POCKET COMPUTER NEWSLETTER

WWW. PC-1500 .INFO

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

PART III

Issues 34 to 40

IMPROVED VERTICAL LISTER PROGRAM

Here is an improved version of the Vertical Lister previously described in Issue 27 of *PCN*. This one is written entirely in machine language.

It can be left residing in memory and used at any time without the need for MERGEing a BASIC portion. It will print token words used by programs utilizing the CE-158 (provided that the CE-158 is connected).

Listings may be made in either CSIZE 1 or CSIZE 2. In size 2, each "page" printed will contain 12 lines of up to 42 characters-per-line. In size 1, 24 lines having up to 84 characters each may be printed. Long lines will wrap-around to the following line.

If memory contains MERGEd programs, they will all be listed.

Entering the Program

The listing supplied with this article places the Vertical Lister program in the Reserve memory area of a computer having the CE-155 8K memory module installed. However, this program is completely relocatable. You can put it wherever you want it. If you have a PC-1500A, it is suggested that you put it into memory starting at the address &7D00. If you have a 16K RAM module and do not want to lose the Reserve area, you may wish to reallocate the computer's memory by executing NEW &01A0 and then putting the program into the area beginning at &0D00. If you do not have a Monitor program, you will have to enter the program using POKE statements.

When the program has been loaded into memory, it may be saved on tape by execution of the command string CSAVE M "VERTICAL LISTER";&3800,&38C4. (Naturally, if you locate your copy of the program elsewhere, you will have to change the starting and ending addresses used in your CSAVE M statement accordingly.) The program may be reloaded from tape using the CLOAD M command into the same addresses from which it was originally CSAVEd. Alternately, load it into some other location by specifying the starting address in the CLOAD M command.

Using the New Vertical Lister Program

To produce a vertical listing, simply execute CALL &3800 (or whatever address you have used as the beginning address of the machine language program). If you interrupt the vertical listing process with BREAK, execute TEXT following the break in order to reset the printer.

The program is not totally idiot-proof. If you execute it when there is no BASIC program in memory, the result may be having some garbage printed!

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

Program *Improved Vertical Lister.*

```
3864 B7 3A ED 79   3800 E9 78 50 00
3868 F4 01 8B 02   3804 EB 79 F0 FF
386C B7 63 83 24   3808 BE 02 6F BE
3870 44 F5 88 03   380C AC 97 DD AE
3874 FD 0A 84 5E   3810 79 F2 B5 DB
3878 3A ED 79 F4   3814 FB B1 09 ED
387C 01 8B 02 5E   3818 79 F4 01 89
3880 63 83 14 9E   381C 02 B1 09 FD
3884 36 4A 10 14   3820 C8 48 7B 4A
3888 00 2A BE B4   3824 7F 43 B5 00
388C F4 BE D2 B3   3828 43 43 0E BE
3890 83 10 8E 12   382C AC 07 BE A7
3894 FD 0A 46 46   3830 69 5A 10 CC
3898 CA 50 4A 10   3834 50 6B 08 E9
389C 14 00 2A BE   3838 78 50 00 51
38A0 B4 F4 FD 8A   383C 51 BE 10 48
38A4 99 92 FD C8   3840 7B 4A B0 F5
38A8 48 7B 4A 7F   3844 05 B7 3A 83
38AC B5 00 43 43   3848 04 B7 30 33
38B0 B5 C0 43 B5   384C 0A B5 3A 51
38B4 FD 0E BE AC   3850 45 B7 0D 8B
38B8 07 BE A7 69   3854 30 B7 E0 81
38BC FD 8A 9B 85   3858 1E 28 45 2A
38C0 BE AC BB CD   385C FD 88 CD 1C
38C4 42            3860 04 DF 2A 12
```

LINEAR SYSTEMS

This program for the PC-1500/PC-2 will find the solutions of a system of linear equations. The size of the system that it can handle depends on the amount of memory present. When used with the PC-1500A and a CE-161 RAM module, it can solve a system of 50 equations. Using the PC-1500 and an 8K RAM module, a system may contain up to 32 equations.

An Example

This example illustrates how the program may be used. Assume we wish to solve the system of equations:

$$\begin{aligned} 3W + 2X + 6Y + 2Z &= 2 \\ W + 4X + 3Y + 2Z &= 6 \\ 4W + 2X + 5Y + Z &= 1 \\ W + 5X + 3Y + 3Z &= 3 \end{aligned}$$

Execute RUN after loading the program. Respond to the prompt "NO OF EQ?" with 4. The display will show prompts for inputting the coefficients, using A1, A2, etc., with the constant term designated C. Thus, in this example, when asked for "EQ 1: A1?", the first coefficient in the first equation should be inputted, which is 3. Similarly, respond to "EQ 1: A2?" with 2, to the next prompt with 6 and then with 2. When the prompt "EQ 1: C?" appears, the constant term on the right side of the equation (which is 2 here) should be inputted.

After inputting the values, the prompt "EDIT?" will be displayed. If you do not wish to edit or review the inputs you have entered, key ENTER. If editing is desired, input "Y" followed by ENTER. If the latter option is taken, the equation number that you wish to review will be requested. After this information has been given, each coefficient in that equation will be presented for review, followed by a question mark. If you wish to alter the coefficient, enter the new value. Otherwise, just press ENTER.

