allow up to about 187 lines. Each line may contain up to 80 characters.

If you want to leave a blank line between paragraphs, just enter a single 'space' character on the line. To get out of the Input mode, press the ENTER key without putting any characters in the line.

### Editing

You use the Edit mode to go back and change text that was originally entered while in the Input mode. The Edit mode allows you to examine the text buffer and make alterations. You can insert characters and entire lines, if desired.

Several keys have different functions than normal when in the Edit mode. The DEF key is used to activate the ability to insert or delete lines. After the DEF key is pressed, the PC will ask whether the user wishes to insert or delete. Enter 'I' to insert. Enter 'D' to delete. When in the delete phase, the CL key deletes individual characters. Holding the key down will cause it to repeat and delete a series of characters. The MODE key enables the user to insert characters at the current cursor position. When this insert function is in effect, the symbol > will appear in its inverse form (filled-in) on the screen. The key normally used to select the desired Reserve mode (◇) is used to append lines to the end of the text. The scroll line up (^) key and the scroll line down (v) key are used to examine lines above or below the current text line. The symbols <,,>, ?, :, ; and comma are available through the use of the SHIFT key. Use the SML key to enter lower case characters. Tip: When entering text, put the unit in the lower case mode and use the SHIFT key when a capital letter is needed.

The six softkeys can be customized to allow the use of characters not usually available on the keyboard. Program lines 247 – 252 can be altered for this purpose.

Press the keys firmly when using the Edit mode to allow time for the character to be recognized. Most keys will repeat if held down in this mode.

To exit the Edit mode, press the RCL key.

### Printing

This program prints "lengthwise" on the paper. Options for selecting the pen color as well as single- or double-spacing are provided.

### Use GOTO

Once you have text in memory, always enter the program using a GOTO directive, not RUN. Using RUN will clear out any text that you have placed in memory.

### There Is No Stopping

This program disables the ON/BREAK key during operation, except when printing.

*[While this last feature is interesting, I am not convinced that it is desirable. I was recently using the program while away from the office, got into the Edit mode, and forgot how to get out. With the BREAK key disabled, I had no choice but to leave the unit on until I got back to the office, as I did not want to reset the PC and lose everything that I had in the unit. I don't think the risk of accidently striking this key, nor the possible consequences of doing so, are worth not being able to shut the unit off if I forget how to operate the program. In any event, a word to the wise: Keep the operating instructions handy until you have memorized all of the options available! – N.W.]*

Program *Memo Printer (Part 1)*

```
  5:ARUN :WAIT 0:
    POKE# &F00D,
    PEEK# &F00DOR
    128
 10:IF STATUS 3=&5
    800DIM P1$(0)*
    80,P2$(0)*80,A
    $(32)*80:N=33
 11:IF STATUS 3=&6
    000DIM P1$(0)*
    80,P2$(0)*80,A
    $(87)*80:N=88
 12:IF STATUS 3=&6
    400DIM P1$(0)*
    80,P2$(0)*80,A
    $(186)*80:N=18
    7
 13:POKE STATUS 3-
    23,72,118,74,0
    ,5,189,255,65,
    78,78,153,8
 14:POKE STATUS 3-
    11,76,119,139,
    6,72,119,74,0,
    158,18,154
 15:PRINT "Input
       Edit    Print
     End"
 16:V=ASC INKEY$ :
    IF VCALL
    STATUS 3-23
 17:IF VAND ASC
    INKEY$ THEN 17
 18:ON V-16GOTO 20
    ,16,200,16,40,
    600
 19:GOTO 15
 20:LOCK :L=0:
    USING
 25:ON ERROR GOTO
    0
 30:PRINT "Line";L
    ;">";:INPUT ""
    ;A$(L):CLS :L=
    L+1:IF L<NGOTO
    30
 35:M=L-1:UNLOCK :
    BEEP 1:CLS :
    GOTO 15
 40:PRINT " BLK BL
    U GRN RED  -CO
    LOR-"
 41:V=ASC INKEY$ :
    IF VCALL
    STATUS 3-23
 42:IF VAND ASC
    INKEY$ THEN 42
 43:C=V-17:IF V<17
    OR V>20GOTO 40
 44:PRINT " SNG     
      DBL      -SPA
    CING-"
 45:V=ASC INKEY$ :
    IF VCALL
    STATUS 3-23
 46:IF VAND ASC
    INKEY$ THEN 46
 47:IF V<>17AND V<
    >19GOTO 44
 48:S=(V=17)+2*(V=
    19):T=11*(V=17
    )+6*(V=19)
 49:CLS :GRAPH :
    LINE -(220,0),
    0,C:TEXT :LF 5
 50:GRAPH :SORGN :
    ROTATE 1:CSIZE
    2
 52:POKE# &F00D,
    PEEK# &F00DAND
    127
 55:G=0
 60:FOR X=1TO 61
    STEP 20
 70:FOR Y=T-1TO 0
    STEP -1
 75:X$=MID$ (A$(Y+
    G),X,10)
 80:IF X$=""GOTO 1
    10
 90:GLCURSOR (200-
    19*Y*S,-X*12)
100:LPRINT X$
110:NEXT Y
120:FOR Y=0TO T-1
125:X$=MID$ (A$(Y+
    G),X+10,10)
130:IF X$=""GOTO 1
    60
140:GLCURSOR (200-
    19*Y*S,-(X+10)
    *12)
150:LPRINT X$
160:NEXT Y
170:NEXT X
```

program continued on next page

**Program** *Memo Printer (Part 2)*

```
175:G=G+T:IF M>=G
    TEXT :LF 5:
    GRAPH :SORGN :
    ROTATE 1:GOTO
    60
180:TEXT :LF 5:
    BEEP 1:POKE# &
    F00D,PEEK# &F0
    0DOR 128:GOTO
    15
200:ON ERROR GOTO
    350:L=0
210:CLS :WAIT 0:P=
    1:Q=1:R=5:IF D
    LET L=L-(L>0)
220:USING "####":
    GOSUB 580
221:Z=0:E=0:F=0:
    GOTO 224
223:Q=1:R=P+4:IF P
    >21LET Q=P-20:
    R=25
224:PRINT L;">";
    MID$ (A$(L),Q,
    21):IF ZCURSOR
    4:GPRINT 0;127
    ;62;28
225:Y=0:CURSOR R:
    PRINT CHR$ 127
226:K$=INKEY$ :V=
    ASC K$:IF V>32
    GOTO 270
227:IF VGOTO 230+V
228:X=X+1:IF X<6
    GOTO 226
229:X=0:Y=(Y=0):IF
    YCURSOR R:
    PRINT CHR$ 127
    :GOTO 226
230:CURSOR R:PRINT
    MID$ (A$(L),P,
    1):GOTO 226
231:W=2:GOSUB 550:
    E=(2=(2AND
    PEEK &764E)):
    GOTO 225
232:W=8:GOSUB 550:
    F=(8=(8AND
    PEEK &764E)):
    GOTO 225
238:P=P-(P<>1):
    GOTO 223
239:M=M+(M<N-1):L=
    M:GOSUB 500
240:L=L+(L<M):P=1:
    Q=1:R=5:GOTO 2
    21
241:L=L-(L<>0):P=1
    :Q=1:R=5:GOTO
    221
242:P=P+(P<=LEN A$
    (L)):GOTO 223
243:P=1:Q=1:R=5:
    GOSUB 580:
    GOSUB 500:GOTO
    221
247:K$="!":GOTO 27
    2
248:K$=",":GOTO 27
    2
249:K$="#":GOTO 27
    2
250:K$="$":GOTO 27
    2
251:K$="%":GOTO 27
    2
252:K$="&":GOTO 27
    2
254:P1$(0)=LEFT$ (
    A$(L),P-1):P2$
    (0)=MID$ (A$(L
    ),P+1,80-P):A$
    (L)=P1$(0)+P2$
    (0):GOTO 223
255:GOSUB 500:
    GOSUB 580:L=M:
    GOTO 15
257:W=128:GOSUB 55
    0:CLS :GOTO 40
    0
261:Z=(Z=0):GOSUB
    500:GOTO 224
262:GOTO 272
270:IF E+F=1IF V>6
    4LET K$=CHR$ (
    VOR 32)
271:IF V>39IF V<48
    IF E=1GOSUB V-
    39+600
272:POKE &764E,253
    AND PEEK &764E
    :E=0
273:IF P>80GOTO 35
    0
274:IF Z=0POKE (
    STATUS 3+6+P+8
    0*L),ASC K$:P=
    P+1:GOTO 223
275:P1$(0)=LEFT$ (
    A$(L),P-1)
280:P2$(0)=MID$ (A
    $(L),P+(Z=0),8
    0-P)
290:A$(L)=P1$(0)+K
    $+P2$(0)
300:P=P+1:GOTO 223
350:BEEP 1:GOTO 22
    5
400:PRINT "Insert
    or Delete?":
    POKE &764E,127
    AND PEEK &764E
401:V=ASC INKEY$ :
    IF VCALL
    STATUS 3-23
402:IF VAND ASC
    INKEY$ THEN 40
    2
403:IF V<>68AND V<
    >73THEN 400
410:IF V=68INPUT "
    Delete line#";
    D:FOR X=DTO M:
    A$(X)=A$(X+1):
    NEXT X:A$(M)="
    ":M=M-1:L=D:
    GOTO 210
420:INPUT "Insert
    line after lin
    e#";I:I=I+1
430:FOR X=M-(M=N-1
    )TO ISTEP -1:A
    $(X+1)=A$(X):
    NEXT X:A$(I)="
    ":M=M+1:L=I+1:
    GOTO 210
500:IF ASC INKEY$
    THEN 500
510:RETURN
550:U=PEEK &764E:
    IF UAND WPOKE
    &764E,U-W:
    GOSUB 500:
    RETURN
555:POKE &764E,U+W
    :GOSUB 500:
    RETURN
580:POKE &764E,245
    AND PEEK &764E
    :E=0:F=0:
    RETURN
600:POKE# &F00D,
    PEEK# &F00DAND
    127:END
601:K$="<":RETURN
602:K$=">":RETURN
603:K$=":":RETURN
604:K$=";":RETURN
606:K$=",":RETURN
608:K$="?":RETURN

STATUS 1
                  3376
```

## STOPWATCH PROGRAM FOR R. S. PC-1/SHARP PC-1211

Radio Shack PC-1 and Sharp PC-1211 users can use this program to precisely time events down to 1/10th of a second intervals!

**Timer Design**

Two conditions (rules) should be met when designing an accurate time keeping device: 1) There should be a large number of cycles ("ticks") per second, and 2) The regularity of each cycle must be as stable as possible. Both of these rules are met quite nicely by the Cesium Atomic Clock. It "ticks" at a frequency of 9,192,631,770 Hertz (cycles per second). Its regularity is within 1 part in 10,000,000,000,000. That makes it the most accurate timing device known to man.

The design rules mentioned, of course, can be applied to the design of computer "clock" programs. Unfortunately, many programmers do not follow these rules. Consequently, inaccurate (errors up to several minutes each hour) times results.

Environmental temperatures can also have an effect on accuracy. Laboratory experiments performed on the Radio Shack PC-1 and the Sharp PC-1211 indicate that program execution speed increases slightly with increasing temperature. In equation form, it was found that:

$$S(T)/S(T=0) = 2.0261E-5*(T+273.15)+0.994466$$

when T is between 0 and 40 degrees Centigrade. S(T) is the relative execution speed, T is the temperature in Celsius of the PC, and S(T=0) is the reference speed at 0.0C (defined, arbitrarily, as 1.000) in this equation. Using this equation, one can observe that at 40 degrees C., a program executes about 0.08% faster than the same program at a temperature of zero. Though these effects are relatively small, they can be significant over an extended period of time. The program provided

## EPSON'S BASIC

(continuation from previous page)

| | |
|---|---|
| ERROR n | Forces the error number n to occur, used for debugging or other purposes. |
| RESUME 0 | Resumes program execution after an ON ERROR GOTO error trap. Can specify resumption with the same line, the next line or any specified line. |
| AUTO m,n | Begins automatic line numbering at line m with an increment of n. |
| DELETE m–n | Delete program lines m through n. |
| DEF FN x(y,z) | Define a function (similar to a one-line subroutine) named x with parameters y and z or more, if desired. |
| CLEAR n | Reserve n memory locations for string space. Default is 200 bytes. |
| OPTION BASE n | Set the first array element to be at 0 or 1. |
| RANDOMIZE n | Provide a seed to the random number generator. |
| ERASE x | Erases the array x to open up memory space. |
| LINE INPUT x | Accepts a line of input, but will not stop execution if only the RETURN key is used. |
| SCROLL n | Sets up the number of lines that the display will roll in either direction when display control keys are pressed. |
| SOUND x,y | Plays sound tone n (1 -56) for duration y. |
| MON | Passes control to the machine language monitor. |
| CLS | Clears the screen. |
| WIDTH x,y | Sets up the number of columns (x) and rows (y) which may be used by the virtual screen. Only 20 by 4 lines may be viewed at one time. Default value is 40 by 8. |
| COLOR n | Selects color from 0 to 7. (Plotter, TV?) |
| LOGIN x | Places you into workspace number 1 through 5. Each program workspace is separate, with its own line numbers. Like Casio's P0 through P9. |
| TITLE x | Gives the current workspace a title, Prevents the space from being accidently erased. Title appears on the main menu. |
| FRE(x) | Gives the number of bytes remaining in memory. |
| POS(x) | Yields the current column position. |
| EOF(x) | Provides the end-of-file status for file x. |
| CINT | Convert numbers from other forms to INTeger, |
| CSNG | SINGle-precision or |
| CDBL | DouBLe-precision format. |
| FIX(x) | Returns the fixed-point portion of the number x. Is not the same as INT(x) when dealing with negative numbers. |
| SPACE$(n) | Provides n spaces. |
| HEX$ | Converts numbers to hexadecimal notation. |
| OCT$ | Converts numbers to octal notation. |
| MID$ | Can be used on the left side of an equation to change the characters in a string at specified locations. |
| INSTR(x,y) | Locates where substring y begins in string x. |
| VARPTR x | Shows where the variable x is located in memory. |
| TIME$ | Gives the time as hh:mm:ss |
| DATE$ | Gives the date as mm/dd/yy |
| DAY | Gives the day of the week, Sunday = 0 through Saturday = 6. |
| INPUT$(x) | Waits for the receipt of x characters. Does not need RETURN. |
| POINT(x,y) | Give the status of the dot at position x,y. |
| PSET(x,y) | Turn on the dot at position x,y. |
| PRESET(x,y) | Turn off the dot at position x,y. |
| SCREEN n | Select screen 0 or 1. |
| COPY | Puts a copy of the screen on the printer. |
| TAPCNT | Displays the microcassette tape counter value. |

## UNDERSTANDING THE PC-1500

This is the fourth article in a series being presented by *Norlin Rober, 407 North 1st Avenue, Marshalltown, IA 50158.* Unless otherwise noted, the information presented by Norlin in this series also applies to the Radio Shack PC-2 unit.

Since it was first introduced, the Sharp PC-1500 has undergone a number of changes. Most of these are nearly unnoticeable to the average user. Many of these changes have been to correct minor errors. A few of the improvements, however, are rather significant.

The integrated circuit containing the ROM chip is stamped with one of the following identifying numbers:

| | |
|---|---|
| A01 | Original version. |
| A03 | First revision. Also used in Radio Shack's PC-2. |
| A04 | Latest version. |

Apparently, there is no version A02 in widespread use.

You can readily identify the version in your machine (without physically opening your unit) by peeking at a few locations and using the following information:

| Version | PEEK &C443 | PEEK &C5BD |
|---|---|---|
| A01 | 56 | 129 |
| A03 | 59 | 129 |
| A04 | 59 | 74 |

I have identified many of the effects (but not all) of the modifications that have been made by various revisions to the ROM. Here are comments regarding these alterations.

### Minor Improvements

1.) In version A01, when program execution is begun automatically using an ARUN statement, the BUSY signal in the display fails to turn on. This was corrected in version A03.

2.) In the original version, a program line may not consist of only a label. If you attempt to use such a line, ERROR 21 will result whenever the line is reached during program execution. A label alone may be used as a program line when ROM versions A03 and A04 are installed.

3.) In version A01, a statement containing USING, followed by the name of a numeric variable, will produce either a fatal crash or an execution of the power-up routine (with NEW0? :CHECK displayed). One may demonstrate this phenomena by typing USING A, followed by ENTER. This defect was corrected in version A04.

4.) In the REServe mode, a "template" of exactly 26 characters will place a zero byte in the location immediately following the template. This cancels a previously defined REServe key or template. (This only applies to the A01 ROM version). To illustrate, do the following:

A. Set the REServe mode, clear REServe memory with NEW.

B. Set to Group III. Assign something to one of the REServe keys.

C. Enter a template containing exactly 26 characters. (Note that now the assignment made to the REServe key has been lost.)

D. Set to Group II. Enter a template of exactly 26 characters. Now the template previously set for Group III has disappeared!

This defect was corrected in version A03.

5.) In REServe mode, with a version A01 ROM, it is not possible to clear the phrase assigned to a REServe key by overwriting with spaces.

6.) Under certain circumstances, when the input buffer is filled, the display continues on to include the contents of &7800 and beyond. This particular quirk exists in version A01 and in at least some PCs with version A03. In at least some cases, it does not occur with version A04. To demonstrate the problem, clear the display, then fill it with the token for RETURNs by repeatedly keying in DEF Y. When the end of the input buffer has been reached, there will probably be something unexpected appearing at the right end of the display. (If nothing happens, try POKE &7800, 87, 72, 65, 84, 63 and then repeat the above.)

7.) Another oddity, occurring only in version A01, can be observed by doing the following:

A. Enter the program line: 10 PRINT "ANYTHING"

B. In the RUN mode, key the following:

```
RUN
ENTER
↑
```

ENTER
Up arrow
Cursor left

Note that the program line has been placed into the display, without the usual colon following the line number.

8.) Here is another quirk involving the display that appears in version A01 only. Enter the program line:

```
10 INPUT "A";A
```

Now RUN the program. In response to the prompt, enter just a plus sign (rather than a legal input). The display will show: ERROR 1. Note that using the CL key will not completely remove the error message.

9.) Enter the program line:

```
10 GCURSOR 152:INPUT A
```

When the program is executed, the display will show ERROR 32 IN 10, as expected. However, in version A01, the use of CL or Up arrow will not clear the error message. In version A03, the error message is cleared, but the computer continues to wait for input, with a partial question mark displayed at the right end of the LCD. (Version A04 works fine.)

**Major Improvements**

1.) In version A01 only, it is possible to completely mess up a program when variable storage in main memory occupies exactly one more byte than for which there is room. (Thanks to *Stan Kameny* for noting this problem.) What happens is that the "stop byte" following the last program line is overwritten by the variable. To illustrate the defect:

A. Clear using NEW0 and enter the program line:
```
10 ABCDEFGH
```
B. In RUN mode, execute one of the following, depending on how much RAM expansion you have installed in your PC:
No expansion: DIM A(228)
4K expansion: DIM A(2, 246)
8K expansion: DIM A(6, 178)

C. Now set PRO mode and LIST the program. Use the Down arrow to observe that the stop byte has been lost. Entering more program lines now will cause additional confusion, extending to the contents of REServe memory. Enter the program line:
```
60000 ABCDEFGH
```
D. Note the effect on REServe memory.

2.) In versions A01 and A03, the use of Boolean operators will produce incorrect results when inputs are negative numbers that are represented internally in binary form. Since ordinarily these Boolean operators are used with positive inputs only, the errors are of less consequence than it might seem at first. To illustrate:

A. NOT NOT 1 returns -31233. The correct result is 1.
B. 1 OR (-1 OR 1) returns 31233. The correct result is -1.
C. 1 AND (-1 OR 1) returns 0. The correct result is 1.

In version A04, none of these errors occur.

3.) In the RADIAN and GRAD modes (only), certain input arguments used with trigonometric functions produce incorrect results. This bug was corrected in version A04. In RADIAN mode, any input whose mantissa digits are 174532925199 will be treated as zero. In GRAD mode the same applies to mantissas having digits: 111111111111. To illustrate:

A. In RADIAN mode, SIN($\pi$/18) returns 0. The correct result is .1736481777.
B. In GRAD mode, TAN(100/9) returns 0. The correct result is .1763269807.

**Some Changes in the Operation of BASIC**

In version A04 of the ROM, two changes were made to make Sharp BASIC more compatible with other BASICs. These changes must be taken into account if a program is to operate correctly amongst all PC-1500 and PC-2 computers.

1.) A change was made in the interpretation of a statement beginning with an IF expression. (Example: IF A THEN 50.) In versions A01 and A03, such a statement was regarded by the PC as true if the value of the expression was positive and false if zero or negative. In A04 it is considered as true if non-zero, false if zero.

2.) A change in the interpretation of FOR/NEXT loops affects the number of passes made through the loop for cases in which the final value of the index variable does not exactly equal the test value. In versions A01 and A03, when NEXT is executed the index variable is first compared with the test value. If it is less than the test value, it is incremented by the step size and the loop is repeated. In version A04, NEXT always increments the index variable and then compares the incremented result to the test value. The loop is repeated unless the new (incremented) index variable exceeds the test value. (Of course, if a negative number is being used as a step value, then this description must be modified.) The following summary should clarify the differences:

| | A01, A03 | A04 |
|---|---|---|
| A. Loop: FOR X=0 TO 3 | | |
| Values of X used in loop: | 0, 1, 2, 3 | 0, 1, 2, 3 |
| Value of X after loop finished: | 3 | 4 |
| B. Loop: FOR X=0 TO 5 STEP 2 | | |
| Values of X used in loop: | 0, 2, 4, 6 | 0, 2, 4 |
| Value of X after loop finished: | 6 | 6 |

**A Warning!**

Do not use the Boolean NOT operator in an IF statement. The use of NOT in a conditional test is not interpreted by the PC-1500 as meaning the negation of a logical statement. When Sharp chose to make the logical value of a true statement 1 and a false statement 0, they automatically ruled out the possibility of allowing NOT to mean negation. (Note that in Sharp's system, 0 and 1 may not have the same truth value. But, NOT 0 and NOT 1 are both calculated as negative numbers, thereby making NOT 0 and NOT 1 identical in truth value!) Perhaps this should be treated as a choice made by the designers, rather than as an error?