After editing, the prompt "PRINT?" will be displayed. Respond with a "Y" to print a list of the coefficients or key ENTER to skip printing.

The correct solutions for the equations used in this example are W=-2, X=3, Y=2 and Z=-5. The display should show these as X1, X2, X3 and X4. Due to limitations of floating-point arithmetic, the first solution will be given as -1.999999999 rather than as -2 and the fourth as -4.999999999 rather than -5.

The query "PRINT?" will be shown again, offering the option of printing the calculated results.

Execution Times

A system of five equations will take about 6 seconds. Ten equations: 42 seconds. Twenty equations: 5 minutes and 20 seconds. Thirty equations: just under 18 minutes.

The message "NO SOL" will be displayed when there is no unique solution to the system. This occurs when the system is either inconsistent or dependent.

Norlin Rober, 407 North 1st Avenue, Marshalltown, IA, 50158, submitted this routine.

Program *Linear Systems*.

```
10: INPUT "NO OF E
   Q? ";N:DIM A(N
   ,N+1):GOTO 12
11:GOTO 10
12:WAIT 0:FOR I=1
   TO N:FOR J=1TO
   N
13:CLS :PRINT "EQ
   ";I;": A";STR$
   J;:INPUT "? ";
   A(I,J):NEXT J:
   GOTO 15
14:GOTO 13
15:CLS :PRINT "EQ
   ";I;": C";:
   INPUT "? ";A(I
   ,N+1):NEXT I:
   GOTO 24
16:GOTO 15
20: INPUT "EQ NO?
   ";I:IF I>0AND
   I<=NTHEN 22
21:GOTO 20
22:FOR J=1TO N:
   CLS :PRINT "EQ
   ";I;": A";STR$
   J;"=";A(I,J);:
   INPUT "? ";A(I
   ,J)
23:NEXT J:CLS :
   PRINT "EQ";I;"
   : C=";A(I,N+1)
   ;:INPUT "? ";A
   (I,N+1)
24:CLS :INPUT "ED
   IT? ";A$:GOTO
   20
30:WAIT :INPUT "P
   RINT? ";A$:
   GOTO 32
31:GOTO 40
32:FOR I=1TO N:
   LPRINT "EQUATI
   ON";I:FOR J=1
   TO N:LPRINT
   STR$ J+" ";A(I
   ,J):NEXT J:
   LPRINT "C ";A(
   I,J):LPRINT :
   NEXT I
40:CLS :FOR I=1TO
   N-1:IF A(I,I)<
   >0THEN 50
41:FOR J=I+1TO N:
   IF A(J,I)=0
   NEXT J:PRINT "
   NO SOL":END
42:FOR K=1TO N+1:
   M=A(J,K),A(J,K
   )=A(I,K),A(I,K
   )=M:NEXT K
50:FOR J=I+1TO N:
   FOR K=I+1TO N+
   1:A(J,K)=A(J,K
   )-A(I,K)*A(J,I
   )/A(I,I):NEXT
   K:NEXT J:NEXT
   I
51: IF A(N,N)=0
   PRINT "NO SOL"
   :END
52:M=N+1:FOR I=N
   TO 2STEP -1:
   FOR J=1TO I-1:
   A(J,M)=A(J,M)-
   A(J,I)*A(I,M)/
   A(I,I):NEXT J:
   NEXT I
60:FOR I=1TO N:
   PRINT "X"+STR$
   I+"=";A(I,M)/A
   (I,I):NEXT I
70: INPUT "PRINT?
   ";A$:LPRINT "S
   OLUTION:":FOR
   I=1TO N:LPRINT
   STR$ I+" ";A(I
   ,M)/A(I,I):
   NEXT I:LPRINT
```

STATUS 1 = 890

ISOMETRIC DRAWINGS

Richard H. Chrystie, 14824 E. Walbrook, Hacienda Heights, CA 91745, provides this program that is designed to make isometric drawings. The number of components per drawing is dependent on the amount of memory in your PC as shown in the accompanying table.

Components are rectangular or cylindrical solids. The axis of rotation for the cylinders is either the X, Y or Z axis. A scale factor is inputted by the user. If a scale of 1 is selected, all dimensions are in inches. The color of each item may be specified. After an item is drawn, the program asks if you want another item, thus you may make drawings of single or multiple items.

The program is written in BASIC so it is slow. Drawings are limited in size to the distance that the printer can back up.

After an initial view has been drawn you can alter the scale or viewing angle. A rotation of 30 degrees and a tilt of 10 degrees is nice. A rotation of 0 degrees and a tilt of 0 degrees produces a side view. Rotating to 90 degrees results in an end view.

Program ***Isometric Drawings.***

```
  5  "3VIEW2"
 11  CLEAR
 15  TEXT :COLOR 0
 20  C=-1
 50  DIM X(15,26),Y
     (15,26),Z(15,2
     6),C(15)
 70  DIM R(26),T(26
     ),B$(15),G(15)
 72  GOSUB 300
 75  INPUT "SCALE?"
     ,S:S=126.5823*
     S
 80  CSIZE 1
 81  LPRINT "SCALE=
     ",S/126.58:LF
     1
 82  LPRINT "      NO
           UP     RT
           BK"
 83  LF 1
100  INPUT "WANT AN
     OTHER ITEM?",A
     $
101  IF A$="N"THEN
     430
105  INPUT "ITEM NU
     MBER?",N
106  INPUT "COLOR?"
     ,C(N)
110  INPUT "BOX or
     CYLINDER (B/C)
     ?",B$(N)
111  IF B$(N)="C"
     THEN 700
115  INPUT "DIST UP
     ?",W
116  INPUT "DIST RI
     GHT?",U
117  INPUT "DIST BA
     CK?",V:V=-V
120  INPUT "BOX HEI
     GHT?",Z
125  INPUT "BOX WID
     TH?",X
130  INPUT "BOX DEP
     TH?",Y:Y=-Y
133  GOSUB "E"
134  G(N)=16
135  X(N,1)=U:Y(N,1
     )=V:Z(N,1)=W
140  X(N,2)=U+X:Y(N
     ,2)=V:Z(N,2)=W
145  X(N,3)=U+X:Y(N
     ,3)=V+Y:Z(N,3)
     =W
150  X(N,4)=U:Y(N,4
     )=V+Y:Z(N,4)=W
155  X(N,5)=U:Y(N,5
     )=V:Z(N,5)=W
160  X(N,6)=U:Y(N,6
     )=V:Z(N,6)=W+Z
165  X(N,7)=U+X:Y(N
     ,7)=V:Z(N,7)=W
     +Z
170  X(N,8)=U:Y(N,8
     )=V:Z(N,8)=W+Z
175  X(N,9)=U:Y(N,9
     )=V+Y:Z(N,9)=W
     +Z
180  X(N,10)=U:Y(N,
     10)=V+Y:Z(N,10
     )=W
185  X(N,11)=U:Y(N,
     11)=V+Y:Z(N,11
     )=W+Z
190  X(N,12)=U+X:Y(
     N,12)=V+Y:Z(N,
     12)=W+Z
195  X(N,13)=U+X:Y(
     N,13)=V+Y:Z(N,
     13)=W
200  X(N,14)=U+X:Y(
     N,14)=V+Y:Z(N,
     14)=W+Z
205  X(N,15)=U+X:Y(
     N,15)=V:Z(N,15
     )=W+Z
210  X(N,16)=U+X:Y(
     N,16)=V:Z(N,16
     )=W
220  GOTO 100
250  INPUT "TAPE RE
     ADY?",B$
252  IF B$="Y" THEN
     256
254  GOTO 250
256  RETURN
300  INPUT "LOAD FR
     OM TAPE?",B$
305  IF B$="Y"THEN
     315
310  RETURN
315  INPUT "WHAT TI
     TLE?",I$
320  GOSUB 250
330  INPUT #I$;X(*)
     ,Y(*),Z(*),N,S
     ,C(*),G(*),B$(
     *)
335  RETURN
350  INPUT "SAVE ON
     TAPE",B$
355  IF B$="Y"THEN
     365
360  RETURN
365  INPUT "WHAT TI
     TLE?",I$
370  GOSUB 250
375  PRINT #I$;X(*)
     ,Y(*),Z(*),N,S
     ,C(*),G(*),B$(
     *)
380  RETURN
430  GRAPH
450  GLCURSOR (30,-
     250):SORGN
455  T=25:F=35.26
460  FOR M=1TO N
465  GOSUB "DRAW"
470  NEXT M
475  GLCURSOR (0, -3
     00)
480  INPUT "NEED AN
     OTHER STRIP?",
     D$
485  IF D$="N"THEN
     491
486  GLCURSOR (-215
     ,-500):SORGN
488  K=K+215
490  GOTO 460
491  INPUT "WANT TO
     REVISE DATA?"
     ,A$
492  IF A$="N"THEN
     500
493  GOTO 100
500  INPUT "WANT AN
     OTHER VIEW?",A
     $
510  IF A$="N"THEN
```

Tilting to 90 digrees gives a top view. Large drawings are produced by pasting together strips side-by-side.

Drawings may also be stored on tape for long term storage.

Caution: Be sure to change the DIMension statements in lines 50 and 70 in accordance with the accompanying table! Failure to properly size the arrays to match the memory available in your system can result in improper operation and loss of data.