Here are a few examples you may use to convince yourself that NOT doesn't belong in an IF statement.

A. With ROM versions A01 or A03, enter the program line:
```
10 IF NOT (1=2) BEEP 1
```
Since it is not the case that 1=2, one would expect to hear a beep. There won't be one if you have ROM A01 or A03. However, the corrections made in A04 (regarding Boolean operators) affect the the use of NOT in IF statements. Thus, this example when run on a PC having ROM version A04, does produce a beep. However, the program line:
```
10 IF NOT (1=1) BEEP 1
```
does yield a beep with an A04 ROM, contrary to expectations. (There is no beep in the earlier ROMs with this statement.) *

**In Conclusion**

It is to Sharp's credit that they are making efforts to constantly upgrade the PC-1500. However, anyone considering purchasing a new unit might first wish to make certain that they are getting the latest version of the PC. Users of earlier units may take care when programming to avoid the pitfalls discussed in this article.

## HISTOGRAM PROGRAM

*Russel Doughty, 3604 Northwick Place, Bowie, MD 20716,* developed the program shown in the accompanying listing. He also provided the following comments regarding its use and application.

If you have ever used statistics, you know that they can often be misleading. Suppose you have a group of one hundred people with an average income of \$20,000 (\$20K). This could mean that they all make \$20K or that fifty make \$30K and fifty make \$10K or that ten make \$200K and the rest are unemployed!

In short, that single "average" number does not tell you much about the true distribution of incomes. It takes a look at the standard deviation, variance, minimum and maximum values to begin to get a true picture. Even then it can take a fair amount of interpretation to sort things out. But, your trusty Radio Shack PC-2 or Sharp PC-1500 and the plotting capabilities of its printer/cassette interface are a natural for creating "histograms." This is a statistical method that literally draws a picture of the data distribution.

You create a histogram by sorting your data into equally spaced "cells." Then you count the number of data points in each cell. For

* oplossing: NOT (vergelijking) +2
vb. IF NOT (1=2)+2 BEEP → "BEEP"
IF NOT (1=1)+2 BEEP → X
werkt ongeacht ROM-versie.

the income example just mentioned, you might find (using your PC to analyze the data, of course!) that the minimum and maximum incomes were $10K and $100K respectively. You might choose the width of cells, then, to be $10K. You would then sort the data into nine cells: $10–20K, $20–30k, . . . $80–90K and $90–100K. You would then count the number of data points falling within each cell and plot the results. You end up with a visual representation of the data distribution. Interpretation becomes easy.

That is essentially what the accompanying program does. You will need a memory expansion module to use the program. It is written to utilize an 8K module. I believe the program will work satisfactorily in a system having only a 4K module, provided that you reduce the array named X that is DIMensioned in line 100.

Start the program using a RUN directive. The program will ask if you need instructions. Responding "Y" for yes results in the printout of a summary of the DEFined keys that have been established to facilitate selecting various program operations.

Initially, the program prompts for data inputs. When you have

Program *Histogram*

```
100:CLEAR :H=1:DIM
    X(255):DIM F(5
    1)
108:WAIT 100:PRINT
    "          HIST
    OGRAM"
112:INPUT "NEED IN
    STRUCTIONS(Y/N
    )? ";Z$:IF Z$=
    "Y"GOSUB 2000
114:PRINT "ENTER D
    ATA"
200:"A"WAIT 5
210:PRINT "X(";H;"
    )":CURSOR 8:
    INPUT X(H):CLS
220:H=H+1:GOTO 210
500:"S"TEXT :
    LPRINT :COLOR
    0:LPRINT "STAT
    ISTICS":LPRINT
510:FOR J=1TO H:
    FOR Q=1TO H-J
530:A=X(Q):C=X(Q+1
    )
550:IF A<CTHEN 580
560:X(Q)=C:X(Q+1)=
    A
580:NEXT Q
590:NEXT J
600:Z=0:S=0:FOR I=
    2TO J:S=S+X(I)
    :NEXT I:M=S/(J
    -1)
605:FOR I=2TO J:Z=
    Z+X(I)*X(I):
    NEXT I
607:V=(Z-(J-1)*M*M
    )/(J-2):CSIZE
    1
610:LPRINT USING "
    ######.##";"DA
    TA MIN = ";X(2
    )
620:LPRINT "DATA M
    AX = ";X(J)
630:LPRINT "MEAN
        = ";M:
    LPRINT "STD DE
    V  = ";√V
640:LPRINT "VARIAN
    CE = ";V
645:LPRINT "ENTRIE
    S  = ";H-1
650:WAIT :PRINT
805:"H"INPUT "CELL
     WIDTH ";W
810:Y=INT ((X(J)-X
    (2))/W)+1
815:FOR I=0TO Y:F(
    I)=0:NEXT I
820:FOR I=0TO Y
830:FOR Q=2TO J
835:R=INT (X(2))
840:IF X(Q)>=(R+(I
    *W))AND X(Q)<(
    R+((I+1)*W))
    THEN 848
844:GOTO 850
848:F(I)=F(I)+1
850:NEXT Q:NEXT I
905:"G"TEXT :LF 2:
    GRAPH :
    GLCURSOR (30,-
    150)
910:SORGN :LINE -(
    180,170),0,1,B
920:A=0:M=0
930:FOR I=0TO Y
940:IF A<F(I)THEN
    LET A=F(I)
945:M=M+F(I)
950:NEXT I
960:D=A
970:M=INT (D/M*100
    )
 1000:A=150/D:E=14
      0/(Y+1)
 1010:GLCURSOR (6,
      0):SORGN
 1020:FOR I=0TO Y
 1030:LINE ((I*E),
      0)-(((I+1)*E
      ),(F(I)*A)),
      0,3,B
 1040:NEXT I
 1043:GLCURSOR (-6
      ,0):SORGN
 1044:FOR I=0TO 7
      STEP 2
1045:LINE (0,(10*
     D*A-I*D*A)/1
     0)-(180,(10*
     D*A-I*D*A)/1
     0),2,1:NEXT
     I
1050:TEXT :CSIZE
     1:COLOR 0:LF
     4:LPRINT
     USING "#####
     .##";INT (X(
     2));TAB 26;
     INT (X(2))+W
     *(Y+1)
1055:LF -18:
     LPRINT TAB (
     15);"HISTOGR
     AM":LF -2
1056:LPRINT "%":
     LPRINT "TOT"
     :USING "###"
     :LPRINT TAB
     (1);M
1058:FOR I=2TO 7
     STEP 2:LF 2:
     LPRINT TAB (
     1);INT ((10*
     M-I*M)/10):
     NEXT I
1060:USING "#####
     .##";LF 5:
     LPRINT TAB (
     6);"MAX FREQ
     . = ";D:
     LPRINT TAB (
     6);"CELL WID
     TH= ";W
1065:WAIT :PRINT
1070:"C"LPRINT :
     LPRINT :
     LPRINT TAB 1
     5;"CELL DATA
     "
1080:LPRINT :FOR
     I=0TO Y:
     USING "####"
     :LPRINT TAB
     (5);"CELL ";
     I+1;" = ";:
     USING "####.
     ##":LPRINT F
     (I);
1081:LPRINT INT (
     X(2)+I*W):
     NEXT I
1082:WAIT :PRINT
1085:"L"LPRINT :
     LPRINT :
     LPRINT TAB (
     12);"RAW DAT
     A LISTING":
     LPRINT
1090:FOR I=2TO H:
     USING "####"
     :LPRINT TAB
     (9);"DATA PT
     .";I-1;" = "
     ;:USING "###
     .##":LPRINT
     X(I)
1091:NEXT I
1092:WAIT :PRINT
     :GOTO 100
2000:CSIZE 1:
     LPRINT "DEF/
     S = STATISTI
     CS":LPRINT "
     DEF/H = HIST
     OGRAM"
2010:LPRINT "DEF/
     C = HISTOGRA
     M CELL DATA"
     :LPRINT "DEF
     /L = RAW DAT
     A LISTING"
2015:LPRINT "DEF/
     A = ADD MORE
      DATA"
2020:LPRINT "YOU
     CAN ENTER UP
      TO 255 DATA
      POINTS"
2030:LPRINT "YOU
     MUST USE DEF
     /S & H BEFOR
     E C OR L"
2040:LF 7:RETURN
STATUS 1        1853
```

entered a sufficient number of points, select the following options using DEFined keys:

DEF/S: Performs statistical analysis on the data and prints minimum, maximum, standard deviation and variance figures.

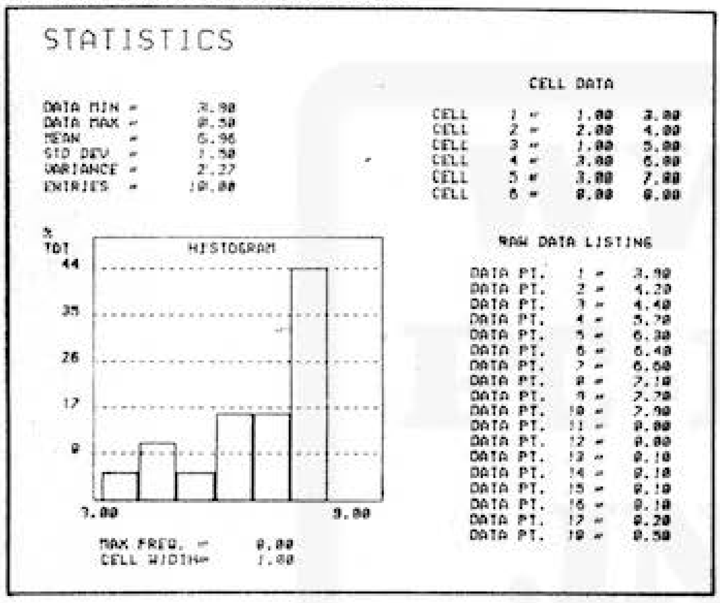
DEF/H: Draws a histogram of the data using the data previously entered and your response to the prompt for the cell width to use. The drawing is done using three pen colors. The vertical axis presents the number of data points in each cell expressed as a percentage of the total number of data points in all cells.

DEF/C: Gives a listing of the counts by cell.

DEF/L: Provides a listing of the data sorted in ascending order.

Have a little patience when working with a lot of data points. Your PC is pretty fast, but it cannot match an IBM 370. Oh yes, the program is designed to accept data having up to two signficant digits on either side of the decimal point. You may change this by altering the USING statements in the program.

Example *Operation of Histogram Program*



```
STATISTICS

                                        CELL DATA
DATA MIN =      3.90
DATA MAX =      8.50             CELL   1 =   1.00   3.00
MEAN     =      6.96             CELL   2 =   2.00   4.00
STD DEV  =      1.50             CELL   3 =   1.00   5.00
VARIANCE =      2.27             CELL   4 =   3.00   6.00
ENTRIES  =     18.00             CELL   5 =   3.00   7.00
                                 CELL   6 =   8.00   8.00

%                                     RAW DATA LISTING
TOT          HISTOGRAM
 44                                 DATA PT.   1 =   3.90
                                    DATA PT.   2 =   4.20
 35                                 DATA PT.   3 =   4.40
                                    DATA PT.   4 =   5.70
 26                                 DATA PT.   5 =   6.30
                                    DATA PT.   6 =   6.40
 17                                 DATA PT.   7 =   6.60
                                    DATA PT.   8 =   7.10
  8                                 DATA PT.   9 =   7.70
                                    DATA PT.  10 =   7.90
                                    DATA PT.  11 =   8.00
    3.00                 9.00       DATA PT.  12 =   8.00
                                    DATA PT.  13 =   8.10
    MAX FREQ. =      8.00           DATA PT.  14 =   8.10
    CELL WIDTH=      1.00           DATA PT.  15 =   8.10
                                    DATA PT.  16 =   8.10
                                    DATA PT.  17 =   8.20
                                    DATA PT.  18 =   8.50
```

## THREE-DIMENSIONAL PLOTS

Here is a program that makes it possible to represent three-dimensional surfaces. Curves consisting of the interesection of the surface with a number of equally-spaced "cutting planes" are drawn by the plotter.

To use the program with your PC-2/PC-1500 and plotter unit, line 100 must contain the formula for computing Z as a function of X and Y. A second function beginning at line 200 may be added, if desired.

Once the program has been started, prompts are given for the minimum and maximum values of X, Y and Z. You must also indicate the number of sections (cutting planes) to be drawn and the "increment size" (distance between the cutting planes).

It can take a number of minutes to produce one drawing since the number of points to be plotted can be quite large. Using relatively large increment sizes and reducing the number of cutting planes can speed up the process. However, this results in less detail being shown. Suitable compromise values for the number of sections seems to be in the range of 7 to 11 planes.

The X, Y and Z axes are not drawn by this program in order to avoid unnecessary clutter.

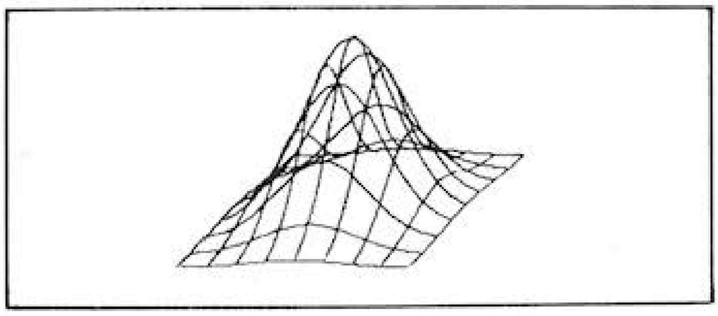
Here are the values used to draw the accompanying sample (for the function included at line 100 in the program listing):

```
MIN X? -2.5
MAX X? 2.5
MIN Y? -2.5
MAX Y? 2.5
MIN Z? 0
MAX Z? 1.3
NUMBER OF SECTIONS? 9
INCREMENT SIZE? 2
```

For variety, you might want to experiment with using different colors to draw the cutting planes. The effects could be interesting.

This program submitted by: *Norlin Rober, 407 North 1st Avenue, Marshalltown, IA 50158.*

Example *Three-Dimensional Plot*



Program *Three-Dimensional Plotting*

```
10:INPUT "MIN X?
   ";A
11:INPUT "MAX X?
   ";B
12:INPUT "MIN Y?
   ";C
13:INPUT "MAX Y?
   ";D
14:INPUT "MIN Z?
   ";E
15:INPUT "MAX Z?
   ";F
16:INPUT "NUMBER
   OF SECTIONS? "
   ;G:G=G-1
17:H=1:INPUT "INC
   REMENT SIZE? "
   ;H
18:N=1:INPUT "NUM
   BER OF FUNCTIO
   NS? ";N
20:CLS :TEXT :
   CSIZE 2
21:LPRINT "GRAPHE
   D:";FOR I=1TO
   N:LLIST 100*I,
   100*I+99:LF -3
   :NEXT I:LF 1
22:LPRINT "MIN X=
   ";A:LPRINT "MA
   X X=";B:LPRINT
   "MIN Y=";C:
   LPRINT "MAX Y=
   ";D:LPRINT "MI
   N Z=";E:LPRINT
   "MAX Z=";F
30:XS=(B-A)/144,Y
   S=(D-C)/144,ZS
   =144/(F-E),GS=
   144/G
40:GRAPH :
   GLCURSOR (0,-2
   20):SORGN
50:FOR I=1TO N
51:FOR J=0TO G:K=
   1:XC=J*GS:FOR
   YC=0TO 144STEP
   H:GOSUB 60:
   NEXT YC:NEXT J
52:FOR J=0TO G:K=
   1:YC=J*GS:FOR
   XC=0TO 144STEP
   H:GOSUB 60:
   NEXT XC:NEXT J
53:NEXT I
54:GLCURSOR (0,-7
   0):END
60:X=B-XS*XC:Y=C+
   YS*YC:GOSUB 10
   0*I:L=.5*XC+YC
   :M=.5*XC+ZS*(Z
   -E)
61:IF Z<EOR Z>F
   LET K=1:RETURN
62:IF KGLCURSOR (
   L,M):K=0:
   RETURN
63:LINE -(L,M):
   RETURN
100:Z=EXP (-(X*X+Y
    *Y)/2)
199:RETURN
200:REM  2ND FUNCT
    ION
299:RETURN
STATUS 1            735
```

## A MACHINE-LANGUAGE MONITOR FOR THE PC-1500/PC-2

This Monitor for the Sharp PC-1500/Radio Shack PC-2 is designed to make it easier to write, load and debug machine-language programs. The program was created by *Norlin Rober.* Norlin is the originator of the mnemonics which *PCN* uses when discussing these units' machine-language. (The instruction set was presented in the Special Edition of *PCN* published in September of 1982. A summary of the mnemonics and machine codes, on a pocket-size card for easy reference, is being distributed with this issue of *PCN.*)

This is what is sometimes referred to as a "hybridized" program. The portion of the program involving operator inputs and outputs is written in BASIC. The remainder is in machine language. One reason for using the latter is that execution is much faster, particularly for operations such as MOVE and HUNT.

The Monitor includes the commands shown in the accompanying list. They are executed by the indicated user-defined keys.

List *Monitor Commands*

| | |
|---|---|
| DEF/S | STORE codes into memory. |
| DEF/V | VIEW memory contents. |
| DEF/L | LIST memory contents using printer. |
| DEF/H | HUNT for a specified byte. |
| DEF/M | MOVE a block of stored codes. |
| DEF/B | BREAKPOINT insertion in user's m.l. program. |
| DEF/J | JUMP to specified address in user's m.l. program. (If a breakpoint is reached, contents of CPU registers and stack may be printed.) |
| DEF/A | ALTER contents of CPU registers at breakpoint. |
| DEF/C | CONTINUE execution following breakpoint. |
| DEF/F | FIX program (remove breakpoint). |
| DEF/X | EXIT from STORE, VIEW or HUNT mode. |
| DEF/= | Convert decimal number in display to hexadecimal. |

### Loading the Monitor into Memory

This program is intended for use in a PC having an 8K memory module. The first step in loading the program is to execute the command: NEW &4000. This reserves a portion of memory for machine-language purposes. There are then two parts to entering the actual Monitor:

1). The accompanying machine-language codes must be entered into an appropriate section of memory using POKE statements. Although this is a tedious process, once it is finished the codes may be stored on cassette for easy loading in the future. It is important that the codes be stored in the exact locations specified.

2). The accompanying BASIC portion of the program is then keyed into memory in the usual way. It too may be saved on cassette for ease in future loading of the Monitor.

It is a good idea to save the program on cassette *before* testing it! Then, in the event you have made any keying errors that result in a bad program crash, you will not have to key everything all over again.

### Saving/Loading the Monitor Program Using a Cassette

After the two parts of the program have been keyed into memory, they should be recorded on cassette as follows:

1). Execute CSAVE M "MONITOR PART 1";&3E4F, &3FFF
2). Execute CSAVE "MONITOR PART 2"

To reload the program from cassette, use:

1). NEW &4000
2). CLOAD M
3). CLOAD

### Monitor Memory Map

The Monitor program uses memory areas as follows:

| | |
|---|---|
| 3800 – 38C4 | REServe memory area, not used by Monitor. |
| 38C5 – 3DFF | Available for user's m.l. routines (1339 bytes). |
| 3E00 – 3E4B | Monitor temporary registers and stack area. |
| 3E4F – 3FFF | Monitor machine-language routines. |
| 4000 – 5FFF | BASIC storage area. Monitor uses 929 bytes. |

The Monitor also uses variables A, B, C, A$, B$, C$ and D$. Thus, those memory areas (&7900 – &7917 and &78C0 – &78FF) may be altered when the Monitor is in operation.

### Operating the Monitor

Place the PC in the RUN mode and use the DEFined keys to select the various operations. Always enter addresses as four hexadecimal digits (using 0 – 9 and A – F). Machine codes should be entered as two-digit hexadecimal numbers. Note that (for example) the code 0B should be entered by keying both 0 and B, not simply B. The & prefix normally used to indicate hexadecimal values to the PC is not necessary when responding to Monitor prompts.

The following discussion provides information on the use of each of the Monitor commands:

STORE: DEF/S places the computer in the mode used to load and store machine code. The Monitor will prompt for the beginning address at which codes are to be stored. Respond to this prompt with a four-digit hexadecimal address (such as 38C5) and then press the ENTER key. The program will then show the address into which the next byte of code will be placed. The code must be entered as a two-digit hexadecimal number (for example, B5) which is terminated by the ENTER key. After each byte has been accepted, the display shows the next sequential address. You may thus continue entering object code into successive memory locations. If you want to skip over an address, then just press ENTER without any input. This advances the Monitor to the next address, leaving the contents of the skipped address unchanged. To exit this mode, key DEF/X.

VIEW: Key DEF/V to place the Monitor in the mode used to examine the contents of memory. Enter a beginning address when prompted. The computer will then display the contents of memory in groups of four bytes. Press the ENTER key to advance to the next group of four bytes. Use DEF/X to terminate this mode.

LIST: Initiated by DEF/L, this command operates similarly to VIEW, except that the memory contents are printed instead of being displayed on the LCD. However, two address prompts will appear: one to indicate the starting address, the other denoting the ending location.