Table *Memory Requirements.*

Items	RAM Req'd	Equipment	Data Storage
5	7.3K	8K Module	About 90 in.
10	10.7K		
15	14.1K	16K Module	About 270 in.
20	17.5K	16K Module	About 360 in.
25	20.1K	26K System	

```
    990
511 INPUT "WANT AN
    OTHER SCALE?",
    C$
512 IF C$="N"THEN
    520
514 INPUT "SCALE?"
    ,S:S=126.5823*
    S
520 INPUT "HORIZ.
    ROTATION?",T
530 INPUT "VERTICA
    L TILT?",F
536 GLCURSOR (K,-2
    00):SORGN
537 K=0
540 GOTO 460
700 INPUT "DIST UP
    ?",W
701 INPUT "DIST RI
    GHT?",U
702 INPUT "DIST BA
    CK?",V
703 INPUT "RADIUS?
    ",F
704 INPUT "HEIGHT?
    ",H
705 G(N)=26
706 TEXT :CSIZE 1:
    USING "###.###
    ":LPRINT N;W;U
    ;V
707 TAB 7:LPRINT "
    R= ";F;"  H= "
    ;H:LF 1
708 INPUT "MAJOR A
    XIS (X,Y,Z)?",
    B$
709 IF B$="X"THEN
    730
710 IF B$="Z"THEN
    750
714 FOR A=15TO 375
    STEP 30
715 K=1+A/30
716 X(N,K)=U+F*SIN
    A:Y(N,K)=-V-F*
    COS A:Z(N,K)=W
717 NEXT A
718 FOR K=14 TO 26
719 X(N,K)=X(N,K-1
    3):Y(N,K)=Y(N,
    K-13):Z(N,K)=H
    +W
720 NEXT K
721 GOTO 100
730 FOR A=15TO 375
    STEP 30
731 K=1+A/30
732 Z(N,K)=W+F*COS
    A:Y(N,K)=-V-F*
    SIN A:X(N,K)=U
733 NEXT A
734 FOR K=14TO 26
735 Z(N,K)=Z(N,K-1
    3):Y(N,K)=Y(N,
    K-13):X(N,K)=H
    +U
736 NEXT K
737 GOTO 100
750 FOR A=15TO 375
    STEP 30
751 K=1+A/30
752 X(N,K)=U+F*COS
    A:Z(N,K)=W+F*
    SIN A:Y(N,K)=V
753 NEXT A
754 FOR K=14TO 26
755 X(N,K)=X(N,K-1
    3):Z(N,K)=Z(N,
    K-13):Y(N,K)=V
    -H
756 NEXT K
757 GOTO 100
760 FOR K=1TO 13
761 LINE (R(K),T(K
    ))-(R(K+13),T(
    K+13))
762 NEXT K
763 RETURN
800 "DRAW"
810 FOR I=1TO G(M)
820 R(I)=X(M,I)*
    COS T-Y(M,I)*
    SIN T
830 T(I)=-X(M,I)*
    SIN T*SIN F-Y(
    M,I)*COS T*SIN
    F+Z(M,I)*COS F
840 R(I)=R(I)*S:T(
    I)=T(I)*S
842 NEXT I
844 C=C+1
845 IF C>3THEN LET
    C=0
847 GLCURSOR (R(1)
    ,T(1))
848 FOR I=2TO G(M)
850 LINE (R(I-1),T
    (I-1))-(R(I),T
    (I)),0,C(M)
860 NEXT I
865 IF B$(M)="C"
    THEN GOSUB 760
870 GLCURSOR (0,0)
880 RETURN
940 "E":TEXT :
    CSIZE 1
952 USING "###.###
    ":LPRINT N;W;U
    ;V
953 TAB 7:LPRINT Z
    ;X;Y
954 LF 1
955 RETURN
990 GOSUB 350
995 PAUSE "BY-BY!"
999 END
STATUS 1       2961
```

INSIDE THE PC-1500

This is the fourth part of the current series on the Sharp PC-1500. (Most of this information is also applicable to the Radio Shack PC-2.) This valuable data is supplied through the dedicated efforts of *Norlin Rober, 407 North 1st Avenue, Marshalltown, IA 50158.* This issue picks up from where we left off in Issue 33 of *PCN.* That is, with a continuation of a brief summary of a number of the routines contained in the ROM of the PC-1500. Specifically, Norlin was discussing the subroutine at address &D2EA that is capable of searching for a BASIC program line

A data byte must follow the instruction calling this subroutine. If the line sought is not found, this data byte is added to the return address, with UH containing 0B.

As an alternative, a search for a specified line number (but not for a label) may be begun at the address pointed to by START OF EDIT. This is done by subroutine D2E0, which is identical in all other respects to the above subroutine.

KEYBOARD

POLL OF KEYBOARD

Subroutine E42C loads A with the code for the key (other than BREAK) that is currently depressed. If no key is depressed, A will be loaded with zero. Register Y is unchanged by the subroutine.

The codes determined by the keys are as follows:

01	SHIFT	0F	OFF	19	RCL	2D	-	35	5	43	C	4B	K	53	S
02	SML	11	F1	1B	DEF	2E	.	36	6	44	D	4C	L	54	T
08	left	12	F2	1F	MODE	2F	/	37	7	45	E	4D	M	55	U
09	template	13	F3	20	SPACE	30	0	38	8	46	F	4E	N	56	V
0A	down	14	F4	28	(	31	1	39	9	47	G	4F	O	57	W
0B	up	15	F5	29	)	32	2	3D	=	48	H	50	P	58	X
0C	right	16	F6	2A	*	33	3	41	A	49	I	51	Q	59	Y
0D	ENTER	18	CL	2B	+	34	4	42	B	4A	J	52	R	5A	Z

WAIT FOR INPUT FROM KEYBOARD

Subroutine E243 polls the keyboard until a key code is obtained. Except for the SHIFT, SML, and DEF keys (which act as prefixes), the use of a key produces a return, with a character code in A. The 7-minute powerdown timer operates during the wait. Register Y is unchanged.