HUNT: Key DEF/H. Enter the byte being sought at the prompt. A second prompt asks for the address at which the Monitor is to begin searching. When a byte having the indicated value is located, the display will show the address, the byte itself, and the three bytes that follow it. Press ENTER to continue the search for another occurrence of the same byte.

As an example, the Monitor command makes it easy to locate the codes used at the beginning of BASIC program lines. Simply HUNT for the byte 0D! (The code 0D is the ASCII representation for "carriage return" used to terminate lines of BASIC.) Given 4000 as a beginning address (assuming this Monitor program is in memory), the display will show 401E 0D 00 0C 08. This indicates that the first BASIC program line (of the Monitor) terminates at address 401E. The next line number (using two bytes starting at 401F) is 00 0C (line 12 in decimal). The last byte (08) in the display represents the "link" byte. It indicates the number of bytes that remain in that BASIC program line. If you press the ENTER key, you will find where the next line ends. Use DEF/X to exit from this mode.

The HUNT command is also highly useful as an aid to locating specific instruction codes when investigating ROM.

The speed at which HUNT operates dramatically illustrates the advantage of using machine language.

MOVE: DEF/M initiates this mode. Using it, a block of stored codes of any size may be moved, in either direction, to any specified RAM location. The new location may overlap the original. Unless there is overlapping, the original block of code remains unchanged by the operation.

Prompts request the beginning and ending addresses of the original block of memory, and then the beginning address of the location to which it is to be transferred.

Be careful when using this command. If an incorrect address is inadvertently given, you may destroy parts of memory that you did not want altered! A wrong move operation can alter the Monitor program itself or disturb pointers in system RAM, possibly resulting in a system crash.

The following example may be used to familiarize yourself with the command. The variable J is stored (by BASIC) in RAM addresses 7948 to 794F. Variable M begins at address 7960. You can copy the contents of variable J into M using MOVE. Use BASIC to store some arbitrary test number in J. Key DEF/M to initiate MOVE. Enter the address 7948 in response to the prompt BEGIN BLOCK and address 794F for END.BLOCK. In response to MOVE TO, key in the address 7960. Press ENTER at the end of this address and the MOVE command will be executed. You can then use BASIC to display the contents of the variable M. It should contain the same value as that originally placed in J.

BREAKPOINT: This command is useful in debugging machine-language programs. What it does is remove three bytes from a program in RAM, at whatever address is specified, and save them elsewhere. The three locations are then loaded with opcodes for an instruction to jump to a register-saving routine, which saves the contents of the CPU registers, flag status register and the stack. If execution is resumed later, (for example, by the Monitor's CONTINUE command), the CPU registers, flags and stack will be restored.

Note that the BREAKPOINT command executed by DEF/B simply inserts an instruction to jump to the register-saving routine. It does not execute the user's machine-language program. Also note that only one breakpoint should be in a program at any given time.

As a note of caution: be sure that the address entered for insertion of a breakpoint is the address of the *first byte* of a machine-language instruction!

JUMP: Key DEF/J to use this routine. The address entered in response to the prompt will be taken as the beginning address of the code to be executed.

This command may be used in lieu of a BASIC "CALL" instruction to begin executing a machine-language program. It *must* be used when debugging a program containing a breakpoint.

If a program containing a breakpoint is executed (using the JUMP command), BREAKPOINT REACHED will be displayed when the breakpoint is found. The user may then initiate a printout of the CPU registers, flags and stack by pressing ENTER.

ALTER: DEF/A will begin the routine for altering the contents of the CPU registers following the reaching of a breakpoint. When the prompt for a particular CPU register appears, the new value should be entered. Note that CPU register A contains one byte, requiring inputting of two hexadecimal digits. Registers X, Y and U, on the other hand, each hold two bytes, hence four-digit values should be entered for those registers.

If you want to leave a particular register undisturbed, press ENTER alone when the prompt for that register appears.

After all desired alterations have been entered, the three bytes that had been removed from the program by the BREAKPOINT command will be replaced. Execution of the machine-language program then continues from where it left off, with the CPU registers now containing the altered values specified.

CONTINUE: Execution of DEF/C will continue execution of a program following a breakpoint, with no alteration of the CPU registers.

FIX: The DEF/F command replaces the three bytes that were removed by a breakpoint directive. It is only needed if neither ALTER or CONTINUE are used following establishment of a breakpoint, since those two commands make the same replacement.

EXIT: Use DEF/X to exit from the STORE, VIEW or HUNT modes. The command is provided as a convenience, rather than a necessity. (The use of SHIFT/CL would accomplish the same effect.)

HEXADECIMAL equivalents of calculated results may be obtained using DEF/=. This is useful for such things as calculating the required size of a relative branch. An example illustrating hexadecimal conversion follows:

Enter 35 * 61 into the display. Now key DEF/=. The product will be shown as 0857 which is the hexadecimal equivalent of the product of the decimal numbers 35 and 61. Enter &3A56 - &39EB and key DEF/=. You will have an instant answer for a hexadecimal subtraction!

You may also simply convert a decimal number to hexadecimal by entering the number into the display and using DEF/=.

**The BREAKPOINT Illustrated**

To amplify how BREAKPOINT and related commands operate, I will illustrate their use with a specific example.

To begin, input a short, simple machine-language program. Put the codes into memory using the STORE command. Here is such a routine:

| Address | Opcodes | Mnemonics |
|---|---|---|
| 38C5 | 48 12 | LDXH #12 |
| 38C7 | 4A 34 | LDXL #34 |
| 38C9 | 58 56 | LDYH #56 |
| 38CB | 5A 78 | LDYL #78 |
| 38CD | 68 9A | LDUH #9A |
| 38CF | 6A BC | LDUL #BC |
| 38D1 | B5 00 | LDA #00 |
| 38D3 | FD 88 | PSHX |
| 38D5 | FD 98 | PSHY |
| 38D7 | FB | SETC |
| 38D8 | FD 1A | POPY |
| 38DA | FD 0A | POPX |
| 38DC | AE 39 00 | STA 3900 |
| 38DF | A4 | LDA UH |
| 38E0 | AE 39 01 | STA 3901 |
| 38E3 | 24 | LDA UL |
| 38E4 | AE 39 02 | STA 3902 |
| 38E7 | 9A | RTS |

A careful examination of this example routine will reveal that it does not accomplish anything of any practical importance. The purpose of the routine is simply to illustrate how some of the Monitor features perform.

After storing the example codes, use VIEW to make certain that they have been entered correctly. Then try executing the program with the JUMP command, giving address 38C5 as the response to the BEGIN prompt. To see whether the program did what it was designed to do, use VIEW again to see if addresses 3900 to 3903 contain appropriate values.

Try a breakpoint at the address 38D8 by keying DEF/B and entering 38D8 at the prompt. Then execute JUMP again using 38C5 as the starting location. The BREAKPOINT REACHED message should soon appear. Assuming that your PC is connected to its printer, press the ENTER key. Compare the printout against what the machine-language routine should have accomplished to that point. (The "CIZ" in the printout means that flags C, I and Z were set at the time the breakpoint was encountered.)

You can then alter the contents of, say, registers A and U, then see whether the altered values eventually find their way to addresses 3900 to 3902. First, key DEF/A. The prompt "A:" appears to request a new value for register A. Enter, for trial purposes, "BB" and press the ENTER key. Skip over changing X and Y by just pressing ENTER when their prompts come up. When "U:" appears, input FFEE as a new value and then press ENTER.

Use VIEW starting at address 3900 to verify that the CPU alterations took effect.

Key DEF/F to repair the breakpoint. You will be informed that there is no breakpoint to be fixed! Remember, using the ALTER feature automatically "fixes" the breakpoint. If you try either ALTER or CONTINUE at this point (without putting in another breakpoint), you will get the same response.

However, there is no built-in protection against your entering a second breakpoint without having fixed the first one. If you put in more than one breakpoint at a time, the FIX routine will only correct the last one entered. (The Monitor will have then lost track of where you placed the first breakpoint and what the original bytes were.)

Program *Machine Language Portion of Monitor*

```
3E4F BE 3F DE 58     3F2B 46 9A BE 3F
3E53 3F 5A 19 84     3F2F DE FD 5A F4
3E57 51 04 51 F5     3F33 79 05 FD 28
3E5B F5 F5 5B FF     3F37 FB A5 79 0E
3E5F 46 B5 71 43     3F3B 00 2A A5 79
3E63 B5 3E 43 B5     3F3F 0D 80 28 84
3E67 BA 0E 9A EB     3F43 96 81 0E 89
3E6B 3E 70 FF FD     3F47 04 04 16 81
3E6F 5E FF FD C8     3F4B 08 F5 88 03
3E73 FD 88 FD 98     3F4F FD 62 93 07
3E77 FD A8 FD AA     3F53 9A 24 FD CA
3E7B FD C8 FD 48     3F57 FD DA 84 A2
3E7F FD 88 46 58     3F5B 08 94 A2 18
3E83 3E 5A 00 F5     3F5F 47 53 88 04
3E87 4E 49 91 05     3F63 FD 62 93 08
3E8B AA 78 48 E9     3F67 9A F2 48 79
3E8F 3E 70 00 9A     3F6B 4A 10 BE DC
3E93 48 3F 4A 19     3F6F 20 58 78 5A
3E97 BE 3F 9A 5A     3F73 E0 48 7A CD
3E9B D0 48 3E 4A     3F77 6C B7 00 8B
3E9F AF A5 3E 02     3F7B 02 B5 2D BB
3EA3 6A 04 D5 81     3F7F 20 51 D0 00
3EA7 02 F5 46 44     3F83 04 FD 28 8E
3EAB 88 08 8E DE     3F87 32 56 CD 2A
3EAF 43 49 5A 56     3F8B 65 05 8E 44
3EB3 48 48 3E 4A     3F8F BE 3F B8 6A
3EB7 03 BE 3F 9A     3F93 03 BE 3F AE
3EBB 5A F0 BE 3F     3F97 88 05 9A 5A
3EBF 9C 5A C0 BE     3F9B E0 58 78 B5
3EC3 3F 9C 8E E3     3F9F 20 51 8E 02
3EC7 48 3F 4A 19     3FA3 5A E0 BE 3F
3ECB 45 18 45 1A     3FA7 B5 8E 0B 58
3ECF F5 F5 F5 49     3FAB 78 5A D0 B5
3ED3 00 9A 48 3E     3FAF 20 51 8E 02
3ED7 4A 09 8E FC     3FB3 5A D0 45 8E
3EDB 4A 07 8E 06     3FB7 07 5A C0 84
3EDF 4A 05 8E 02     3FBB BE 3F BF 04
3EE3 4A 03 48 3E     3FBF 58 78 28 F1
3EE7 BE 3F D7 8E     3FC3 BE 3F C7 A4
3EEB ED AA 3D FF     3FC7 F9 B9 0F B3
3EEF FD 0A FD 4E     3FCB 30 B7 3A 81
3EF3 44 58 3E 5A     3FCF 02 B3 06 51
3EF7 02 55 41 4E     3FD3 59 00 FB 9A
3EFB 49 91 06 FD     3FD7 5A D0 BE 3F
3EFF 8A FD EC FD     3FDB E9 41 9A 5A
3F03 2A FD 1A FD     3FDF C0 BE 3F E9
3F07 0A 8B 08 FD     3FE3 08 BE 3F E9
3F0B 8A ED C0 00     3FE7 0A 9A 58 78
3F0F FF 8E 06 FD     3FEB BE 3F F6 F1
3F13 8A ED C0 00     3FEF 28 BE 3F F6
3F17 00 BA 00 00     3FF3 A2 FB 9A 55
3F1B 00 00 00 00     3FF7 B7 40 81 02
3F1F A5 79 15 F7     3FFB B3 08 B9 0F
3F23 99 03 46 BE     3FFF 9A
3F27 3F 8F 46 46
```

Program *BASIC Portion of Monitor*

```
10:"="AREAD C:
   CALL &3F68:
   CURSOR 21:
   PRINT C$:END
12:"S"GOSUB 68
14:CALL &3FB8,A:
   WAIT 0:PRINT A
   $;:WAIT :INPUT
   ": ";B$:CLS :
   CALL &3FD7,A:
   GOTO 14
16:CLS :A=A+1:
   GOTO 14
18:"V"GOSUB 68
20:CALL &3F8F,A:
   PRINT A$:GOTO
   20
22:"L"GOSUB 72
24:CALL &3F8F,A:
   LPRINT A$:IF A
   >BEND
26:GOTO 24
28:"H"CLS :INPUT
   "BYTE SOUGHT:
   ";A$:CALL &3FD
   E,C:GOSUB 68
30:CALL &3F1F,A:
   PRINT A$:GOTO
   30
32:"M"CLS :INPUT
   "BEGIN BLOCK:
   ";A$:CALL &3FD
   E,A:INPUT "END
    BLOCK: ";A$:
   CALL &3FDE,B
34:INPUT "MOVE TO
   : ";A$:CALL &3
   F2D:END
36:"X"CLS :END
38:"B"CLS :INPUT
   "BREAKPOINT: "
   ;A$:CALL &3E4F
   :END
40:GOTO 38
42:"J"GOSUB 68:
   CALL &3E6A,A:
   IF PEEK &3E70
   END
44:PRINT "BREAKPO
   INT REACHED"
46:CALL &3E93:
   LPRINT "ADDRES
   S:";C$:LPRINT
   "FLAGS: ";B$
48:CALL &3EB4:
   LPRINT "A:";B$
   :LPRINT "X:";A
   $:LPRINT "Y:";
   D$:LPRINT "U:"
   ;C$
50:B=64-PEEK &3E0
   1:IF B>0LPRINT
   "STACK:";:A=&3
   E0A:FOR C=1TO
   B:CALL &3FAA,A
   :LPRINT B$;:
   NEXT C:LPRINT
52:LPRINT :END
54:"A"GOSUB 76:
   INPUT "A: ";B$
   :CALL &3ED5
56:INPUT "X: ";B$
   :CALL &3EDB
58:INPUT "Y: ";B$
   :CALL &3EDF
60:INPUT "U: ";B$
   :CALL &3EE3
62:CALL &3EEC:END
64:"C"GOSUB 76:
   GOTO 62
66:"F"GOSUB 76:
   END
68:CLS :INPUT "BE
   GIN: ";A$:CALL
   &3FDE,A:RETURN
70:GOTO 68
72:GOSUB 68:INPUT
   "END: ";A$:
   CALL &3FDE,B:
   RETURN
74:GOTO 72
76:CLS :IF PEEK &
   3F1ECALL &3EC7
   :RETURN
78:PRINT "NO BREA
   KPOINT":END
STATUS 1          929
```

## Crash Recovery

Anyone experimenting with machine language soon learns that the PC (or any computer!) is totally unforgiving of errors. There are none of the friendly "ERROR" messages as found in BASIC. In many cases, the simplest mistakes will be punished by "crashes" which result in the PC becoming completely locked up. The only thing to be done in such situations is to use the "All Reset" key on the back of the PC. But, all is not necessarily always lost!

Instead of following the instructions given in the manual when a crash occurs, try using the following approach. Nine times out of ten your program will still be intact: First, do *not* hold in the ON key while pressing "All Reset." Secondly, after you obtain "NEW0? CHECK" on the display, do *not* execute NEW0! Instead, key SHIFT and then CL (SHIFT/CL). Then, survey the damage, if any, to the contents of memory. If you find things messed up beyond repair, then you will have to key in NEW0 and start over. Often, however, you will find memory contents retained with little alteration.

The Monitor program does contain a few safeguards. If you neglect to key in an *address* before pressing ENTER, the computer will repeat the prompt. If you forget any of the three addresses required for a MOVE, it will restart the entire MOVE routine from the beginning. And, if you use the hexadecimal conversion routine with a result exceeding FFFF (or less than -FFFF), the Monitor will put the word ERROR into the display. Otherwise, you are on your own!

### Combined Monitor/Disassembler

The use of this Monitor program can be greatly enhanced by coupling it with the Disassembler program published in *PCN* (Special Edition accompanying Issue 17). Those who have the Disassembler on a cassette can simply MERGE that program with this Monitor. To facilitate joint use of the Monitor and Disassembler, change the Disassembler as follows:

1). Insert "D" as a label at the beginning of line 10.
2). Delete lines 90, 91 and 92 as they are no longer needed.
3). It would also be advisable to eliminate lines 12, 80 and 82.

Now DEF/D may be used to DISASSEMBLE! A printed list of mnemonics generated by the Disassembler is often of considerable help in finding elusive machine-language programming errors!

If you are interested in doing machine-language programming on your PC-2/PC-1500, you should find this Monitor of considerable value. The 1339 bytes (from address &38C5 through &3DFF) that are available for machine-language routines should be adequate for most purposes. (If you are able to write longer routines, you should have little difficulty figuring out how to relocate the Monitor program!)

## SIDELISTER

*Mel Beckman, 717 West Broadway, Winona, MN 55987,* developed this program and the following narrative:

One inconvenience of pocket computers that I had resigned to put up with is the hard to read program listings produced on those teeny weeny printers. Recently it occurred to me that Sharp had nearly provided an alternative to this in the form of sideways printing. The ability to print sideways going up, down or inverted is provided by the ROTATE command. ROTATE 0 causes text to print normally. Using ROTATE 1 turns the output 90 degrees clockwise; ROTATE 2, 180 degrees; and ROTATE 3, 270 degrees. After a ROTATE has been executed, all subsequent LPRINT directives will print in that orientation. You might expect that entering ROTATE 1 followed by LLIST would cause the program listing to print sideways. The nice thing about such a listing would be the ability to print long lines without those eye-jogging continuations that occur when long lines have to be printed down a column. Alas, it is not that simple.

### Where There Is a Will. . .

A somewhat clumsy method of obtaining a sideways program listing is to write a small basic program that PEEKs at the entire program and prints it using ROTATE and LPRINT. That is what this program does. It is really not an ideal solution to the problem, but it does produce a sideways listing. The program is designed to be appended to the end of a user's program. Thus, it uses high line numbers (64000+) to avoid conflict with the user's code. The output may be printed in either CSIZE 1 or CSIZE 2. The former permits up to 84 characters per line and is nice for archival listings. CSIZE 2 prints up to 42 characters per line and is good for debugging sessions. The program automatically advances to a new "page" (after 20 lines at CSIZE 1 or 10 lines at CSIZE 2). It also correctly continues unusually long statements onto the next line when necessary.

Program *Sidelister*

```
64000:"L"REM  SIDE
      LISTER
64025:Z=2:X=212-Z*
      6
64050:GRAPH :
      GLCURSOR (X,
      0):SORGN :
      ROTATE 1:
      CSIZE Z
64100:X=0:L=0:C=0:
      M=0:Q=STATUS
      2-STATUS 1:
      LPRINT "*TOP
      *";TIME
64105:GOSUB 64200
64110:IF PEEK (Q)=
      &FFEND
64115:LPRINT USING
      "######";
      PEEK (Q)*256
      +PEEK (Q+1);
      ":";
64120:Q=Q+3:C=C+7
64125:IF PEEK (Q)=
      &0DLET Q=Q+1
      :GOTO 64105
64130:GOSUB 64400
64135:GOTO 64125
64200:IF C>MLET M=
      C:IF M>(84/Z
      )LET M=84/Z
64205:C=0:X=X-Z*10
      :GLCURSOR (X
      , 0)
64210:L=L+1:IF L=
      INT (20/Z)
      GOSUB 64300
64215:RETURN
64300:L=0:X=0:Y=-(
      Z*M*6.4):
      GLCURSOR (X,
      Y):SORGN :M=
      0:RETURN
64400:IF PEEK Q>&E
      5GOTO 64450
64405:IF C>(84/Z)
      GOSUB 64500
64410:LPRINT CHR$
      PEEK Q:C=C+1
      :Q=Q+1:
      RETURN
64450:V=PEEK (Q):W
      =PEEK (Q+1):
      U=&C053
64460:N=PEEK UAND
      &0F
64470:IF (PEEK (U+
      N+1)=V)AND (
      PEEK (U+N+2)
      =W)GOTO 6449
      0
64475:U=U+N+5:IF U
      >&C348LET U=
      &B054:GOTO 6
      4460
64480:IF (U<&C053)
      AND (U>&B0E2
      )LPRINT "? "
      ;:Q=Q+2:C=C+
      2:RETURN
64485:GOTO 64460
64490:IF (C+N+1)>(
      84/Z)GOSUB 6
      4500
64495:FOR J=1TO N:
      LPRINT CHR$
      (PEEK (U+J))
      ;:NEXT J
64498:LPRINT " ";:
      C=C+N+1:Q=Q+
      2:RETURN
64500:GOSUB 64200:
      LPRINT "
          ";:C=C+7:
      RETURN
STATUS 1          739
```

**How It Works**

The first task of Sidelister is to determine where the program it is going to list starts in memory. STATUS 2 less STATUS 1 provides that value. Sidelister thus starts PEEKing at that address and stops when it finds a hexadecimal value of FF as that is the value used to signify the end of a program.

BASIC statements are not actually stored character-for-character in the PC-1500/PC-2. To conserve space the line number is saved as a two-byte binary value and keywords are converted into two-byte "tokens". The format of such a compressed BASIC statement line is:

| 2-byte line # | length byte | statements | &0D |
|---|---|---|---|

The text length byte shown in the diagram is used by the BASIC interpreter to enable it to rapidly scan forward through a program. Since the Sidelister program will be processing every program line, it can ignore this byte. The code &0D at the end of every program line represents an ASCII carriage return. It is used by Sidelister to advance to the next line to be printed.

Tokens are expanded back into keywords using the keyword/token table stored in ROM. This is the same table that the BASIC interpreter uses. There are actually two tables. One starts at address &C053, the other at &B054 (for the CE-150 printer/cassette statements). Each table entry is formatted as follows:

| length byte | keyword | token code | address |
|---|---|---|---|

The first byte of the table entry contains the length of the keyword in the four low-order bits. The high-order bits are apparently used as some sort of flag. Extracting the four low-order bits is accomplished by using an AND with the value &0F. The expanded keyword appears next in the table, followed by the two-byte token value used to represent the keyword. Tokens always begin with a byte greater than &E5 which makes them easy to identify. The last two bytes of each entry contain the address of the ROM routine that processes the statement.

A simple BASIC loop is used to scan these tables and translate the tokens into corresponding keywords for the listing. A blank is added to the end of each keyword for spacing.

Referring to the Sidelister program listing, lines 64000 – 64100 perform "housekeeping". Note variable Z in line 64025. Set this to select the CSIZE value (1 or 2) desired. While values of 3 through 9 could conceivably be used, the listings would be unduly large. Lines 64105 – 64135 constitute the main program loop. It extracts and prints the line number, checks for a carriage return code, and terminates upon encountering an &FF byte. The "new line" routine begins at line 64200, the "new page" at 64300, and the print routine at line 64400. The print routine fetches a byte, determines if it is a token that needs expansion, expands it if required, then prints appropriately. Line 64500 is used to continue long statements onto the next line.

**Is It Practical?**

After comparing various program listings in vertical and sideways formats, I strongly prefer sidelistings. The elimination of a lot of continuation lines improves readability. Debugging a program from a sideways listing is less strenuous on the eyes. The logical flow of a program seems more apparent. It is nice to be able to fold a listing between "pages" thereby eliminating annoying creases in the middle of a line. The horizontal arrangement is easier to scan than a skinny vertical printout. I would use this listing format exclusively if it could be obtained without undue effort.

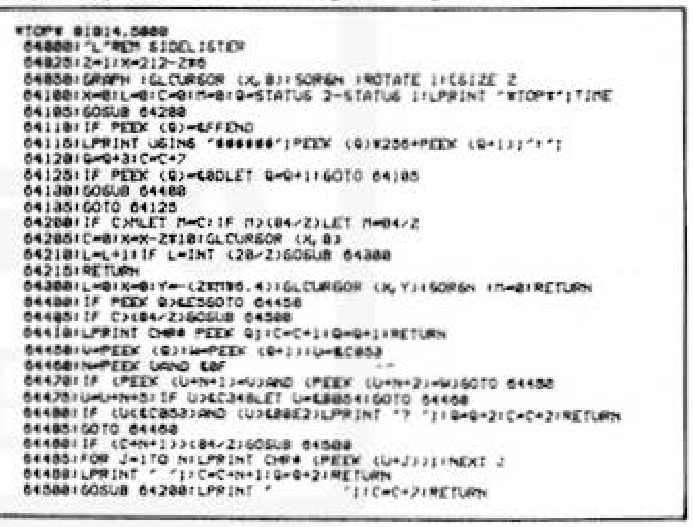
Earlier I stated that this was not the ideal solution to the sidelisting problem. The best solution would be to have the capability provided in ROM. Maybe some future version of the unit will include this feature. The program suffers from a number of disadvantages that make practical application doubtful in its present form. Not the least of these is its slow execution speed. Looking up tokens with a BASIC search routine is not quick (taking 1 – 8 seconds each). Another drawback is having to append the program at the end of each program that is to be so listed. This burden can be eased by merging the routine from tape, but the code still utilizes valuable memory space.

However, I did not write Sidelister as a practical answer to the problem. It was an experiment; a tool to investigate the usefulness of such listings. In this capacity it has served well. The program is functional, but not efficient. It was developed using the simplest methods available without regard to execution speed or memory usage. This is a viable way to approach many software projects: programs can always be "tuned up" later; get them running first!

Immediate improvements readily come to mind. Somewhere in ROM is a routine that will return a keyword when given a token as input. Finding this routine would considerably reduce the execution time and cut the program size perhaps in half. There may well be a ROM routine that returns expanded statements, line-by-line. If not, machine-language routines could accomplish tremendous speed enhancements. I do hope that other ingenious persons will take the outline presented here and produce a Son of Sidelister. Perhaps additional features, such as the ability to print just selected lines or skip to a new page could be added.

If nothing else, perhaps the information disclosed about the internal storage of the token tables and BASIC program storage may inspire others to create additional programming tools!

Example *Output from Sidelister Program Using CSIZE 1*



```
*TOP* 01014.5000
64000:"L"REM SIDELISTER
64025:Z=1:X=212-Z*8
64050:GRAPH :GLCURSOR (X,0):SORGN :ROTATE 1:CSIZE Z
64100:X=0:L=0:C=0:M=0:Q=STATUS 2-STATUS 1:LPRINT "*TOP*";TIME
64105:GOSUB 64200
64110:IF PEEK (Q)=&FFEND
64115:LPRINT USING "######";PEEK (Q)*256+PEEK (Q+1);":";
64120:Q=Q+3:C=C+7
64125:IF PEEK (Q)=&0DLET Q=Q+1:GOTO 64105
64130:GOSUB 64400
64135:GOTO 64125
64200:IF C>MLET M=C:IF M>(84/Z)LET M=84/Z
64205:C=0:X=X-Z*10:GLCURSOR (X,0)
64210:L=L+1:IF L=INT (20/Z)GOSUB 64300
64215:RETURN
64300:L=0:X=0:Y=-(Z*M*6.4):GLCURSOR (X,Y):SORGN :M=0:RETURN
64400:IF PEEK Q>&E5GOTO 64450
64405:IF C>(84/Z)GOSUB 64500
64410:LPRINT CHR$ PEEK Q;:C=C+1:Q=Q+1:RETURN
64450:V=PEEK (Q):W=PEEK (Q+1):U=&C053
64460:N=PEEK UAND &0F
64470:IF (PEEK (U+N+1)=V)AND (PEEK (U+N+2)=W)GOTO 64490
64475:U=U+N+5:IF U>&C348LET U=&B054:GOTO 64460
64480:IF (U<&C053)AND (U>&B0E2)LPRINT "? ";:Q=Q+2:C=C+2:RETURN
64485:GOTO 64460
64490:IF (C+N+1)>(84/Z)GOSUB 64500
64495:FOR J=1TO N:LPRINT CHR$ (PEEK (U+J));:NEXT J
64498:LPRINT " ";:C=C+N+1:Q=Q+2:RETURN
64500:GOSUB 64200:LPRINT "       ";:C=C+7:RETURN
```

## TI CC-40 *(concluded from page 1)*

plotter and a Wafertape digital tape drive. Other add-ons, scheduled to be available by the third quarter of 1983 include: a wand input device, modems, printers and a black and white TV monitor. These devices are also said to operate with the new TI-99/2 Basic Computer, and, using an appropriate adapter, will work with the TI-99/4A Family Computer.

List prices of the three immediately-available peripherals are: $99.95 for the RS-232 interface, $199.95 for the 4-color printer/plotter, and $139.95 for the Wafertape drive.

The Wafertape drive will be capable of transferring data in digital format at approximately 1 kilobyte per second. This is considerably faster than most other portable systems now on the market.

Texas Instruments has also announced an array of 22 software application packages as being immediately available, with another 53 packages planned for introduction by the third quarter of 1983. Some software will be supplied on Wafertapes at $19.95 each. These include: Elementary Dynamics, Regression/Curve Fitting, Pipe Design, Production and Planning, Inventory Control, Electrical Engineering, Thermodynamics, Photography and more. Other software packages will come in plug-in ROM cartridges. These include: Mathematics, Finance, Perspective Drawing, Statistics, Business Graphics, Nonparametric statistics, and Advanced Electrical Engineering (at $59.95 each); an Editor/Assembler ($124.95) and two games packages (at $39.95 each).

## 26K OF RAM IN THE SHARP PC-1500 OR RADIO SHACK PC-2

*The project discussed in this article should only be attempted by experienced electronic technicians. While the staff of PCN has successfully implemented this memory expansion on a Sharp PC-1500, it is our opinion that the procedure could best be described as risky. The amount of risk is related to the degree of experience one has had working with micro-electronics. Anyone attempting this project should be well aware that a mistake could result in damage to the pocket computer and/or memory module(s). Furthermore, the very act of attempting to install this modification will void any warranty on the part of Sharp or Radio Shack. Thus, if after reviewing the article, you do not feel you have the experience to perform the operations required, we suggest you have patience. Real 16K memory modules may be available sooner than you think!*

It is possible to install 16K bytes of CMOS RAM inside the Sharp PC-1500 or Radio Shack PC-2. The installation does not require cutting any circuit traces or changing any chips. The module slot on the back will still accept any RAM/ROM plug-ins as long as they do not use any addresses below &4800. Thus, a 4K or (slightly modified) 8K module can be plugged in, making available up to 26,426 bytes of BASIC program space. The installation requires about 25 hours work and two 8K modules (CE-155).

One of the very nice features of the Sharp is that, each time a hard (15 second) reset is done, the computer automatically checks itself to see how much contiguous memory is installed and sets the pointers accordingly. There are no switches to set manually. Thus, memory can start at address 0 and extend to at least &6800 (hexadecimal 6800 or 26,624 decimal).

The unexpanded Sharp has 2K of user RAM at address &4000 to &47FF. Adding a 4K module extends the range to &57FF. You would expect the 8K module to address the range &4000 to &67FF. It does not. Instead, the 8K module addresses a range from &3800 to &5FFF. For some unknown reason, Sharp did not connect the internally decoded Y4 NOT (S4) signal from the 40H138 integrated circuit to any pin of the module socket. Therefore, the upper 6K of an 8K module is selected by S1, S2, and S3, while the lowest 2K is addressed by another 40H138 (1 of 8 decoder chips) inside the module. This apparent oversight and resulting correction is very fortunate for us. The module decoder chip will be used to decode the addresses from 0 to &3FFF. This makes implementation of the 16K expansion easy.

In the Radio Shack PC-2 version, the S4 signal is connected by a jumper to the NC pin of the module socket. Never-the-less, the Radio Shack 8K modules which I have examined do not use it. They contain the same 40H138 for decoding, just as the Sharp version does.

The 8K RAM modules are not sealed shut. The upper and lower halves of the cases will readily snap apart to reveal the multilayer printed circuit board inside. The covers can be snapped back together with no damage. These covers should be kept on as much as possible to avoid damage caused by static electricity. Also, when making the modifications described here, you should use a 3-wire grounded soldering iron (Weller model WP-25-3 or equivalent).

The description that follows assumes that you want to put 16K inside the Sharp and plug in an additional 8K (giving a total of 26K). Before starting, be sure to test each of the 8K modules to verify that they work. You might find it useless to complain about a defective one after you have soldered it.

### Opening the Case

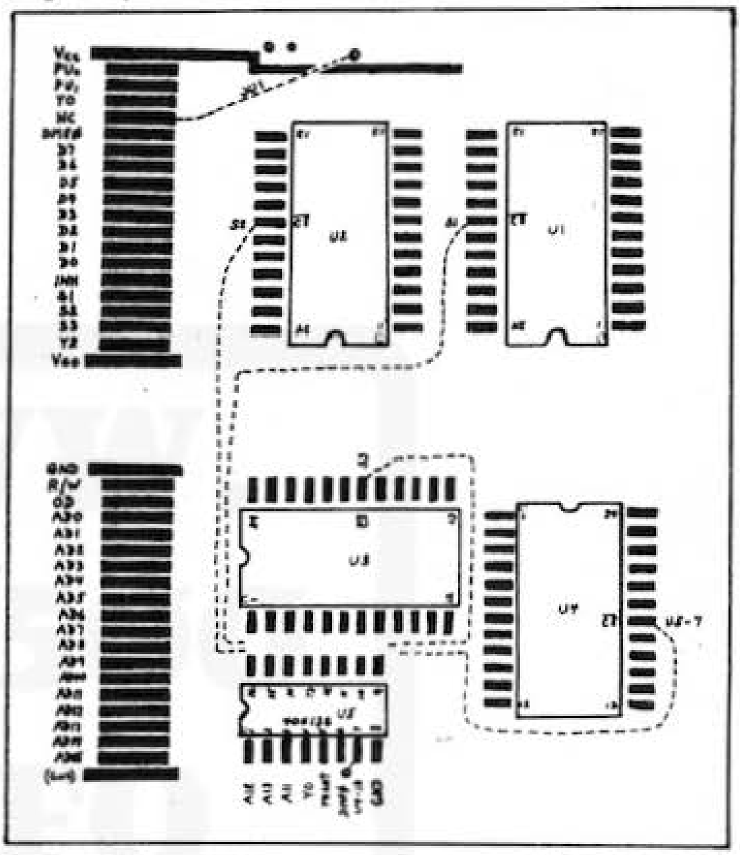
To access the internal circuit boards of the PC it is necessary to remove the back cover. This may be accomplished by removing a total of eight screws from the back of the unit. Five of these screws are visible on the back cover. Three more may be found inside the battery compartment. Once the screws have been removed you can gently separate the back of the unit from the front. They remain hinged together by a piece of flexible circuitry.

### Add the S4 Jumper

This change is required only if you plan to use plug-ins in addition to the 16K being added internally. However, I strongly recommend doing it because the Radio Shack PC-2 version has it and some of their plug-in ROM's may not work in the PC-1500 if you don't put it in. If you have the Radio Shack PC-2, you can skip this step as a jumper will already be installed, though not at the same place.

On the main circuit board of the Sharp solder a jumper from the eyelet next to the screw of the 40 pin module socket to pin 11 (Y4 NOT) of the 40H138. This eyelet connects to the "NC" pin of the module socket. See the accompanying diagram.

Diagram *Top and Bottom View of 8K Memory Module*



### Change Plug-In Module Addressing to &4800 to &67FF

An accompanying illustration shows both sides of a module printed circuit board. After opening a module, very carefully unsolder and lift pin 7 of the 40H138 (U5) chip away from the printed circuit board until it is parallel to the board. The easiest way that I have found to do this is to loop a piece of bare number 30 wire under the pin and pull gently on it while heating the solder joint. Next, connect a jumper (JU1) from the eyelet above U2 to the module pin designated "NC" in the diagram. Use solder sparingly and don't overlap the pin much. Most of the pin should remain shiny gold. Make a small V-notch in the cover so as to avoid pinching the wire. Replace the module covers.

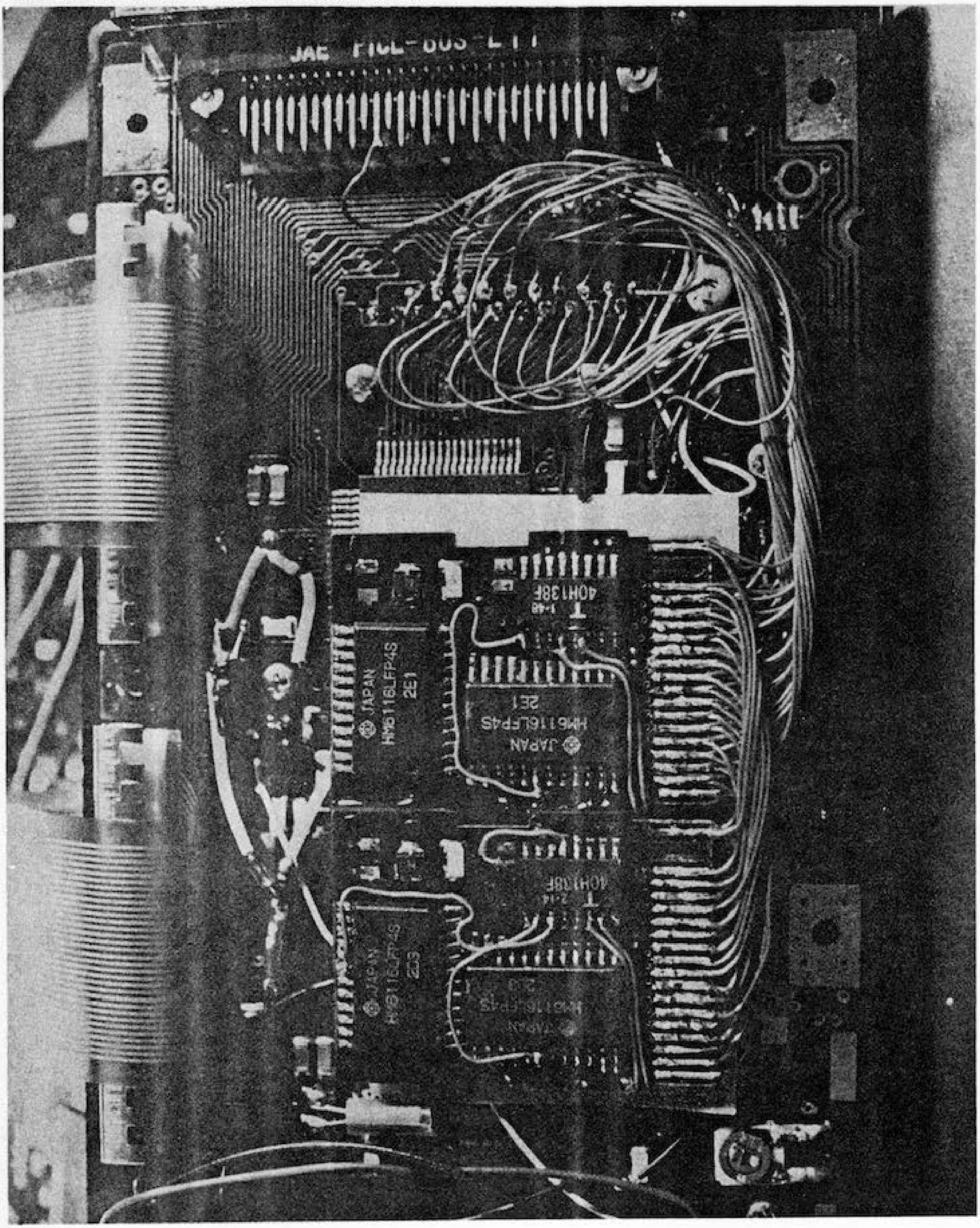
Plug the modified module into the module slot. (Always follow the usual precautions of taking a battery out and holding the reset for 15 seconds.) After NEW0, MEM should now give 10042 as it did before these changes. However, PEEK &7863 should now show &40 (64) instead of &38. STATUS 3 should yield &6800 (26624). These tests will show whether the address range has been moved.

When the tests have been satisfactorily completed, take out the module. Write the new address range on the label (&4800-&67FF). Put the module aside until all the rest of the modifications have been made.

### Change a Module to Address &2000-&3FFF

Open another module. Unsolder and lift the CE NOT pins (pin 18) of U1, U2, and U3. Add three jumpers to connect the CE NOT pins (not the foil) of U1, U2, and U3 to decoder U5 pins 12, 13, and 9 respectively. Route the jumpers as shown by the dotted lines in the diagram. (Note that two of the jumpers will wrap around the edge of the board

Photo *Installation and Wiring of Memory Modules to the CPU Board*



from the bottom to the top sides.)

Without replacing the covers, plug the module into the module slot (U3, U4, and U5 visible). Install the batteries and execute a NEW0. MEM should now read 10042. PEEK &7863 should give &20 (32) and STATUS 3 should show &4800 (18432). Remove the module from the slot and set it aside. You may leave the covers off.

**Change a Module to Address &0000-&1FFF**

Open a third module. Unsolder and lift the CE NOT pins of U1, U2, U3, and U4. Add four jumpers to connect the CE NOT pins of U1, U2, U3, and U4 to the decoder U5 pins 14, 15, 11, and 10 respectively.

Once again, leaving the covers off and being careful to have the right side up, plug in this modified module.

MEM should now yield 7994. This is because the automatic memory check will count only contiguous memory. It will not see the 2K of memory at &4000 to &4800. PEEK &7863 should give 0. STATUS 3 should show &2000 (8192). Remove the module, leave the covers off and set it aside.

**Prepare the 16K Assembly**

Needless to say, space inside the PC is limited. There is not enough room for even one module with the module covers on as the covers add a significant amount of thickness. Without covers, however, two modules will fit comfortably side-by-side in the space between the CPU and ROM and the flexible printed circuits which connect to the second board.

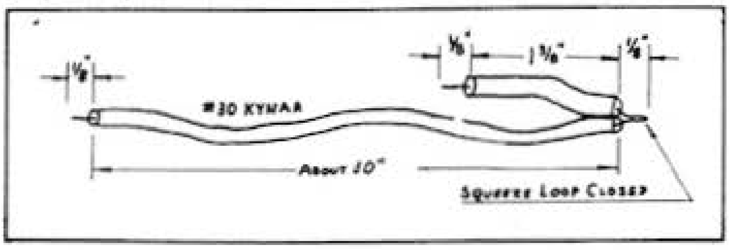
Epoxy the two coverless modules together, edge to edge, same sides up, so that both rows of connector traces are along one side as shown in the accompanying photograph. It doesn't matter which module is at the left or right. This step will make handling and soldering the 16K assembly a neat package which can be removed and transferred to a different computer. Remember, in a year or two, a new, shinier toy may come out.

When the epoxy has hardened, tin the eighty gold-plated connector traces. Use plenty of rosin flux to ensure good wetting and to avoid bridging between traces. The assembly will now be completely wired together before any of the leads are connected to the computer.

**Wire the 16K Assembly**

Using number 30 insulated wire (preferably Kynar or Teflon), prepare 29 wires, each about 8 inches long. Do this by stripping a one-fourth inch section of each wire, one and three-fourths inches from one end. Then strip one-eighth inch from each of the ends. Fold the quarter-inch section back on itself and squeeze together with pliers to form a tight hairpin as shown in the accompanying diagram.