The codes obtained for display characters are their ASCII codes.

The codes for control and editing keys requiring the SHIFT prefix are as follows:

1A	CA	1C	INS	1D	DEL	1E	SHIFT/MODE

The codes for alphabet keys prefixed by DEF are their ASCII codes plus 40. The other keys accepting the DEF prefix use these codes:

80	DEF SPACE	9D	DEF =

The code for BREAK is 0E. If BREAK has been keyed, subroutine E243 will continue to return with this code in A each time it is called, until bit 1 of address F00B in the I/O Port is cleared.

DETECTION OF BREAK KEY

Call A6 (E451) will clear flag Z if BREAK has been keyed. There is no effect on the CPU registers or on other flags.

If BREAK has been keyed, this subroutine will continue to return with flag Z cleared each time it is called, until bit 1 of address F00B in the I/O Port is cleared.

DISPLAY

CURSOR POINTER

The contents of the Cursor Pointer (7875) specify a column of dots in the LCD display; columns are numbered from 00-9B (0-155 decimal).

TO CLEAR DISPLAY

Call F2 (EE71) clears the display. Registers X and Y are unchanged.

TO DISPLAY DATA IN R0 (OR STRING POINTED TO BY R0)

Subroutine E8CA may be used to display numeric data contained in R0 (given in decimal or binary), or a string pointed to by R0. Display will be in default format. Strings are displayed left-justified, and numeric data right-justified. Precede by storing 20 as the value of the Display Parameter (7880).

TO DISPLAY NUMERIC CONTENTS OF R0, FORMATTED

The display will be formed beginning at the location pointed to by the Cursor Pointer (7875). Contents of R0 must be in decimal.

(1) Store format specifications as follows:

7895: Editing characters. 10, comma separation; 20, forced sign; 40, asterisk fill; 80, scientific format

7896: Number of characters preceding decimal point, including spaces for sign (and commas, when specified).
7898: Number of characters including and following decimal point

(2) Load A with 00, 01, or 02, to specify the position of the characters to displayed, according to the following:

A=00: data displayed right-justified in 13-character block
A=01: data displayed left-justified in block of whatever size is required (up to a maximum of 16 characters)
A=02: data displayed right-justified in 26-character block

Then execute Call 96 (EA78), to form a string of characters (in the area 7A10-7A34) to represent the data to be displayed.

(3) Execute the instruction STX U; then execute Call 92 (ED00).

The cursor pointer will be updated to point to the next available display position. Flag C will be set if the display has been filled.

TO DISPLAY A STRING POINTED TO BY R0

Execute Call DC (DEBC), followed by Call 92 (ED00). The cursor pointer will be updated to point to the next available display position. Flag C will be set if the display has been filled.

TO DISPLAY CONTENTS OF OUTPUT BUFFER

Subroutine ECFA displays the contents of the Output Buffer starting at 7B60, ending when the display is filled. Characters are entered into the display beginning at the position specified by the Cursor Pointer.

TO DISPLAY A SEQUENCE OF CHARACTERS

Subroutine ED3B will display a sequence of characters whose codes are located anywhere in memory. Precede by loading U with the beginning address of the stored codes, and XL with the number of codes. The characters will be entered into the display beginning at its left end.

TO DISPLAY A PROGRAM LINE

The line to be displayed must be put into the Input Buffer, starting with its two-byte line number, omitting the link byte, and continuing with the codes for the BASIC statements in the line. A line may be placed into the Input Buffer in that form in any of the following ways:

(a) Subroutine D26F may be used to place the first BASIC line into the Input Buffer; the SEARCH pointer will be set to point to that line.
(b) Subroutine D2EA may be used to locate the line whose number has been loaded into U. (A data byte, which will be added to the return address if the specified line does not exist, must follow the instruction calling this subroutine.) After execution of subroutine D2EA, load XH and XL with the contents of 78A6 and 78A7; decrement X twice; and execute subroutine D2D0. The specified line will be in the Input

Buffer, and the SEARCH Pointer will Point to that line.
(c) The Program line that follows the one currently Pointed to by SEARCH may be Placed into the InPut Buffer, with automatic uPdating of the SEARCH Pointer, by execution of subroutine D2B3.

After the BASIC line has been Placed into the InPut Buffer, it is disPlayed by execution of subroutine E8CA, Provided that an aPProPriate value has been stored as the DisPlay Parameter (7880). With 10 as that valüe, the line is disPlayed from its start, without a colon following the line number in the disPlay; with 14, the colon will be included.

The cursor will be included in the disPlay if the DisPlay Parameter contains 50 (no colon) or 54 (colon included). Register Y must be loaded with the address (in the InPut Buffer) at which the cursor is to aPPear.

TO DISPLAY CONTENTS OF INPUT BUFFER, WITH TOKENS EXPANDED

Subroutine E8CA, with the DisPlay Parameter (7880) containing zero, disPlays the contents of the InPut Buffer, starting at 7BB0, ending when a 0D code is reached. (The InPut Buffer may be filled with 0D codes Preceding this, by execution of subroutine D02B.)