Diagram *Preparation of Insulated Wires*



Place the epoxied module assembly in front of you with the U3, U4, U5 side up and the connectors facing you. Solder the hairpin of a prepared wire to pin AD13 of the righthand module. Solder the short end of the same wire to pin AD13 of the lefthand module. Continue soldering hairpins and short ends to the remaining sixteen pins to the left of AD13. (The three pins to the right of AD13 will not be used.)

At the end of each of the wires coming from GND, R/W, OD, AD11, AD12, and AD13, fasten a small piece of masking tape with the wire identification written on it.

Dress the wires neatly out to the right and twist them around each other so they form a neat spiral bundle for a couple of inches. This will keep the bundle flexible and compact. Tie a string or tape around the bundle to hold it together while you work on the other side.

Turn the assembly over and solder the remaining twelve prepared wires to pins Vcc, Y0, DME0, D7 through D0, and Vgg on both modules. This time mark the four long wires going to Vcc, Y0, DME0, and Vgg with masking tape. Finish this side the same way you did the first side, and make a separate spiral bundle. Arrange the bundles side-by-side so they are no thicker than the modules.

That completes the module assembly. A good way to protect and insulate it is to cut two pieces of plastic packaging tape measuring approximately 1½ by 2½ inches, and place one on each side of the assembly.

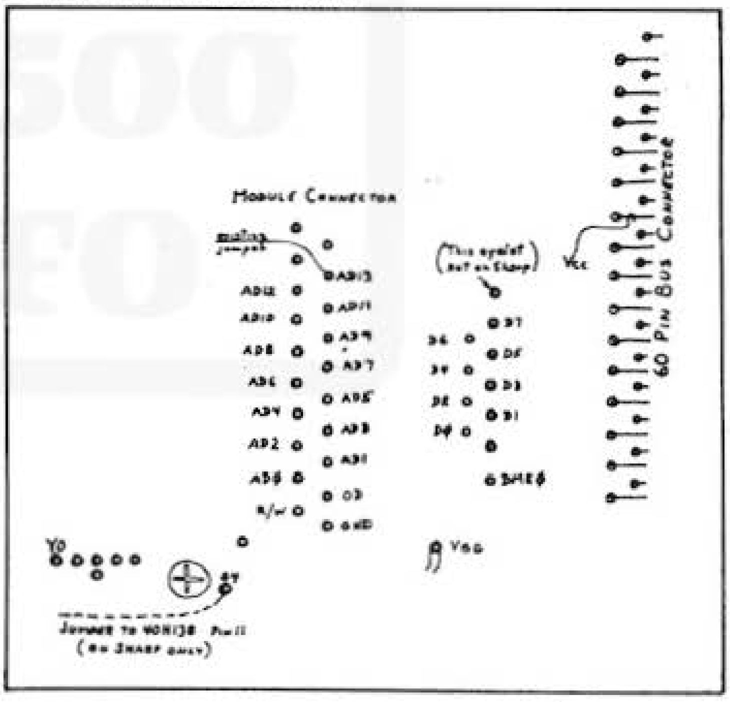
**Connect the Assembly to the Computer**

Notice that you were not directed to mark the individual data wires nor the address wires AD0 through AD10. Any data wire from the assembly can be connected to any data eyelet in the computer in mixed order. The same is true for the address lines, except for the three highest AD11, AD12, and AD13. These latter three should go to their corresponding points in the computer to facilitate troubleshooting if necessary.

Position the assembly over the CPU and ROM with the wire bundles on the side away from the flex circuit connections as shown in the photograph. Temporarily fasten it there with masking tape.

Start with the lower bundle of wires. Route each wire as neatly as possible, cut it to the right length, and strip 1/8 inch of insulation from the end. Then solder it to the proper eyelet, as shown in the accompanying photograph and diagram. Start with DME0, Vgg, and Y0, then the eight data wires D0 through D7, and finally connect Vcc to the wire on the sixty pin connector.

Diagram *Connections to CPU Board*



Connect the wires from the upper bundle next. Starting with GND, R/W, and OD, cut, strip, and form a small loop around the appropriate pin. Squeeze the loop with needle nose pliers to hold it in place while you solder it. Next, connect the address lines AD0 through AD10 in any order. Finally, connect the last three address lines AD11, AD12 and AD13 to the proper pins. Pin AD13 already has one jumper connected to it. Be sure it stays in place when you solder your wire.

Remove the masking tape which temporarily held the assembly in place. Make sure nothing will be scraped or pinched as you reassemble the computer. There will be a very light pressure required as you push the two halves back together. This is the flex circuit pushing down on the assembly and will not cause any problem.

**Checkout the Installation**

After NEW0, MEM should now read 18234. PEEK &7863 should give 0. STATUS 3 should show &4800 (18432).

You may now plug in the modified 8K module that you prepared earlier. After NEW0, MEM should read 26426. PEEK &7863 should show 0 and STATUS 3 should yield &6800 (26624).

With the 8K plug-in module installed, and including the 1-½K of permanent memory and 16K of ROM, you now have 43-½K of memory or 51-½K if the printer-plotter is connected. So the next time someone asks how much memory your "calculator" has, I guarantee you'll see a surprised reaction when he finds out. You may have much more in a one-pound package than he has in his desktop computer.

**Using the Expanded Memory**

Entering NEW0 always sets the SOB (Start of BASIC) pointer at &C5 bytes above the lowest memory address (leaving 197 bytes for the soft keys). After you have installed this 16K memory expansion, NEW0 will set SOB to &00C5.

Most programs can be entered or CLOADed without regard to where the SOB pointer is set. However, if some of the programs you now use or acquire in the future address specific memory locations, but have not been written to be relocatable, then they may not run properly. For example, if a line renumbering program expects to read the value of the first BASIC line number of a program by using a statement such as:

```
10 N=PEEK &38C5 * 256 + PEEK &38C6
```

or some variation of this, then it will be looking at the wrong address in a computer that has this 16K memory addition installed.

In general, any time a program has statements containing PEEK, POKE, or CALL, you should examine it to see whether the author allowed for different memory starting addresses.

If the program is not relocatable, you can always run it by entering NEW followed by either &40C5 or &38C5, whichever is appropriate for the program, before you load it. This will cause the computer to effectively ignore the added memory.

A much better solution is to write new programs and to modify existing ones so that they are relocatable. One way to do this is by utilizing address &7863. This address (after a hard reset) always contains the upper byte of a two byte value equal to the lowest address available in the PC. For instance, in an unexpanded PC-1500, PEEK &7863 will show &40, indicating that the lowest memory address is &4000. After the 16K expansion, PEEK &7863 will give 0 because memory starts at &0000.

You can make a program relocatable by substituting PEEK &7863 for &40 or &38, wherever they occur. Thus, &38C5 becomes:

```
PEEK &7863 * 256 + &C5
```

Line 10 in the example could be replaced by:

```
10 S=PEEK &7863 * 256 + &C5
11 N=PEEK S * 256 + PEEK (S+1)
```

or, if you must put it all on one program line, it could be written:

```
10 N=PEEK (PEEK &7863 * 256 + &C5) * 256 + PEEK (PEEK
   &7863* 256 + &C6)
```

This method will always work if the computer is initialized with a NEW0. However, it may not work if some area of memory has been set aside for another purpose by using NEW followed by some value other than 0.

A more general method of making a program completely relocatable is to use the SOB pointer as a reference. You can always obtain the value of SOB by:

```
SOB=PEEK &7863 * 256 + PEEK &7866
```

or by:

```
SOB=STATUS 2 - STATUS 1.
```

Line 10 in the example could thus be replaced with:

```
10 S=PEEK &7865 *256 + PEEK &7866
11 N=PEEK S * 256 + PEEK (S+1)
```

or by:

```
10 S = STATUS 2 - STATUS 1
11 N = PEEK S * 256 + PEEK (S+1)
```

This is not as awkward or extravagant with memory as it might at first appear, because the line to calculate SOB (line 10) has to be used only once in the entire program. Thereafter, the program can simply refer to S whenever SOB is needed. This will probably shorten the overall program.

When this latter method is used to make the program relocatable, any legal value can be entered following NEW and the program will load and run properly.

If program authors will begin using this technique, they will not have to specify special instructions on how to load their programs, according to how much memory a user has installed. They can merely specify the minimum amount of RAM required by their programs. Then the programs themselves can take care of any variations in memory configuration. The programs could even automatically check to see how much memory was installed, and notify the user if there was not enough!

**For the Future...**

You may have noticed that U5 in the plug-in module now serves no purpose. It could be removed altogether. Perhaps someone will want to figure out how to put a tiny battery in that space, so that a module could be removed, put on a shelf, then reinstalled without losing its memory.

Having this much memory will make you painfully aware of just how slow the cassette interface is. It takes about forty minutes to load or save the entire memory. I hope someone discovers the secret of changing the baud rate. And wouldn't a three-inch disk drive be a perfect match?

Another possibility as the price of modules comes down would be to build two more 16K assemblies, mount them in the printer, and address them as &0 to &7FFF in the alternate address range. Programs could be moved from there to main memory as needed.

This impressive memory enhancement article was designed and described by: *Don. L. Carter, Boeing JW-26, 220 Wynn Drive, Huntsville, AL 35805.*

---

## MEMORY MAPS

The Sharp PC-1500/Radio Shack PC-2 come standardly equipped with 2K of RAM (random access memory) at the hexadecimal addresses &4000 - &47FF. Part of this block is set aside for use by the Reserved Softkeys. The remainder is available for user programs.

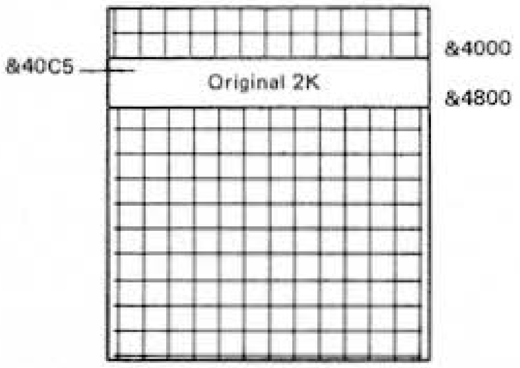
If you plug in a CE-151 (or equivalent) 4K memory module into the back of the PC, you will find that the new memory is addressed in the range &4800 - &57FF. No surprises there as the memory block simply picks up where the original 2K block ends.

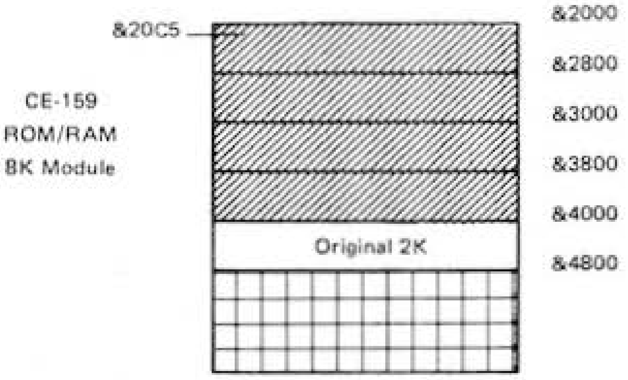
However, if you plug in a CE-155 8K memory module, you may be a little surprised to learn the following: the module effectively splits its memory into two separate address ranges to surround the 2K block provided in the basic unit! A 2K section of the module occupies the address range &3800 - &3FFF. Then, a 6K block picks up at &4800 and extends through &5FFF. The operating system provided in the PC's ROM is able to tell where RAM begins and ends. Thus, when a CE-155 module is installed, the PC changes pointers so that the area set aside for the Reserved Softkeys starts at the address &3800 instead of &4000.

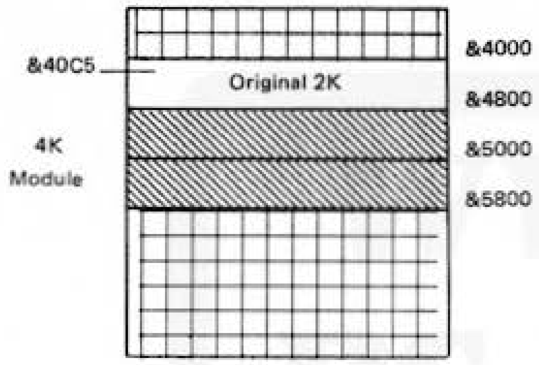
In the memory expansion article provided in this issue of *PCN*, Don Carter shows how a CE-155 memory module can be modified so that it operates as one continuous 8K block accessed as addresses from &4800-&67FF. This is done so that the 16K that is to be installed inside the PC will not overlap in the range &3800 - &3FFF. Again, the operating system of the PC is able to determine where RAM begins and to set aside an area for the Reserved Softkeys accordingly.

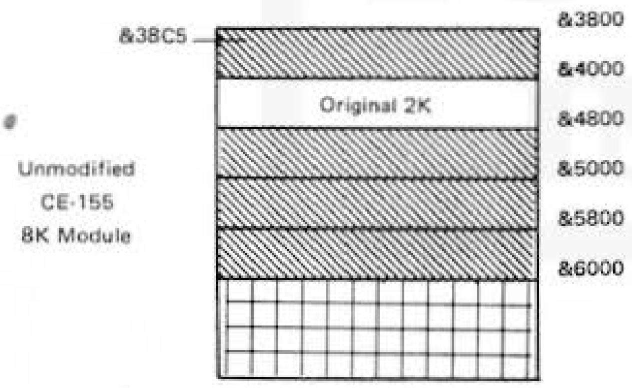
Sharp Electronics, however, has sprung another surprise with its CE-159 memory module. This is a special battery-equipped 8K module that can be configured to operate effectively as ROM or partitioned to operate partially as RAM and partially as ROM. Want to guess at the address range for which this module was designed? It is not the same as that of the CE-155, which one might presume. No, it uses the range &2000 - &3FFF!

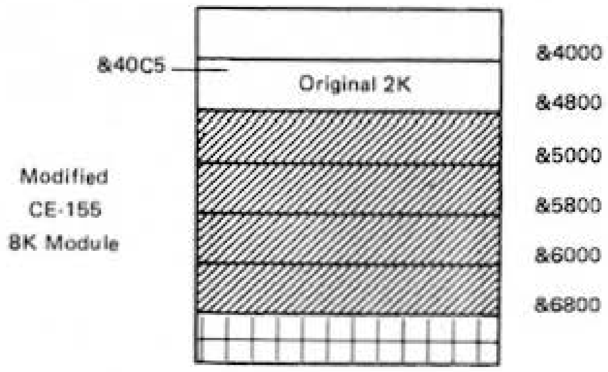
Figure *Memory Maps*











Now, what difference does it make as to what address range the various memory expansion modules might utilize? Well, if you only program in BASIC it will not make any difference. If you start using machine language routines, however, it can make a big difference. Many machine language programs are address dependent. For instance, the machine language portion of Norlin Rober's monitor program published in Issue 22 of *PCN* is designed to reside specifically in the addresses &3E4F - &3FFF. Once installed in memory at those addresses, the machine language instructions must be protected from interference by future BASIC programs, accomplished by using the NEW&4000 directive. The memory is effectively partitioned so that all BASIC statements are stored above the address &4000 and thus above the machine language routines.

Note that when this is done with a CE-155 module installed in the system, there will still be about 8K of RAM available for BASIC because the RAM elements extend through address &5FFF. Alas, however, if one performs the same directive with a CE-159 module installed, then a mere 2K of memory is left for BASIC! This is because all of memory in that module is addressed below &4000, leaving just the original 2K of PC memory available once the partitioning NEW&4000 directive has been given!

The upshot of these addressing differences among the various memory expansion modules is that users planning on employing address-specific machine language routines will have to plan their module purchases carefully.

## EXPANDED DISPLAY ROUTINE

The accompanying program uses the graphics capability of the PC-2 to increase the display from 26 to approximately 40 characters. The message can be up to 80 characters long in which case it will wrap to the beginning of the screen. The message is placed in variable M$(1) and the extended display is invoked by GOSUB 100. ASCII characters 32 to 90 are covered. This includes A-Z, 0-9, and most of the special characters. Characters outside this limit, such as lower case, will cause an error. Numeric information must be in alpha format so STR$ X should be used to insert numeric data into the string.

Statements 1, 10 set up the graphic matrix. 100, 105 is the expanded display routine. 1000-1120 contains the graphic data (but can be put anywhere at the end — or in a preprocessor that loads the matrix and then CHAINs in the program containing the expansion subroutine). The sample has a simple input routine at 500, 510 but the actual program can be placed where desired with the appropriate adjustment to statement 10.

This program submitted by: *H. David Jackson, 126 Smithfield Drive, Endicott, NY 13760.*

## FILE MAINTENANCE PROGRAM

The file maintenance program allows the setup of a filing system to a user's specific needs with regard to record size, number of fields, and field length. The complete file is memory resident for purposes of access. The record size is dependent upon the number of fields and their length. Once a file has been created, records may be added, changed or retrieved and the file may be saved on cassette. The program was developed on the Radio Shack PC-2, and I suspect it will run on the Sharp PC-1500. Using an 8K memory module, the following conditions are possible:

| | |
|---|---|
| Record Length | 80 Bytes Max |
| Number of Records | 80 Max (if record length is 80) |
| Fields per Record | 20 Max |
| File Name | 16 Bytes Max (fixed) |
| Field Name | 16 Bytes Max (fixed) |
| Field Length | 80 Bytes Max |
| Field Search | 5 Fields Max |

To run the program, execute a run command and a continuous menu will be displayed. To select a menu item, a single key stroke of the appropriate letter or number is all that is necessary. The program prompts the user and validates all inputs. An invalid input will cause a return to the prompt. Capacity overflows are indicated by error messages. The execution of the menu subroutine executes a clear command; therefore, using the Break key and restarting the program will erase any previous files and data in memory. Upon the completion of a menu function, the menu will be displayed. The following menu functions are available:

S – Setup new file
O – Output file
I – Input file
1 – Print file parameters
A – Add a record
P – Print a record
F – Find record(s)
C – Change a record

A delete function was not incorporated in order to conserve program space. If a record is created and is no longer needed, label one of its fields deleted. When a new record is desired, use the Find function to locate the record and the Change function to edit its contents to the new data.

If peripheral programs are to be written on the data base, the following file information will be helpful:

| | | |
|---|---|---|
| F$ | = | File name |
| N$ (array) | = | Field names |
| R$ (array) | = | Records |
| L (array) | = | Length of fields |
| F | = | Number of fields in record |
| R | = | Number of records in file |
| X | = | Length of record |
| U | = | Number of records used |

This program submitted by: *Stephen Tomback, 19 Maplewood Way, Pleasantville, NY 10570.*

Program *File Maintenance*

```
  1:REM File maint
    enance
  2:REM FM 10/11/8
    2  12/04/82
 10:REM MENU
 15:CLEAR :LOCK :
    DIM U$(0)*80
 20:CLS :WAIT 0:E=
    8:K=0:RESTORE
    :FOR I=1TO E:
    READ U$(0):
    PRINT U$(0)
 25:FOR J=1TO 60:U
    $=INKEY$ :IF U
    $<>""LET J=60
 30:NEXT J
 35:IF U$=""THEN 5
    5
 40:FOR J=1TO E:IF
    MID$ ("SOI1APF
    C",J,1)=U$LET
    K=J:J=E
 45:NEXT J
 50:IF KCLS :ON K
    GOSUB 100,400,
    500,600,700,85
    0,900,1000:
    GOTO 20
 55:NEXT I:GOTO 20
 60:DATA "S - SETU
    P NEW FILE","O
     - OUTPUT FILE
    ","I - INPUT F
    ILE"
 65:DATA "1 - PRIN
    T FILE PARAMET
    ERS","A - ADD
    A RECORD","P -
     PRINT A RECOR
    D"
 70:DATA "F - FIND
     RECORD(S)","C
     - CHANGE A RE
    CORD"
100:REM NEW FILE
105:CLEAR :INPUT "
    NEW FILE NAME:
     ";F$
110:IF LEN F$<1
    THEN 105
115:INPUT "HOW MAN
    Y RECORDS: ";R
120:IF R<1OR R>255
    THEN 115
125:INPUT "FIELDS
    PER RECORD: ";
    F
130:IF F<1OR F>20
    THEN 125
135:DIM N$(F-1):
    DIM L(F-1)
140:FOR I=1TO F
145:CLS :PRINT "FI
    ELD";I;" NAME:
    ":CURSOR 14:
    INPUT "";N$(I-
    1)
150:IF LEN N$(I-1)
    <1GOTO 145
155:CLS :INPUT "HO
    W MANY BYTES:
    ";L(I-1)
160:IF L(I-1)=0
    THEN 155
165:IF X+L(I-1)<81
    LET X=X+L(I-1)
    :GOTO 175
170:PRINT 80-X;" B
    YTES LEFT":
    BEEP 15:GOTO 1
    55
175:NEXT I
180:IF STATUS 3-
    STATUS 2-R*X-1
    81<1THEN 190
185:DIM R$(R-1)*X:
    R$(0)="":DIM I
    $(0)*80:DIM U$
    (0)*80:RETURN
190:PRINT "NOT ENO
    UGH MEMORY":
    BEEP 15:GOTO 1
    05
400:REM OUTPUT
405:IF LEN F$=0
    GOSUB 2000:
    RETURN
410:INPUT "TAPE RE
    ADY? Y to outp
    ut ";I$
415:IF I$<>"Y"THEN
    410
```

Program *File Maintenance (conclusion)*

```
420:PRINT #F$;F,R,
    X,U
425:I$=F$+"*"
430:PRINT #I$;N$(*
    ),R$(*),L(*):
    RETURN
500:REM INPUT
505:CLEAR :INPUT "
    INPUT FILE NAM
    E: ";F$
510:INPUT #F$;F,R,
    X,U
515:DIM N$(F-1):
    DIM R$(R-1)*X:
    DIM L(F-1)
520:I$=F$+"*"
525:INPUT #I$;N$(*
    ),R$(*),L(*)
530:DIM I$(0)*X:
    DIM U$(0)*X:
    RETURN
600:REM PARAMETERS
605:IF LEN F$=0
    GOSUB 2000:
    RETURN
610:TEXT :CSIZE 1:
    LF 5
615:LPRINT "FILENA
    ME : ";F$
620:LPRINT "NO OF
    RECORDS :";R
625:LPRINT "RECORD
    S USED :";U
630:LPRINT "BYTES
    PER RECORD :";
    X
635:LPRINT "MEMORY
     LEFT :";
    STATUS 3-
    STATUS 2:LF 1
640:FOR I=1TO F:
    LPRINT "FIELD"
    ;I;" ";N$(I-1)
    ;TAB 28;USING
    "###";"BYTES";
    L(I-1):USING
645:NEXT I:LF 6:
    RETURN
700:REM ADD
705:IF LEN F$=0
    GOSUB 2000:
    RETURN
710:FOR I=0TO R-1:
    REM FIND 1ST A
    VAILABLE RECOR
    D
715:IF LEN R$(I)=0
    THEN 725
720:NEXT I:PRINT "
    NO MORE RECORD
    S!":BEEP 15:
    RETURN
725:FOR J=0TO F-1
730:CLS :I$(0)="":
    PRINT N$(J);":
     ":CURSOR LEN
    N$(J)+2:INPUT
    "";I$(0)
735:IF LEN I$(0)>L
    (J)THEN 760
740:IF LEN I$(0)=L
    (J)THEN 750
745:I$(0)=I$(0)+"
    ":GOTO 740
750:R$(I)=R$(I)+I$
    (0):I$(0)=""
755:NEXT J:U=U+1:
    CLS :PRINT "AD
    D RECORD";U:
    BEEP 15:RETURN
760:GOSUB 3000:
    GOTO 730
800:REM OUT TO PRI
    NTER
805:TEXT :CSIZE 1:
    LF 1:J=0
810:LPRINT "RECORD
    :";N:LF 1
815:FOR I=0TO F-1
820:IF I=0LET J=1
825:LPRINT N$(I);"
    : ";MID$ (R$(N
    -1),J,L(I)):J=
    J+L(I)
830:NEXT I:LF 1:
    RETURN
850:REM PRINT
855:IF U=0GOSUB 20
    00:RETURN
860:N=0:INPUT "REC
    ORD NUMBER: ";
    N
865:IF N<1OR N>U
    THEN 860
870:GOSUB 800:LF 5
    :RETURN
900:REM  FIND
905:IF U=0GOSUB 20
    00:RETURN
910:CLS :Q=0:INPUT
    "HOW MANY FIEL
    D SEARCH? ";Q
915:IF Q<1OR Q>5OR
    Q>FTHEN 910
920:RESTORE 990:
    FOR I=1TO Q:
    READ W$:Z$=
    STR$ I
925:CLS :@(I)=0:
    CURSOR :WAIT 0
    :PRINT Z$;W$:
    CURSOR 4:INPUT
    "FIELD NUMBER:
     ";@(I)
930:IF @(I)<1OR @(
    I)>FTHEN 925
935:CLS :@$(I)="":
    CURSOR 0:PRINT
    "SEARCH";@(I);
    " FOR: ":
    CURSOR 14:
    INPUT "";@$(I)
940:IF LEN @$(I)>L
    (@(I)-1)GOSUB
    3000:GOTO 935
945:IF LEN @$(I)<1
    THEN 935
950:NEXT I
955:FOR P=0TO U-1:
    FOR I=1TO Q
960:CLS :PRINT "RE
    CORD";P+1:BEEP
    1
965:S=@(I):GOSUB 4
    000
970:T=LEN @$(I)
975:IF MID$ (R$(P)
    ,Z,T)<>@$(I)
    NEXT P:RETURN
980:NEXT I
985:N=P+1:LF 2:
    GOSUB 800:NEXT
    P:RETURN
990:DATA "st","nd"
    ,"rd","th","th
    "
1000:REM CHANGE
1005:IF U=0GOSUB
     2000:RETURN
1010:N=0:INPUT "R
     ECORD NUMBER
     : ";N
1015:IF N<1OR N>U
     THEN 1010
1020:P=0:INPUT "F
     IELD NUMBER:
      ";P
1025:IF P<1OR P>F
     THEN 1020
1030:I$(0)="";
     INPUT "CHANG
     E TO: ";I$(0
     )
1035:IF LEN I$(0)
     >L(P-1)THEN
     1050
1040:IF LEN I$(0)
     =L(P-1)THEN
     1055
1045:I$(0)=I$(0)+
     " ":GOTO 104
     0
1050:GOSUB 3000:
     GOTO 1030
1055:IF P=FTHEN 1
     080
1060:S=P+1:GOSUB
     4000:T=X-Z+1
1065:U$(0)=MID$ (
     R$(N-1),Z,T)
1070:I$(0)=I$(0)+
     U$(0)
1075:IF P=1LET R$
     (N-1)=I$(0):
     RETURN
1080:S=P:GOSUB 40
     00:R$(N-1)=
     MID$ (R$(N-1
     ),1,Z-1)+I$(
     0):RETURN
2000:PRINT "MEMOR
     Y EMPTY!":
     BEEP 15:
     RETURN
3000:CLS :PRINT "
     FIELD LENGTH
      EXCEEDED!":
     BEEP 15:
     RETURN
3336:+6701
4000:REM START PO
     S OF FIELD I
     N R$
4001:REM  ENTER/F
     IELD # IN R$
4002:REM  EXIT/PO
     S IN Z
4005:Z=1:IF S=1
     RETURN
4010:FOR K=1TO S-
     1:Z=Z+L(K-1)
     :NEXT K:
     RETURN
STATUS 1        3256
```

/here probabilities are less than 1E-7, the calculated results are ieaningless.

The routine used in the accompanying machine language program ives a full ten digits of accuracy, regardless of the value of Z. (When : exceeds 21.165, the probability is less than 1E-99, so the result nderflows to zero.) Probabilities are calculated with a Taylor series or absolute values of Z less than 1.6, and with a continued fraction or larger values of Z. Execution is fast, taking less than a second.

You can enter the machine language program using a Monitor pro-ram or by using BASIC POKEs. (The program is *relocatable*, so if you /ish you may start it at some address other than &38C5 without iaking any modifications. Just remember to call the right address when ccessing the routine if you do move it elsewhere.)

To use the program, just follow these steps:

1. The value of Z to be used must be assigned to the variable Z.
2. Execute CALL &38C5 (or the starting address if you relocate the machine language routine).
3. Display the variable P to obtain the probability that a normally distributed random variable exceeds Z.

As a bonus, the variable O contains the ordinate of the normal urve for the given Z!

You can test the program's operation (to verify proper loading of he machine language codes) with the following examples:

| Value of Z | Resulting P | Resulting O |
|---|---|---|
| -2 | 9.772498681E-01 | 5.399096651E-02 |
| 1.3 | 9.680048459E-02 | 0.171368592 |
| 4 | 3.167124183E-05 | 1.338302256E-04 |

## ROUTINE DELETES LINES FROM BASIC PROGRAMS

*ohn Norton, % Pencept, Inc., 39 Green Street, Waltham, MA 02154,* ubmitted this program that makes it easy to remove groups of lines rom a BASIC program. Here is what John has to say about his routine:

I have often wished for a simple way to eliminate sections of a 3ASIC program in a Sharp PC-1500/Radio Shack PC-2. Trying to emove a number of lines by specifying each line number is just too low, cumbersome and subject to error. By capitalizing on the material hat has appeared in *PCN*, I have been able to create a routine that will ermit the deletion of a group of lines from a BASIC program.

To use this routine, it must be loaded into memory prior to your levelopment of a BASIC program. Your BASIC program is then created starting with a line number greater than 14 as this is the highest line iumber in the deletion routine). The routine uses a JUMP statement o that a RUN directive causes the PC to skip over it and execute the iser's program.

The deletion routine may be utilized at any time by pressing the )EFine and D keys. It begins by prompting for two items: the line iumber at which to begin deleting and the last line number to be leleted. If, for example, you had a program starting at line 500, with a ast line number of 850, and you wanted to delete all the lines from iumber 722 through 751, you would proceed as follows:

| What you type | What the display shows |
|---|---|
| DEF/D (ENTER) | DELETE FROM: |
| 722 (ENTER) | DELETE TO: |
| 751 (ENTER) | WORKING . . . |

That is all there is to it! After these inputs, the routine will issue a ›eep each time it encounters a line less than 722. It will signal when t has found line 722, when it finds line 751, and when it does the leletion of the material between those lines.

The routine operates as follows: It marks the memory location it which line 722 starts. It then re-writes every byte from the first ine after 751 to the end of the program, into memory starting from he marked byte (where line 722 started).

It is important to note the following: The routine will not attempt o delete itself. It does this be starting to look for lines in memory ifter itself. It does this by "knowing" exactly how long it is. Thus, he routine should not be altered by as much as a single character! Make sure you load it exactly as shown in the accompanying listing!

The routine uses default values as follows: If no value is given in response to the prompt "DELETE TO", then the "DELETE FROM" value is used. Thus, to delete a single line from memory, just type: DEF/D ### (ENTER) (ENTER), where ### represents the line number to be deleted.

In the event that the starting line is specified as larger than the ending line, the starting line is not found or the ending line is not located, the routine will display an error message such as "START NOT FOUND" or "END NOT FOUND" and cease operating.

To delete the routine itself from memory, type an asterisk (*) immediately before entering DEFine/D. This will cause the computer to restructure pointers so that the deletion routine is effectively erased!

Program *Line Deletion*

```
 1:GOTO 16
 2:"D":AREAD DP$:
   IF DP$="*"LET
   D1=59:D2=149:
   GOTO 14
 3:CLEAR :WAIT 0:
   INPUT "DELETE
   FROM:";DF:DT=D
   F:INPUT "DELET
   E TO:";DT:
   PRINT "WORKING
   .."
 4:DM=256*PEEK &7
   865+PEEK &7866
   +720:DB=DM:IF
   PEEK DM=255LET
   DM=DM+1:DB=DM
 5:DN=256*PEEK DM
   +PEEK (DM+1):
   IF PEEK DM=255
   OR DN>DFBEEP 5
   ,60:WAIT :
   PRINT "NO STAR
   T":END
 6:BEEP 1,40:IF D
   N<DFLET DM=DM+
   PEEK (DM+2)+3:
   DB=DM:GOTO 5
 7:BEEP 5,30:
   PRINT "START F
   OUND"
 8:IF DN=DTLET DM
   =DM+PEEK (DM+2
   )+3:GOTO 11
 9:IF PEEK DM=255
   OR DN>DTBEEP 5
   ,60:WAIT :
   PRINT "NO END"
   :END
10:BEEP 1,20:DM=D
   M+PEEK (DM+2)+
   3:DN=256*PEEK
   DM+PEEK (DM+1)
   :GOTO 8
11:BEEP 5,10:
   PRINT "...DELE
   TING":DE=256*
   PEEK &7867+
   PEEK &7868:FOR
   DI=0TO DE-DM:
   BEEP 1,0
12:POKE DB+DI,
   PEEK (DM+DI):
   NEXT DI:POKE &
   7867,INT ((DB+
   DI)/256),((DB+
   DI)AND &FF)
13:FOR DI=20TO 0
   STEP -1:BEEP 1
   ,DI:NEXT DI:
   END
14:D2=D2+((PEEK &
   7865<>PEEK &78
   69)OR (PEEK &7
   866<>PEEK &786
   A))
15:POKE &7865,D1,
   D2:POKE &7869,
   D1,D2:END
16:REM *MAIN PROG
   RAM*
STATUS 1          721
```

(Model 100 . . . continuation from page 1)

a "SIG" (Special Interest Group) devoted exclusively to Model 100 enthusiasts. In the first two weeks following its activation, some 1600 users have joined this SIG in order to use the Model 100-related information base. The Compuserve Model 100 SIG (which can be accessed by Compuserve users typing GO PCS-154 at the command (!) prompt) has already had some 30 programs contributed to its user library. These range from several simple "demo" programs on up to useful utilities such as memory dumps, disassemblers, printer formatting

## THE CE-502A GENERAL STATISTICS MODULE

This plug-in ROM module for the Sharp PC-1500/Radio Shack PC-2 contains 16K of statistical programs, all in BASIC, located in memory in the addresses zero to &3FFF.

Since the module is plugged into the slot that is also used by the RAM expansion modules (such as the CE-151, CE-155 or CE-159), only the 2 kilobytes of unexpanded RAM is left available for user programs when the CE-502A is installed. A user program in RAM must begin with an executable label. This is because such a program is separated from the ROM module by the equivalent of a MERGE. It should also be noted that REServe memory is taken over by the CE-502A ROM module.

The module contains seven programs:

1. Means and Moments
2. t-Test, Paired
3. t-Test, Unpaired
4. One-Way ANOVA
5. Two-Way ANOVA
6. Contingency Table
7. Ranked Sum Test

Execution of any of these may be initiated by use of a REServe key. The correct REServe key can be conveniently identified by displaying the REServe memory template.

To get an idea of the amount of flexibility provided by the programs in this module, consider the options given the user in the Means and Moments program. Data may be entered either from the keyboard or from cassette. If the printer is connected, a choice is offered as to whether input data (and output) are to be printed. Inputs may be in the form of either grouped or ungrouped data. Editing of data is permitted. Data can be recorded on tape and a backup copy made. The second moment (variance) may be calculated with either n or n-1 used in the denominator.

The outputs provided by this program include the arithmetic, geometric and harmonic means, the second, third and fourth moments about the mean, the skewness and kurtosis and the number of entries made. However, it will not give you the standard deviation.

The t-tests for paired and unpaired data are used for determining the level of significance of the difference between measurements taken from two populations. The use of the test for differences between population means (the unpaired t-test) is based on the assumption that the two populations are normally distributed, with equal variances. Again, there are options for printing or not printing, entering data from the keyboard or cassette tape, inputting grouped or ungrouped data and recording data onto tape. The computer requests input of the hypothesized difference in means, with a default value of zero. The output includes the value of t, but the program does not calculate the level of significance. You will need to use tables for that. The number of degrees of freedom, together with the number of x entries and y entries, is also given.

The unpaired t-test (also known as the paired-difference test) requires only the assumption that the *differences* are normally distributed. The outputs here include the mean and standard deviation of the differences, the value of t and the number of degrees of freedom. Again, however, you will need to use a t-distribution table to interpret the results.

The programs for one-way and two-way analysis of variance (usually shortened to "ANOVA") provide for a variety of ways of handling the input data. Outputs include all necessary information, including the value of the F statistic. The use of an F-distribution table is required to interpret the results.

The two remaining programs involve nonparametric tests. The contingency table program gives the user the usual cassette options involving data. In this program there is a pleasant surprise. Not only is the value of chi-squared calculated, but its probability is determined too! This avoids the necessity of having to check a table to determine the level of significance. This is particularly useful in view of the fact that almost all chi-squared tables contain only certain selected values.

The final program, the Mann-Whitney Ranked Sum test, provides a way of comparing two means *without* the need for the assumptions required by the t-test. A Shell sort incorporated within the program is used to rearrange the input values, from each of the two samples, into numerical order. (It is possible to display or print the rankings after they have been sorted.) Outputs include the rank sum, rank mean and rank variance for both samples. The test statistics U (indicated as w) and z are given, but you will have to provide your own tables again. (Although the module contains a routine for determining approximate values of normal probability in lines 28205 — 28340, for some strange reason it is not used by *this* routine!)

There are many statistical tests in existence. The writers of the programs contained in this module had to make some choices. They wisely decided to omit some of the common tests that require a relatively small amount of computation, such as tests of hypotheses and determination of confidence intervals involving means, variances and proportions for single populations. Some users, however, might like to have seen inclusion of the chi-square test, linear regression and correlation. But, even 16K of memory can not hold everything, particularly when so many bells and whistles are included in the package.

One nice feature used in all of the programs in this module is a flashing of the default value in the display when yes or no inputs are requested. For example, the display of:

```
RECORD DATA (Y/N)?
```

is shown with the letter N flashing on and off. This tells the user that if ENTER is pressed without an input having been made, the computer will interpret the response as having been N for no! This feature is easy to get used to and it is convenient.

I would rate the manual as excellent. Although such a manual obviously can not include an entire course in statistics, the writers did take the trouble to include at least a brief explanation of what each of the tests is about. The manual ends with a list of examples of applications for which each of the tests in the module would be appropriate.

The manual documents the programs well. There are listings of the variables used by each program. In many cases the formulas used are also provided. This manual was obviously written in the U.S.A. It is not an atrocious translation from some other language!

I have not found any real bugs in the program, but there is a matter that comes pretty close to qualifying as such. The choice of algorithms used and the way in which they are implemented can produce some inaccurate results in certain circumstances.

The results frequently obtained for moments M3 and M4, as well as those for skewness and kurtosis, appear to be the worst. To illustrate: If the data values 865, 866 and 867 are entered, the following results are obtained:

| RESULT | GIVEN BY PROGRAM | CORRECT VALUE |
|---|---|---|
| M2 | 1.00005333 | 1 |
| M3 | 0.10000000 | 0 |
| M4 | -37.0370370 | 0.66666666 |
| Skewness | 0.09999200 | 0 |
| Kurtosis | -37.0330867 | 0.66666666 |

A negative value for kurtosis is, of course, impossible. Note that the second moment also is not extremely accurate.

One cause of the errors is that during the accumulation of the sums $\Sigma X^2$, $\Sigma X^3$ and $\Sigma X^4$, the 10-digit capacity of the computer places limits on the accuracy of the sums. Then the required subtraction of nearly equal quantities having small relative errors produces a difference value having a large relative error.

About all you can do about this problem is code your data. Unfortunately, no provision appears to have been made for this in the program. (In the example presented, if you subtract 866 from each of the given input values and use the results as inputs, you will get correct values for M2, M3, M4, skewness and kurtosis.)

It seems to me that a better solution would have been for the programmer to have used another algorithm. The mean, $\bar{X}$, could have been calculated first. Next the numerator of M2 could have been calculated as $\Sigma(X-\bar{X})^2$ rather than as $\Sigma X^2 - (\Sigma X)^2/N$. (With this approach it would not have been possible to have accumulated the combined data from several tape files. There would have been ways to have gotten around that too, but it would have been rather complicated.) Similar approaches to the calculation of M3 and M4 would have eliminated the significant errors noted in the illustration.

There is another inaccurate procedure used in these programs. The use of exponentiation is detrimental to the accuracy of the t-test and ANOVA programs as well as the Means and Moments program. Using, for example, X^2 in place of X*X has two bad effects: it is much slower and there is a loss of accuracy. The exponentiation operation uses logarithms to calculate an *approximate* result. This often contains slight inaccuracies. These inaccuracies are greatly magnified when subtraction of nearly equal quantities is required. That is exactly what happens in many statistical calculations. For instance, the error in M2 in the earlier example is entirely due to the use of X^2 in the program.

The programs could have been designed to operate somewhat faster. Part of the blame for a loss of speed has to go to the use of the exponentiation operation. The computer's response sometimes seems irritatingly slow when YES/NO inputs are required. While the sorting routine used in the Mann-Whitney test is quite efficient, a machine language sort would have really moved things along.

I have one more rather minor complaint. In four of the programs, data inputs are first placed into a string variable. Their numerical values are then later determined by use of the VAL function. Besides slowing down operation somewhat, this prevents one from using expressions such as 2/3, π or 968-750 as inputs. As a consequence, if you want to code input data, it is not nearly as easy to do. Their reason for using this approach was apparently to permit the input of E to signal the end of data. It would not have required a great deal of imagination to have achieved such a result another way.

However, my overall assessment of the CE-502A module is that it is a powerful and useful package for anyone doing much analysis of statistical data. Just be sure to take the skewness and kurtosis results with a grain of salt!