The cursor will be included in the disPlay if 40 is used as the DisPlay Parameter value; Register Y must be loaded with the address (in the InPut Buffer) at which the cursor is to aPPear.

TO DISPLAY A SINGLE CHARACTER

Subroutine ED4D enters the character whose code is in A into the disPlay, at the Position Pointed to by the Cursor Pointer (7875). The Cursor Pointer will be incremented by 6 each time a character is added to the disPlay, Pointing to the next character Position. If the right end of the disPlay is reached, additional characters wraP around to the left end of the disPlay.

The character whose code is in A may also be Placed into the disPlay by subroutine ED57. In this case, there will be no automatic increment of the Cursor Pointer.

GRAPHIC DISPLAY

Subroutine EDEF disPlays the dots sPecified by the contents of A, in the column Pointed to by the Cursor Pointer. The Cursor Pointer is not incremented automatically; Call 8E (EDB1) may be used to do so.

The dots disPlayed are determined by the bits contained in A:

01	ToP dot	10	5th
02	2nd	20	6th
04	3rd	40	Bottom
08	4th		

TIMED DELAY

Call AC (E88C) produces a delay, equal in duration to the product of 1/64 second by the contents of Register U. If the BREAK key is pressed during execution of this routine, the timing cycle ends, and there is a return with flag C set. Register Y is not affected.

This subroutine may be used to produce the equivalent of a PAUSE by effecting a temporary delay while information is being displayed.

BEEP

Subroutine E669 is equivalent to BEEP 1. It ignores BEEP ON/OFF.

Subroutine E66F produces a tone with pitch determined by UL, and duration determined by X. BEEP ON/OFF is ignored. The frequency of the tone is approximately 59230/(7.47+UL) cps (calculated in decimal). The duration (number of cycles) is &100 less than the contents of X.

POWERDOWN

Subroutine CD71 is equivalent to OFF.

Subroutine E33F is equivalent to the automatic 7-minute powerdown.

TERMINATION AT 'READY'

A machine-language program may be terminated by setting the computer to 'Ready' mode, in which it is awaiting further instructions from the keyboard.

TERMINATE WITH DISPLAY

Call 46 (CA8D) sets 'Ready' mode, retaining the display contents.

If subroutine E8CA has been used to produce a display with the cursor contained in it, the cursor left and cursor right keys will be enabled. (In other cases, however, use of the cursor control keys may result in a meaningless display.)

TERMINATE WITH PROMPT MARK

Call 42 (CA58) will place the computer in 'Ready' mode, with the prompt mark displayed.

POCKET COMPUTER NEWSLETTER
P.O. Box 232, Seymour, CT 06483

INKEY$, INPUT, INPUT #, INT, LEFT$, LEN, LET, LIST, LIST #, LLIST, LLIST $, LN, LOG, LPRINT, MEM, MEM #, MERGE, MID$, NEW, NEW #, NEXT, NOT, ON...GOSUB, ON...GOTO, OR, PASS, PAUSE, PI, PRINT, PRINT #, RADIAN, RANDOM, READ, REM, RESTORE, RETURN, RIGHT$, RND, RUN, SGN, SIN, SQR, STOP, STR$, TAN, TROFF, TRON, USING, VAL and WAIT.

Many of these will be familiar to even early PC-1211 users, but note that there are many new ones, such as those ending in $ (for string operations) and some of those ending in the # sign (such as EQU #) which are associated with the "Easy Simulation Programming" mode.

PC-1500 users as well as 1250 afficionados may also note that there are no PEEK or POKE directives. This may be a disappointment to those people who like to delve into machine language programming. It could be a real challenge - perhaps impossible - to "crack" into this PC and utilize it at the machine level. On the other hand, the lack of PEEK and POKE coupled with the fact that there is a PASS (password) statement, could be encouraging to software developers. The PASS command allows you to effectively protect a program so that it cannot be listed, modified or copied. In earlier models, such as the 1250, the PASS directive could be circumvented by an astute programmer using the PEEK and POKE directives. Since there are lots of astute programmers around these days (especially amongst readers of *PCN*), trying to protect a program using the PASS command has not been a very secure method in the past. The lack of PEEK and POKE in the 1261 may make the password option more effective as a security device.

PC-1500 programmers might note that statements associated with printing operations, such as LCURSOR, are absent. The 1261 appears to be more difficult to program for use with a printer such as the CE-125 or CE-126P. This appears to be due primarily to the fact that in normal use only two fields can be printed on a 24-character line. Alphanumeric (string) data is printed left-justified within a 12-character field. Numeric data is printed right-justified in a similar 12-character field. Two 12-character fields make up a 24-character printer line. This is somewhat restrictive. You might think this could be circumvented by the use of LPRINT USING statements. While the instruction manual suggests that numeric and alphanumeric data can be mixed within a printed line through LPRINT USING formats (such as ##.##&&&&&####&&&&&), tests indicate you can only switch from string data to numeric data (or vice-versa) once within a printed line! This makes it tough to print fancy stuff (do "pretty-printing").

[There is a way around this limitation. It requires that you convert numeric data into string format. You can then build up, say, a 24-character string-variable that combines several alphanumeric segments (including those segments that were converted from numeric format) under one variable name. This entire string can then be printed out by declaring a USING format that consists of, say, 24 alphanumeric characters.]

Considering that the PC-1261 was really designed to fit into a shirt pocket sans printer, it is not surprising that its printer-driving software is somewhat limited compared to, for instance, that of the PC-1500.

All-in-all, the PC-1261 seems to be a nice advance in the state-of-the-art. Having 10K of user memory available in a truly pocketable unit is nice. Being able to maintain your library of programs from the earlier PC-1211/PC-1250 models is definitely a plus factor. And, having advanced features such as 2-dimensional arrays and the "automatic calculation", (coupled with the extra memory) is a nice kicker. If you have been thinking about upgrading your pocket computing power, take a good look at the PC-1261. It may be just the solution to your advancing needs!

POCKET WORDPRO

This machine language program converts the Sharp PC-1500/PC-1500A or Radio Shack PC-2 into a simple word processor. Admittedly, it is stretching things a bit to call it a word processor. It will not relocate paragraphs, search for words, do automatic page numbering or print headers and footers. But, it is easy to learn to use and its capability may surprise you.

When the program is in use, all but about 500 bytes of RAM is used as a text storage area. Using a PC-1500A equipped with a CE-161 (16K) memory module, this means that you have over 22,000 bytes of character storage! You simply type in the text you want printed and then tell the computer to print it. Printing is done in vertical format, separated into pages of twelve 42-character lines each. The computer will determine where to end a line to avoid splitting a word and it will left-justify each printed line. You may indent paragraphs and center headings. However, centering of lines is not automatic.

Editing is simplified by the fact that you use the cursor controls the same way you are accustomed to using them -- with a few improvements! Insertion and deletion are auto-repeat. The computer will continue inserting or deleting as long as you keep the key held down.

Text may be saved, loaded or merged from cassettes, in the same simple way it is done for BASIC programs!

Loading the Program

You will need to load the ML codes in the listing, using either POKE statements or a Monitor program. It is 503 bytes long, but needs to be done only once. You can save the program on cassette. You must have either a CE-155, CE-159 or CE-161 memory module (8K or 16K expansion). Before testing the program, save it on tape using:

CSAVE M "POCKET WORDPRO";&3800,&39F6

Then, if you have made any errors in entering the program, at least you won't have to re-enter the whole listing.

After you have a correctly-entered program, you can relocate it by specifying a different CLOAD M address when loading. The best choice is made by loading with CLOAD M 256 * PEEK &7863. This will insure that you are using all the memory your computer has available. (This program will *not* work if loaded into the 7C00 - 7FFF area of the PC-1500A.)

Since PRO and REServe modes have no use with this program, the PC should be LOCKed in the RUN mode as a precaution against disaster.

Executing the Program

There are two addresses used in CALL statements. One is used when starting up the program and the other when continuing operations following a BREAK. The first of these is the beginning address at which the program is located. This would be &3800 when the program is located as shown in the accompanying listing. If you have relocated the program as suggested above, the address would be 256*PEEK &7863. When you begin execution with CALL &3800 (or CALL 256*PEEK &7863), you will clear the entire text area. Hence, you will want to be careful not to execute this call later on, unless you want to deliberately clear whatever text you have entered.

The second of the addresses to be used in a CALL is &381D (or &1D+256*PEEK &7863). To make things convenient, you can put this address into variable C. Then, if you need to continue the program after a BREAK (either intentional or unintentional), you can simply execute CALL C.

Entering and Editing Text

After you have started up the program will CALL &3800 (or, if relocated, CALL 256*PEEK &7863), you should see an underscore indicating the cursor position at the left end of the display. Start typing. If you want to indent your initial paragraph by 3 spaces, type 3 space before beginning your first sentence. All characters, including the quotation mark, are usable. If you want to enter any of the characters labeled above the REServe keys, you will need to SHIFT for them. Note that you may use the SML key in the usual way.

The cursor-left/right keys operate in the normal fashion, with auto repeat when held.

The up-arrow and down-arrow keys will change the displayed portion of text by 26 characters. They also automatically repeate when held down. If up-arrow is held, you can reach the beginning of text fairly quickly. Down-arrow may be used to locate the end of text. Note that the operation of these keys is somewhat similar to their operation when scanning back and forth through the lines of a BASIC program.

Insert and Delete operate about the same as they normally do. However, for multiple insertions or deletions, you do not have to repeat SHIFT with each use. Furthermore, the insertion and deletion continue rapidly when the corres- ponding key is held down. Thus you may delete sections of text rather quickly.

Keying ENTER will display a solid rectangle that is used to mark the end of a paragraph. This makes it easy to locate paragraph endings as you scan rapidly through the text using the up-arrow or down-arrow keys. Remember to type three spaces following the ENTER if you are indenting paragraphs. Note also that the ENTER key should be used immediately following the concluding punctuation mark of the last sentence in a paragraph.

To see how many bytes of unused memory are remaining, just hold down the RCL key. When you release this key, you will be back to where you were before.