This review presented by: *Norlin Rober.*

Program *RPN Octal–Decimal–Hexadecimal Calculator*

```
100:CLS :CLEAR :A$="0123456789ABCDEF":BA=16:BB=58:U=33:T$="&":WAIT 0
110:ON ERROR GOTO 730:GOTO 600
120:X$=""
130:K$=INKEY$ :IF K$=""THEN GOTO 130
140:BEEP 1
150:K=ASC K$:IF (K>47AND K<BB)OR (K>64AND K<71AND BA=16)THEN LET X$=X$+K$:CLS :PRINT T$+X$:GOTO 130
160:IF X$=""THEN GOTO 240
170:IF X<>0THEN LET T=Z:Z=Y:Y=X
180:IF BA=10THEN LET X=VAL X$:GOTO 230
190:L=LEN X$:X=0:BT=1
200:FOR M=0TO L-1:O$=MID$ (X$,L-M,1):P=VAL O$
210:IF P=0THEN LET P=ASC O$-55:IF P<0THEN LET P=0
220:X=X+P*BT:BT=BT*BA:NEXT M
230:PRINT T$+X$,X:GOTO 250
240:IF K=13THEN LET T=Z:Z=Y:Y=X:GOTO 120
250:IF K=13AND X=0THEN LET T=Z:Z=Y:Y=X:GOTO 120
260:IF K=43THEN LET X=Y+X:Y=Z:Z=T:GOTO 610
270:IF K=45THEN LET X=Y-X:Y=Z:Z=T:GOTO 610
280:IF K=42THEN LET X=Y*X:Y=Z:Z=T:GOTO 610
290:IF K=47THEN LET X=Y/X:Y=Z:Z=T:GOTO 610
300:IF K=24THEN LET X$="":X=0:PRINT T$+X$,X:GOTO 120
310:IF K=09THEN LET K=X:X=Y:Y=K:GOTO 610
320:IF K=10THEN LET K=X:X=Y:Y=Z:Z=T:T=K:GOTO 610
330:IF K=11THEN LET K=T:T=Z:Z=Y:Y=X:X=K:GOTO 610
340:IF K=02THEN LET X=-X:GOTO 610
350:IF K=15THEN END
360:IF K=31THEN GOTO 470
370:IF K=01THEN GOTO 500
380:IF K=78THEN LET X=-X-1:GOTO 610
390:IF K=22THEN GOTO 550
400:IF K=79THEN GOTO 570
410:IF K=83THEN LET S=X:GOTO 610
420:IF K=25THEN LET T=Z:Z=Y:Y=X:X=S:GOTO 610
430:IF K=61AND X<0AND BA<>10THEN LET N=UL+X:S$="":GOTO 640
440:IF K=85THEN GOTO 590
450:IF X$=""THEN GOTO 610
460:PRINT T$+X$,X:GOTO 120
470:IF BA=16THEN LET BA=10:BB=58:T$="%":PAUSE "DECIMAL":GOTO 610
480:IF BA=10THEN LET BA=08:BB=56:T$="#":PAUSE "OCTAL":GOTO 610
490:IF BA=08THEN LET BA=16:BB=58:T$="&":PAUSE "HEXADECIMAL":GOTO 610
500:K$=INKEY$ :IF K$=""THEN GOTO 500
510:BEEP 1:K=ASC K$:IF K=11THEN LET X=√X:GOTO 610
520:IF K=32THEN LET X=Y^X:Y=Z:Z=T:GOTO 610
530:IF K=24THEN LET X=0:Y=0:Z=0:T=0:X$="":PRINT T$+X$,X:GOTO 120
540:GOTO 120
550:GOSUB 670:X=(X2AND Y2)*B15*B15+(X3AND Y3)*B15+(X1AND Y1)
560:Y=Z:Z=T:GOTO 610
570:GOSUB 670:X=(X2OR Y2)*B15*B15+(X3OR Y3)*B15+(X1OR Y1)
580:Y=Z:Z=T:GOTO 610
590:PRINT "# BITS=";U;" ";:INPUT U
600:BEEP 1:UL=1:FOR I=1TO U:UL=UL*2:NEXT I:CLS :PAUSE U;" BITS=";UL;" UP LIM":GOTO 610
610:IF BA=10THEN PRINT T$,X:X$="":GOTO 120
620:IF ABS X>UL THEN LET X=SGN X*UL
630:N=ABS X:S$="":
```

program listing continued on next page

## PAPER HOLDER KIT FOR PC-1500 & PC-2

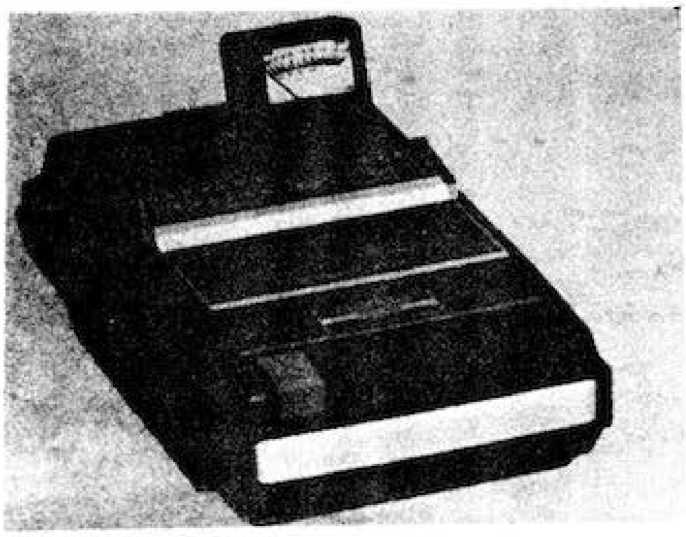
*Bisi Beaver Paper Holder Kit* is an chment for Radio Shack PC-2 and Sharp 500 computers that will accomodate a dard 2-1/4 inch roll of calculator paper. kit includes the paper holder, a template proper alignment and the drilling of holes the printer cover, and instructions for ng the modifications. When not in use the r holder and paper may be stored in the uter's carrying case.

The kit retails in the U.S. for $24.95 (plus 56 sales tax in the state of Washington). s are normally shipped by UPS with shipping rges collect. Orders may be placed with: *si Beaver Software Company, P.O. Box 53, shon, WA 98070.*

## CASSETTE VOLUME MONITOR

If you use a standard audio cassette corder to store computer programs, you know can sometimes be difficult to get the volume justed properly when loading programs.

A new electronic device named *VU-LOAD* may



be what you need to eliminate such problems. It enables you to "observe" the signal level at the ear jack of the cassette recorder. You may then adjust the level for the optimum volume setting.

If your cassette does not have an automatic level control, it is claimed that the *VU-LOAD* monitor will enable you to find the optimum volume setting at which your cassette recorder will "save" your programs. And, it will give a positive "save" indication with most other cassette players.

*VU-LOAD* may be mounted anywhere within reach of its 16 inch connector, which plugs into the ear jack of a cassette recorder.

The U.S. price is quoted as $20.95 plus $2.50 for postage/handling. It is claimed to be fully guaranteed and to be available from: *L & G Enterprises, Box 6854, Silver Spring, MD 20906.*

## VARIABLES LISTER

This is a cross reference program for the Radio Shack PC-2 and Sharp PC-1500. It will list all the variables used in a program, showing them in alphabetical order (ascending ASCII). After each variable name is listed, the line(s) where that variable is (are) used will be shown. If a variable is used several times within a line (such as in A=A+1) then the line number is repeated the appropriate number of times.

The program distinguishes between all four possible variations of a variable assignment (i.e., A, A(), A$() and A$).

A *linked list* was used to conserve memory. Three major arrays are used by the program. T$ holds the variables used in the cross referenced program. M stores the starting node in the linked list for each variable name. O stores the linked list.

| | | |
|---|---|---|
| **T$(n)** | **-** | **Name of variable n.** |
| **M(n)** | **-** | **Index in array O of the first entry for the line numbers associated with variable n.** |
| **O(1,M(n))** | **-** | **First line number associated with variable n.** |
| **O(2,M(n))** | **-** | **Index in O of the next reference in O associated with variable n. That is, the next line number associated with variable n will be stored at O(1,O(2,M(n))). O(2,O(M(n))) holds the index in O of the next reference to the variable n. A value of zero in O(2,M(m) indicates the last reference to that variable.** |

Consider the short example program here:

```
 90 : REM TEST
100 : A = A + 1
```

```
110  : B - 12
120  : C - B + 13
130  : D - 14
140  : E - 15
150  : D - 0
160  : END
```

The accompanying diagram illustrates how the T$, M and O$ arrays in the Variables Lister program would appear when processing the above example program.

The arrays T$ and M have been dimensioned to 25 and the linked list to 250. This currently limits

Example *Use of T$, M and O() Arrays.*

| T$ | M | O(1,n) | O(2,n) |
|---|---|---|---|
| A | 1 | 100 | 2 |
| B | 3 | 100 | 0 |
| C | 4 | 110 | 6 |
| D | 5 | 120 | 0 |
| E | 7 | 130 | 8 |
|  |  | 120 | 0 |
|  |  | 140 | 0 |
|  |  | 150 | 0 |

Program *Variables Lister.*

```
1000:REM  "XREF"
2010:CLEAR :T=1:N
     =1:WAIT 10
2020:DIM M(25),O(
     2,250),T$(25
     )*4
2030:DIM N$(0)*36
2040:FOR I=1TO 25
     :M(I)=0:T$(I
     )="":NEXT I
2050:FOR I=1TO 25
     0:O(1,I)=0:O
     (2,I)=0:NEXT
     I
2080:Q=PEEK &7869
     *256+PEEK &7
     86A
2120:Q=Q+5
2140:REM
2160:N$(0)=N$(0)+
     CHR$ (PEEK Q
     )
2170:Q=Q+1
2190:IF PEEK Q<>&
     0DGOTO 2140
2220:LPRINT N$(0)
     :LF 1:Q=Q+1:
     W$=""
2250:REM
2260:IF PEEK Q=&F
     FGOTO 2610
2280:L=PEEK Q*256
     +PEEK (Q+1):
     Q=Q+3:PRINT
     L
2290:IF PEEK Q=&F
     1AND PEEK (Q
     +1)=&ABTHEN
     LET Q=Q+PEEK
     (Q-1)-1
2310:REM
2320:IF PEEK Q=&0
     DGOTO 2560
2350:IF PEEK Q>&E
     5THEN LET Q=
     Q+2:GOTO 254
     0
2380:IF PEEK Q<&4
     1OR PEEK Q>&
     90GOSUB 6000
     :Q=Q+1:GOTO
     2540
2410:REM
2430:W$=W$+CHR$
     PEEK Q:Q=Q+1
2450:IF (&41<PEEK
     QAND PEEK Q<
     &90)OR PEEK
     Q=&24GOTO 24
     10
2480:IF PEEK Q=&2
     8THEN LET W$
     =W$+CHR$
     PEEK Q:Q=Q+1
2510:P=1:GOSUB 30
     00:W$=""
2540:REM
2550:GOTO 2310
2560:REM
2580:Q=Q+1
2600:GOTO 2250
2610:REM
2630:T=T-1:LPRINT
     "WORDS FOUND
     :";T
2640:N=N-1:LPRINT
     "REFERENCES:
     ";N
2650:GOSUB 4000:
     GOSUB 5000
2670:END
3000:REM "STOTRE"
3020:IF T>1GOTO 3
     080
3040:T$(T)=W$:M(T
     )=N:O(1,N)=L
3050:T=T+1:N=N+1
3070:GOTO 3400
3080:REM
3090:REM
3100:IF (P>T)OR (
     W$=T$(P))
     GOTO 3150
3120:P=P+1
3140:GOTO 3090
3150:REM
3170:IF P>TGOTO 3
     340
3200:P=M(P)
3220:REM
3230:IF O(2,P)=0
     GOTO 3280
3250:P=O(2,P)
3270:GOTO 3220
3280:REM
3300:O(2,P)=N:O(1
     ,N)=L:N=N+1
3330:GOTO 3380
3340:REM
3350:T$(T)=W$:M(T
     )=N:O(1,N)=L
3360:T=T+1:N=N+1
3380:REM
3400:REM
3410:RETURN
4000:REM "SORT"
4020:IF T<=1GOTO
     4230
4050:FOR I=1TO T-
     1
4070:K=I:C=I+1
4100:FOR J=CTO T:
     IF T$(K)>T$(
     J)THEN LET K
     =J
4110:NEXT J
4140:IF I=KGOTO 4
     190
4160:U$=T$(I):U=M
     (I)
4170:T$(I)=T$(K):
     M(I)=M(K):T$
     (K)=U$:M(K)=
     U
4190:REM
4200:NEXT I
4230:REM
4240:RETURN
5000:REM "PRTLIN"
5020:FOR I=1TO T
5040:P=M(I):J=1:
     LPRINT T$(I)
5050:REM GOSUB 70
     00
5070:REM
5090:LPRINT USING
     "######";O(1
     ,P);:P=O(2,P
     ):J=J+1
5110:IF P<>0GOTO
     5070
5130:LPRINT
5140:NEXT I
5160:RETURN
6000:REM "SP CHR"
6020:IF PEEK Q<>&
     22GOTO 6110
6040:REM
6060:Q=Q+1
6080:IF PEEK Q<>&
     22GOTO 6040
6110:REM
6130:IF PEEK Q<>&
     26GOTO 6250
6150:REM
6170:Q=Q+1
6190:IF (PEEK Q>=
     &30AND PEEK
     Q<=&39)OR (
     PEEK Q>=&41
     AND PEEK Q<=
     &46)GOTO 615
     0
6220:Q=Q-1
6250:REM
6260:RETURN
STATUS 1          1542
```

**Example *RUN of Variables Lister.***

```
        TEST

      WORDS FOUND: 5
      REFERENCES: 8
      A
         100    100
      B
         110    120
      C
         120
      D
         130    150
      E
         140
```

the program to processing 25 different variables with no more than 250 total references. If these dimensions are not satisfactory, they may be altered. Remember, however, that increasing the size of these arrays will decrease the amount of memory left available for an application program. To change the dimension of T$ and M, change lines 2020 and 2040. Lines 2020 and 2050 are modified to change the dimension of O( ).

The program assumes that the first line of the application program begins with a REM statement. The contents of this REM are printed as a heading on the variables listing. It is a good idea to place the name of the program in this line.

The LCD contains the line number being processed during operation of the program.

Operation is straightforward:

1. Load XREF into memory.
2. MERGE your program.
3. RUN.

Thanks for this valuable utility program go to: *Diane Campbell, 220 Houston, League City, TX 77573.*

---

**Program *ML Portion of Vertical Lister.***

```
7650 B5 2A ED 79
7654 F4 01 8B 02
7658 B5 50 AE 77
765C 4F FD 88 CC
7660 99 B5 07 FD
7664 CA FD 5A 04
7668 AE 77 4E 6A
766C FB CD BA FD
7670 0A 45 28 45
7674 2A FD 88 CD
7678 10 40 FD 0A
767C B5 3A 51 44
7680 45 B7 E0 83
7684 0D B7 0D 8B
7688 07 FD 88 46
768C B5 01 8E 24
7690 FB 9A 28 45
7694 2A FD 88 48
7698 B0 4A 54 45
769C B9 0F 8B 44
76A0 FD CA 45 A6
76A4 89 39 05 26
76A8 89 35 46 46
76AC 46 05 D9 91
76B0 05 45 B9 0F
76B4 2A DF 28 A5
76B8 77 4E F9 A3
76BC 77 4F 16 91
76C0 07 20 16 89
76C4 02 68 00 83
76C8 08 22 1A B5
76CC 20 51 51 51
76D0 51 62 F5 88
76D4 03 A4 8B 03
76D8 B5 20 51 FD
76DC 0A 9E 5F 44
76E0 44 44 9E 49
76E4 84 B3 08 08
76E8 B7 C8 91 53
76EC FD 1A E4 00
```

## VERTICAL LISTER

The SIDELISTER program published in Issue 22 of *PCN* introduced the concept of vertical printing of BASIC program lines. As *Mel Beckman*, author of the article pointed out, his pioneering effort was slow in operation.

This program uses machine language to expand tokens into a string of characters. It will provide a vertical listing at about the same speed as a regular LLIST command.

### Making the Vertical Lister Cassette

First enter the machine codes shown in the accompanying listing using either POKEs or a monitor program. (As with any machine language program, the correct entry of these codes is absolutely crucial. Check your work.) Save the ML program on cassette by executing:

CSAVE M "VERT LISTER (ML)";&7650,&76EE

Next, clear memory with the NEW command. Now enter the BASIC portion of the Vertical Lister as shown in the accompanying listing. Add this to the same cassette using the standard procedure:

CSAVE "VERTICAL LISTER"

### Using the Vertical Lister

Load the BASIC program you desire to have listed into the computer. Make sure that STATUS 0 yields a value of at least 455. That is the amount of space required by the Vertical Lister program, including the space it uses for the dimensioned variable A$( ).

Program *BASIC Portion of Vertical Lister.*

```
      1:"VL"INPUT "SIZ
        E? ";S:GRAPH :
        CSIZE S:ROTATE
        1:A=STATUS 2-
        STATUS 1:DIM A
        $(3*S-1)*76/S+
        4:X=216
      2:CALL &7650,A:C
        =0
      3:IF X=0GLCURSOR
        (0,-576):SORGN
        :X=216
      4:X=X-9*S:
        GLCURSOR (X,0)
        :LPRINT A$(C):
        C=C+1:IF C<3*S
        IF A$(C)GOTO 3
      5:IF PEEK A<255
        GOTO 2
      6:GLCURSOR (0,-5
        76):TEXT
     STATUS 1
                      196
```

Insert the cassette containing the two parts of the Vertical Lister program into the tape player. Execute CLOAD M. After the machine language portion as loaded, execute MERGE to bring in the BASIC part of Vertical Lister. Now execute the command RUN "VL" and enter the desired print size (1 or 2) in response to the prompt. Your BASIC program will then be listed in vertical format. (Of course, the Vertical Lister program itself is not listed.)

## Some Alternatives

The procedure described above makes it easy to produced vertical listings of programs that you have previously saved on tape. The Vertical Lister program itself takes only a small amount of memory and thus takes less than a minute to load. Furthermore, locating the ML portion in the strings variable area means it is unnecessary to have previously reserved memory space for it.

However, when used in the above manner, the VL program can not be used as an aid in debugging a program that is in the process of being developed. This is because the MERGE of the BASIC portion of the VL program prevents editing of the lines of the BASIC program that is under development. Also, string variables E$ to N$ must be left alone, since the area they reside in is occupied by the ML portion of Vertical Lister. (Note that the use of CLEAR would completely erase the ML portion of VL, too!)

One alternative to this situation is to locate the ML portion of the program elsewhere. Indeed, the ML routine has been written so that it is relocatable without any modification to it. (Of course, the CALL &7650 statement in line 2 of the BASIC part would have to be changed to conform to the new address of the ML routine.) If Reserve memory is not being used for anything else, it would be a good place for the ML routine. Another possibility would be to relocate the beginning of the BASIC storage area using the NEW XXXX statement. (The ML code for VL requires 159 bytes.)

If you desire to avoid the use of a MERGE in order to leave the BASIC program in a form whereby it can be edited, then the six lines of BASIC programming used by VL may be loaded prior to development of the program. (As an alternate, the BASIC portion of VL can manually be typed in at any appropriate point.) Naturally, when a MERGE command is not used to load the BASIC portion of VL, then Vertical Lister itself *will* be included in the vertical listing.

A closing note: This program only searches the token tables of the PC-1500 and CE-150. It will not look for tokens in the CE-158 or other devices. Hence BASIC programs that use statements unique to such peripherals can not be listed by VL.

This program developed by: *Norlin Rober, 407 North 1st Avenue, Marshalltown, IA 50158.*

---

## THREE-DIMENSIONAL PROJECTIONS

This program uses the PC-1500/PC-2 with its printer to plot the projection of a three-dimensional object onto a plane. Two kinds of projections are available. One is an *oblique* projection, in which the plane of projection is paralle to the XY plane. In the second type, an *axonometri* projection, the projecting rays are perpendicular t the plane of projection.

In either type of projection, the user selects th direction of the projecting rays by inputting value of L, M and N. These three values represent respectively, the X-, Y- and Z-components of th direction that the projecting lines are to have. Yo may think of (L,M,N) as a vector in the direction o the projecting lines.

Taking the points in the order in which they ar to be connected, the X-, Y- and Z-coordinates o each point must be specified in DATA statements The letter S should appear in the DATA list t indicate the *start* of each sequence of points to b connected. (Its effect should become clear whe the accompanying example is studied.) Use th letter T as the last item in the DATA list to signa *termination* of the drawing.

## RS-232C INTERFACE PRINT FORMATTER

A limitation of the Radio Shack RS-232C Interface is the inability to change the 18 column, CSIZE 2 output format while in the Terminal mode. This can be particularly frustrating when dealing with numeric data formatted for 40 or 80 column printers because of the wrap-around feature. However, the data stored in RAM can eventually be accessed using BASIC. From there, it can be output to a printer in any CSIZE or format desired.

This program, provided by *Robert Stock, 304 Horseshoe Lane, Downingtown, PA 19335,* serves to simulate a print spooler. (A "print spooler" is a portion of a program that handles data as it is being prepared for output. It places the data into a buffer. On large computer systems the buffer is generally in disk memory.) The program permits scrolling through the data, forwards or backwards, by full and half screens or by single characters. The user may print at CSIZE 1 in the normal fashion. Alternately, "sideways" printing may be invoked at a 42 character width for CSIZE 2 or 84 character width using CSIZE 1. (The sideways printing routine is derived from Mel Beckman's *Sidelister* program that appeared in Issue 22 of *PCN*.) This program also enables the user to save and load data using cassette tape.

Program *RS-232C Print Formatter.*

```
 10:" "TEXT :CSIZE
    1:INPUT ">>>?
    ";L$:GOTO L$
 15:END
 20:"SET"CLS :K=
    STATUS 2:GOTO
    10
 25:"GO"CLS :WAIT
    0:PRINT "  ==>
     <<1 <13 <26 P
    RT SIDE"
 30:F=ASC INKEY$ -
    16:IF F<1OR F>
    6THEN 30
 35:CLS :GOTO 100*
    F
100:N=ASC INKEY$ :
    P=PEEK K
110:IF N=17THEN
    PRINT CHR$ P;:
    K=K+1
120:IF N=13THEN 25
130:GOTO 100
200:K=K-1:GOSUB 45
    0:GOTO 25
300:K=K-13:GOSUB 4
    50:GOTO 25
400:K=K-26:GOSUB 4
    50:GOTO 25
450:S=K:FOR J=1TO
    25
460:Q=PEEK S:PRINT
    CHR$ Q;:S=S+1
470:NEXT J:WAIT :
    PRINT CHR$
    PEEK (S+1):
    RETURN
500:P=PEEK K:
    LPRINT CHR$ P;
510:IF P=13THEN LF
    1:TAB 0
520:K=K+1:GOTO 500
600:INPUT "CSIZE =
     ? ";Z:CLS :X=
    212-Z*6
610:GRAPH :
    GLCURSOR (X,0)
    :SORGN :ROTATE
    1:CSIZE Z
620:X=0:L=0:C=0:M=
    0:GOTO 680
630:IF C>MLET M=C:
    IF M>(84/Z)LET
    M=84/Z
640:C=0:X=X-Z*10:
    GLCURSOR (X,0)
650:L=L+1:IF L=INT
    (20/Z)GOSUB 67
    0
660:RETURN
670:L=0:X=0:Y=-(Z*
    M*6.4):
    GLCURSOR (X,Y)
    :SORGN :M=0:
    RETURN
680:IF C>(84/Z)
    GOSUB 630
690:P=PEEK K:
    LPRINT CHR$ P;
700:IF P=13GOSUB 6
    30
710:C=C+1:K=K+1:
    GOTO 680
720:"SAVE"CLS :
    INPUT "Filenam
    e? ";F$:CSAVE
    MF$;STATUS 2,
    STATUS 3:GOTO
    10
730:"LOAD"CLS :
    INPUT "Filenam
    e? ";F$:CLOAD
    MF$;STATUS 2:
    GOTO 10
STATUS 1
                766
```

### Operating Instructions

Run the program by pressing DEF/SPACE. At the >>>? prompt, input SET and press ENTER to zero the spooler and initialize the printer. Alternately, type GO and press ENTER to see the Command Menu:

**==>  <<1  <<13  <<26  PRT  SIDE**

Select from the menu as follows:

Press F1 (==>) to scroll forward through the data in the buffer. Hold the F1 key down for continuous scrolling. Press ENTER to exit to the Command Menu.

Press F2 (<<1) to move backwards by one character. Press ENTER to return to the Command Menu.

Press F3 (<13) to move backwards by a half screen (13 characters). Press F4 (<26) to move back by a full screen. Either function may be terminated by pressing ENTER.

Press F5 (PRT) to print normally at CSIZE 1. This gives a 36 column format which is great for networks such as Compuserve, The Source or Dow Jones.

Press F6 (SIDE) to print sideways. Respond with a 1 or 2 to the prompt for CSIZE. A 1 will start the printer in an 84 column by 20 lines per page format. A 2 will print in a 42 column by 10 lines per page arrangement.

Once in the print mode (initiated by either F5 or F6), press BREAK to stop printing. DEF/SPACE may then be used to re-initialize the printer or re-run the program.

If you want to save data on cassette tape for future use, set up the recorder as if to CSAVE in the usual manner. At the >>>? prompt type SAVE and press ENTER. Respond to the prompt with an appropriate file name. Press ENTER at the end of the name to send date to the tape recorder. If the data buffer is not completely filled, listen to the audio sound made by the PC as the data is transmitted. When the sound becomes a steady tone, no more data is being sent. Press BREAK and then the CLear key to end the SAVE operation.

(When RAM is full, the tape SAVE operation will end automatically.)

To recover previously saved data, set up the recorder as if to CLOAD in the usual manner. At the >>>? prompt, type LOAD and press ENTER. Input the desired filename when prompted and conclude the name with the ENTER key. The data will be fed to the PC. If the file was made from a full RAM, then the loading operation will conclude automatically. Otherwise, a steady tone will indicate the end of data. Press BREAK and CLear to end the load operation when this tone is heard.

### A Tip

If you use an acoustic modem, such as the Radio Shack Model AC-3, you can tap into the data going over the telephone lines. This can be accomplished through the use of an inexpensive in-line telephone pickup. Record the data on tape. The recorded data can then be played back through an earphone that has been mounted in foam rubber and inserted into the modem microphone cup. With the modem in the originate mode and the tape recorder volume set properly, the data can then be fed into the RS-232C interface. With the PC in Terminal mode, you can receive this data as though it was an actual telephone communication. Practically unlimited amounts of data can be stored on tape and played back as desired for viewing and/or printing!

## FRACTIONAL BASE CONVERSION

Here is a program that will convert between binary, octal, decimal and hexadecimal. It does it without resorting to the use of double-digits (such as 10 and 15 in place of A and F). It has fractional

Program *Fractional Base Conversion*

```
  1:REM S.ETHERIDGE 6-83
 10:PAUSE "  BASE CONVER
    TER W/F": CLEAR :
    DIM W$(1)*24,F$(1)*2
    4,T$(24)*2,D$(12)*2
 20:INPUT "  BASE: ";B:
    INPUT "  CONVERT TO:
     ";C: INPUT "  NUMBE
    R: ";W$(1):L= LEN (W
    $(1))
 30:FOR S=1 TO L:T$(S)=
    MID$ (W$(1),S,1): IF
    T$(S)="." LET R=S:S=
    L
 40:NEXT S
 50:IF R<>0 LET F$(1)=
    RIGHT$ (W$(1),L-R):W
    $(1)= LEFT$ (W$(1),R
    -1)
 60:L= LEN (W$(1)):E=L:P
    =0:X=0:Y=0: GOTO 80
 70:L= LEN (F$(1)):E=0:P
    =1:X=0:Y=1
 80:FOR S=1 TO L:H=1
 90:IF P=0 LET T$(S)=
    MID$ (W$(1),S,1):E=E
    -1
100:IF P=1 LET T$(S)=
    MID$ (F$(1),S,1),E=S
    *(-1)
110:IF B<>8 THEN 130
120:IF T$(S)="8" OR T$(S
    )="9" LET S=L: GOTO
    10
130:IF B=16 GOSUB 330
140:X=X+ VAL (T$(S))*B^E
    : NEXT S
150:IF P=0 LET W$(1)=
    STR$ (X)
160:IF P=1 LET F$(1)=
    STR$ (X)
170:IF Y=1 THEN 200
180:GOTO 70
190:IF R<>0 THEN 140
200:W= VAL (W$(1)):F=
    VAL (F$(1)):A=W+F:
    IF C=10 LET W$(1)=
    STR$ (A): GOTO 320
210:L= LEN (W$(1)):W$(1)
    ="":X=0:Y=0: IF L=0
    THEN 250
220:X= INT (W/C):Y=(((W/
    C)-X)*C):W=X:H=2:T$(
    S)= STR$ (Y): IF C=1
    6 GOSUB 330
230:W$(1)=T$(S)+W$(1):
    IF W=0 THEN 250
240:GOTO 220
250:IF VAL (F$(1))=0
    THEN 320
260:W$(1)=W$(1)+".":M=
    LEN (F$(1)):P=M:X=0:
    Y=0
270:P=P-1
280:X= INT (F*C):Y=(F*C)
    -X:F=Y:H=2:T$(S)=
    STR$ (X): IF C=16
    GOSUB 330
290:W$(1)=W$(1)+T$(S):
    IF Y=0 THEN 320
300:IF P=1 AND X=0 THEN
    280
310:IF P<>-1 THEN 270
320:PAUSE "ANSWER BASE "
    ;C: PRINT W$(1):
    GOTO 10
330:IF D$(1)<>"" THEN 36
    0
340:IF H=1 LET I=1,J=12,
    K=1
350:IF H=2 LET I=12,J=1,
    K=-1
360:FOR D=I TO J STEP K:
    READ D$(D): NEXT D:H
    =0: RESTORE
370:IF T$(S)=D$(1) LET T
    $(S)=D$(2): RETURN
380:IF T$(S)=D$(3) LET T
    $(S)=D$(4): RETURN
390:IF T$(S)=D$(5) LET T
    $(S)=D$(6): RETURN
400:IF T$(S)=D$(7) LET T
    $(S)=D$(8): RETURN
410:IF T$(S)=D$(9) LET T
    $(S)=D$(10): RETURN
420:IF T$(S)=D$(11) LET
    T$(S)=D$(12): RETURN
430:RETURN
440:DATA "A","10","B","1
    1","C","12","D","13"
    ,"E","14","F","15"
```

capabilities. And, it does not require that you enter each digit separately.

*Steven L. Etheridge, 14451 Tyler Street, Sylmar, CA 91342,* submitted this program and adds the following general comments.

The program is designed to run on a Sharp 1250 or 1500. (It should do nicely on a Radio Shack PC-2 or PC-3.) To use the program, just load it and type the command RUN. Respond to the queries for the base (of the number you will be inputting), the "convert to" base and then the actual number that you wish to have converted. The number may be up to 24 characters and may be integer, fractional or a combination. (The limit of 24 characters includes the radix point!) That is all there is to it!

The program is broadly organized as follows:

| | |
|---|---|
| Lines 10 & 20 | Initialization and data input. |
| Lines 30 & 40 | Scan for fractional number. |
| Lines 60 - 200 | Convert input to decimal. |
| Lines 210 - 310 | Convert decimal to final base. |
| Line 320 | Display answer. |
| Lines 330 - 440 | Hexadecimal translation. |

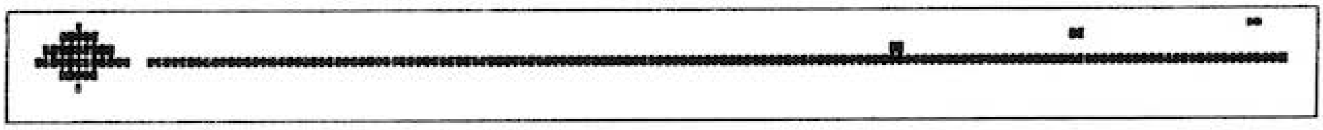
The program only has a few error traps, due to the limited memory in a PC-1250. Those with a PC-1500 may want to add to the trapping that was included to catch the use of 8 and 9 while working in octal.

## HEXADECIMAL LOADER

*William Delinger, Northern Arizona University, Department of Physics, Box 6010, Flagstaff, AZ 86011,* submitted this program. It may be used to load information that is in hexadecimal notation directly into memory. While designed primarily for the PC-1500 and PC-2, changing the line numbers to a lower range should also make it useful to PC-1250 and PC-3 users who are dabbling with machine language.

Program *Hexadecimal Loader.*

```
1000 REM  HEXADECIMAL LOAD PROGRAM
1010 INPUT "STARTING ADDRESS=?";S
1020 INPUT "HEX=?";H$
1030 IF H4="STOP"THEN STOP
1040 L=LEN(H$)
1050 IF L<>2THEN 1130
1060 D1=ASC(LEFT$(H$,1))
1070 D2=ASC(RIGHT$(H$,1))
1080 IF D1>64AND D1<71LET D1=(D1-55)*16:
        GOTO 1110
1090 IF D1>47AND D1<58LET D1=(D1-48)*16:
        GOTO 1110
1100 GOTO 1130
1110 IF D2>64AND D2<71LET D2=(D2-55)+D1:
        GOTO 1150
1120 IF D2>47AND D2<58LET D2=(D2-48)+D1:
        GOTO 1150
1130 PAUSE "WHAT?"
1140 GOTO 1020
1150 POKE S,D2
1160 S=S+1
1170 GOTO 1020
1180 END
```



## METEORS GAME FOR PC-2/PC-1500

*PCN* rarely publishes games. However, we are making an exception for a submission by *David Hergert, 4714 Chickering Avenue, Cincinnati, OH 45232.* His combination BASIC and machine language (hybrid) program illustrates how ML routines can be called upon to enhance critical portions of a program. The program is elegant in its playing simplicity, yet can be rather challenging -- just try getting a perfect score (10 hits) when the game is set for its highest skill level (1).

To play the game, imagine that you are Captain of the spaceship that appears on the display. As you are traveling along you suddenly encounter a meteor shower. There are 15 meteors in the shower but you only have 10 laser-missiles with which to defend yourself. (You have learn to judge which meteors will probably miss your craft!)

Once the program has been loaded, use a RUN command to start the action. Respond to the skill level query according to courage. Level 4 is for beginners. Level 1 brings on the meteors at their fastest rate. Watch out! Any meteor landing a solid hit on your spaceship ends the game. A successful mission on your part, brings a visual reward.

If you want to do some tinkering with the program, variable G in line 40 controls the display time of the laser. The laser firing is displayed via a ML routine that uses an exclusive OR instruction to invert the center row of dots. Line 190 of the BASIC program calls a ML counter that displays the contents of the accumulator for a short amount of time.

Good luck, Captains!

### Program *Meteors*

```
 10:POKE &47C0,&48
    ,&76,&4A,&0,&5
    ,&BD,&88,&41,&
    4E,&4E,&99,&8,
    &4C,&77,&8B,&6
 20:POKE &47D0,&48
    ,&77,&4A,&0,&9
    E,&12,&9A,&FD,
    &6A,&FD,&A8,&C
    D,&10,&80,&FD,
    &2A
 25:POKE &47E0,&62
    ,&6E,&01,&81,&
    07,&FD,&A8,&CD
    ,&10,&80,&9E,&
    E,&9A
 27:WAIT 50:PRINT
    "PRESS + TO FI
    RE"
 30:INPUT "ENTER S
    KILL LEVEL(1TO
    4) ";M:CLS
 40:S=0:W=0:P=0:G=
    50
 50:A=1:Z=((RND 6-
    1)/2+1)*10
 60:WAIT M:P=P+1
 70:FOR C=155TO 6
    STEP -Z
 80:GCURSOR 0:
    GPRINT "08080C
    0C1E1E3F1E1E0C
    0C0808"
 90:IF P=16THEN 27
    0
100:GCURSOR ABS (C
    -1):GPRINT A;A
110:IF INKEY$ ="+"
    AND S<10THEN 1
    80
120:GCURSOR ABS (C
    -1):GPRINT 0;0
130:A=A*2:IF A=128
    THEN 50
140:NEXT C
150:WAIT 50:
    GCURSOR 3:
    GPRINT "412241
    08412241"
160:CLS
170:GOTO 250
180:CALL &47C0
190:CALL &47D7,G
200:CLS :S=S+1:IF
    A=8THEN 220
210:GOTO 80
220:GCURSOR 0:
    GPRINT "08080C
    0C1E1E3F1E1E0C
    0C0808"
230:WAIT 50:
    GCURSOR C:
    GPRINT "412214
    08142241"
240:CLS :W=W+1:
    GOTO 50
250:WAIT 100:PRINT
    "YOU HIT ";W;"
     METEORS."
260:CLS :GOTO 30
270:CLS :WAIT 0
280:FOR X=1TO 155
290:GCURSOR X:
    GPRINT "000008
    080C0C1E1E3F1E
    1E0C0C0808"
300:NEXT X
310:WAIT 50:PRINT
    "YOU MADE IT!"
    :GOTO 30
STATUS 1          829
```

## FROM THE HIP POCKET

Our readers write to share information with others.

### Time

The Sharp PC-1500/Radio Shack PC-2 TIME function is a good calendar clock. But it may take a moment or two to interpret a display like 70403.0017 at 3 AM or when there are distractions. One remedy is to use the Instruction Manual's clock simulation program. But it is a nuisance to type this program back in each time after one has deleted a long program using NEW.

A better remedy is also an interesting example of the sort of small program that can be executed via Reserve keys. Program any two adjacent Reserve keys as follows:

```
Left:  Z=(INT(TIME*100+.7)-INT(TIME/100)*1E4)/
       100●
Right: PRINTUSING"###.##";"Date -";INT(TIME/
       100)/100;" Time -";Z-12*INT(Z/13)●
```

(The use of .7 instead of .5 in the first INT expression is not a misprint. With the one exception that is mentioned below, it rounds base-60 minutes and seconds properly for a base-10 display.)

Pressing LEFT alone gives the time, but a time such as 2:10 PM will appear as 14.1. Pressing LEFT then RIGHT gives a properly formatted date and modulus-12 time. For instance, at 2:10 PM on July 4th the display would read:

```
Date -  7.04  Time -  2.10
```

There is one minor bug. Whenever the time is 30 or more seconds past 59 minutes after an hour, screen output will display that hour and 60 minutes instead of the next hour (3.60, for instance, instead of 4.00). Unfortunately, the programming also uses up most of the reserve area memory, so that eliminating this bug requires simplifying the display. Use three Reserve keys:

```
Left:  (As above)
Next:  Z=Z+.4*(Z-INTZ=.6)●
Right: PRINTUSING"###.##";"D";INT(TIME/100)/
       100;" T";Z-12*INT(Z/13)●
```

I prefer the first program.

*Arthur L. Thomas*
*University of Kansas*
*307 Summerfield Hall*
*Lawrence, KS 66045*

### I/O Ports

I have found the handshake signals of the CE-158 serial interface make convenient one-bit input and output ports for the Sharp PC-1500.

The example provided here uses the Request-To-Send (RTS) signal from pin 4 on the RS-232 connector. This pin is connected to a diode and a 2.2K resistor and then to a solid state relay as shown in the accompanying diagram. In this way the low voltage output of the PC-1500 can be used to control a 110 V.A.C. device. The BASIC command OUTSTAT 0 will turn the relay on and the command OUTSTAT 2 will turn it off. Or, equivalently, these values may be POKEd at hexadecimal location D00E in the alternate memory of the PC. Machine language commands may also be used to control the relay.

As indicated in the diagram, the output from pin 4 on the RS-232 connector is used to control the solid state relay. A suggested relay is the Crydom Model D2402 (priced at about $16.00 at Newark Electronics). This relay will control A.C. currents up to 2.5 amperes. With the arrangement shown, about 1.6 milliamperes is drawn from the CE-158 interface when the potential difference at the D.C. input to the relay is about 3.6 volts. This current is well below the limiting value of the SN75188 driver chip in the interface, which is approximately 10 milliamperes.

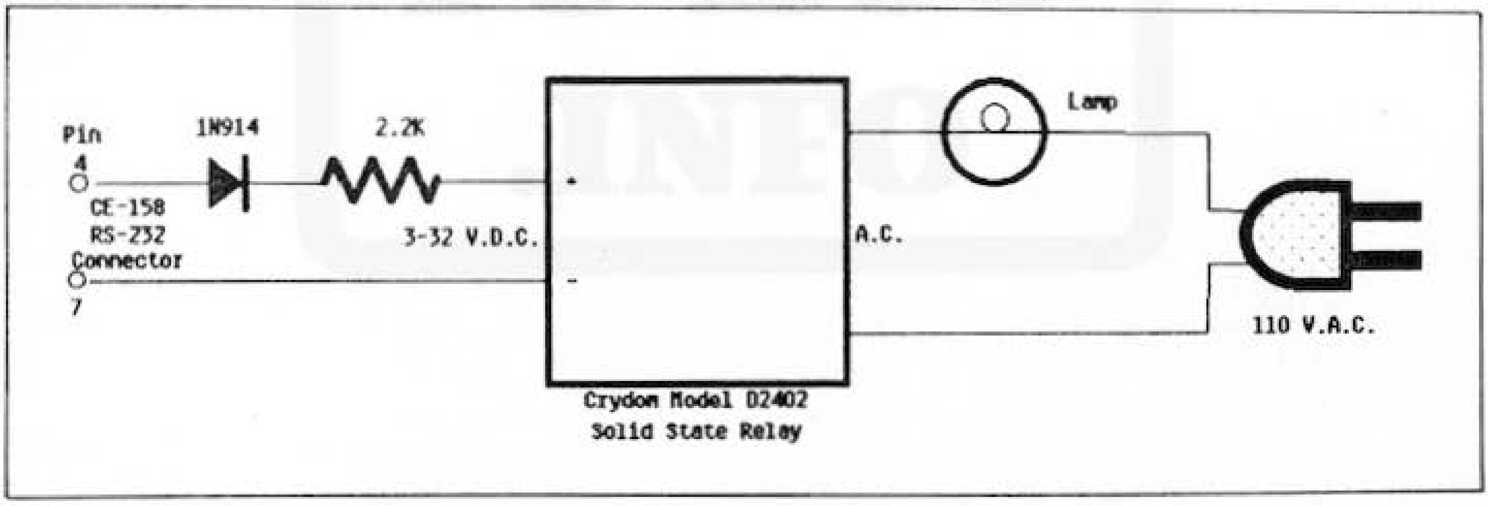
The BASIC program is a simple example illustrating that the latched port can be used to turn on a 110 V.A.C. light after an elapsed time. The first three lines of the program prompt the user for the hour, minute and second. The program then goes into a loop and will turn on the light at the requested time. Of course, a more sophisticated type of control program could be written, but this illustrates the method.

```
 10 : INPUT "HOUR=?";H
 20 : INPUT "MINUTE=?";M
 30 : INPUT "SECOND=?";S
 40 : CLS
 50 : C=10000*H+100*M+S
 60 : A=TIME/100
 70 : B=(A-INT(A))*1000000)
 80 : IF B=C THEN OUTSTAT 0:STOP
 90 : GOTO 60
100 : END
```

I have also used this general scheme to run a stepper motor. Similarly, other signals can be input or output to connect the PC with the outside world.

*William G. Delinger*
*Northern Arizona University*
*Department of Physics*
*Box 6010*
*Flagstaff, AZ 86011*

Diagram *Solid State Relay, Controlled by RS-232C Port, Activates Lamp.*



## ALTERNATE CONTROL OF THE PC-1500 LCD

The liquid-crystal display (LCD) on the Sharp PC-1500 and Radio Shack PC-2 consists of 156 columns by 7 rows for a total of 1092 points or pixels. If you have studied the Instruction Manual, then you know that one way to access individual pixels is to use the GCURSOR and GPRINT statements. The GCURSOR directive will select the starting "column" while GPRINTprovides for the turning on (or off) of selected dots within a column.

Selecting a particular column is not difficult because the GCURSOR parameter allows this to be directly specified: GCURSOR 10 directs the PC to start displaying in column 10 (with the first column being referred to as column zero).

Selecting a particular row within a column can be somewhat more complicated. Rows are numbered from top to bottom, starting with row zero. Rows are assigned values according to a binary positioning scheme. The top row (row 0) is activated with a value of 1. The next row (row 1)