To print the entered text, key DEF P. A blank line is printed between paragraphs, except at the top of a page.

Note that a sequence of more than 42 characters that are not separated by spaces will not fit on a 42-character line. In the unlikely event that such a sequence is encountered, the printer will stop. You will have to do appropriate editing to

Program *Word Processor ("WORDPRO") for the PC-1500/PC-2.*

```
3800 FD 58 FD 40    387C 0A 6C 0B 89    38FC B1 B0 FD CA    397C C6 89 52 ED
3804 B5 F5 FD CA    3880 14 B5 E6 FD    3900 84 A7 78 52    3980 77 4E 0F 8B
3808 CA 65 CA 50    3884 CA FD 42 F4    3904 81 04 4E FF    3984 09 E9 77 4E
380C CC 64 4A 00    3888 78 65 BE DF    3908 93 D6 A4 0E    3988 F0 05 8B 05
3810 46 CA 52 FD    388C E2 81 02 CC    390C CC 50 9E AD    398C 8E 16 45 89
3814 6A CC 65 BE    3890 65 5A B0 9E    3910 6C 19 89 17    3990 0E FD 8A DD
3818 DF E2 BE D3    3894 6E 6C 1C 89    3914 45 99 03 F4    3994 B7 C7 89 3D
381C C5 BE D0 2B    3898 04 B5 0C 8E    3918 78 52 64 BE    3998 FD C8 BE A9
3820 B5 40 AE 78    389C 06 6C 1D 89    391C DF E2 CD 10    399C D5 8E 49 B7
3824 80 CC 50 CA    38A0 48 B5 08 28    3920 00 BE EE EF    39A0 20 9B 15 46
3828 50 FD 98 5A    38A4 FD C8 FD 98    3924 BE E4 2C 99    39A4 CA 56 6A 00
382C B0 6A 19 F5    38A8 15 8B 28 14    3928 05 9E F7 6C    39A8 05 8B 1B 44
3830 88 03 FD 1A    38AC 5A B0 10 FD    392C 90 99 FB A5    39AC B7 7F 89 05
3834 BE E8 CA BE    38B0 CA 6C 08 8B    3930 B0 00 DD 9B    39B0 AE 77 4E 8E
3838 E2 43 28 CC    38B4 10 68 20 05    3934 0C FD 98 45    39B4 11 60 6E 2A
383C 50 81 09 45    38B8 2A A4 41 24    3938 99 03 DD AE    39B8 99 12 47 B7
3840 99 03 46 46    38BC 28 99 08 CC    393C 77 4E AE 79    39BC 20 8B 05 62
3844 CA 67 CD 42    38C0 52 49 00 8E    3940 F2 DD AE 79    39C0 99 08 9E 33
3848 EB 7B 0E 40    38C4 07 FD 6A 44    3344 F4 46 46 CA    39C4 44 44 FD A8
384C 6C 08 89 0C    38C8 45 61 99 04    3948 67 CC 65 CA    39C8 CA 54 FD 2A
3850 5E B0 56 99    38CC CC 50 6A 19    334C 54 EB 79 F0    39CC CC 56 BE B4
3854 21 54 46 05    38D0 F5 88 03 FD    3950 FF BE AC 97    39D0 F4 FD 8A 99
3858 99 33 9E 28    38D4 1A BE E8 CA    3954 B5 D8 FB B1    39D4 7F FD C8 48
385C 6C 0C 89 0C    38D8 BE E2 43 EB    3958 12 FD CB 48    39D8 7B 4A 7F B5
3860 15 9B 2F 54    38DC 7B 0E 40 28    395C 7B 4A 7F 43    39DC 00 43 43 B5
3864 5E CA 91 41    38E0 CC 50 FD 8A    3960 B5 00 43 43    39E0 C0 43 B5 FD
3868 44 56 9E 45    38E4 A6 9B 43 9E    3964 0E BE AC 07    39E4 0E BE AC 07
386C 6C 0A 89 0D    38E8 9D E9 7B 0E    3968 BE A7 69 CC    39E8 BE A7 69 FD
3870 B5 1A FD CA    38EC BF 6C 0D 89    396C 54 ED 77 4E    39EC 8A 9B 9E BE
3874 47 9B 03 DD    38F0 02 68 7F 6C    3970 F0 8B 0C E9    39F0 AC BB FD 1A
3878 9B 06 44 8E    38F4 80 83 19 6C    3974 77 4E 0F FD    39F4 9E CD 00
                    38F8 20 81 15 14    3978 8A FD C8 B7
```

correct this condition before trying to print again.

The BREAK key (actually, the ON key) may be used to discontinue execution of the program. There are two reasons for wanting to do this: (1) you want to save or load text on cassette, or (2) you want to quit and use your computer for some other task. It is also possible to unintentionally press this key. To get back into the program, you can execute CALL C -- provided that you have previously stored &1D + the starting address into variable C, as suggested earlier. Your text will then reappear in the display, right where you left off.

If you should use BREAK to interrupt the printer while it is busy printing your text, you should execute TEXT to reset the printer.

The MODE, CL, OFF and Reserve-Group-Selector keys are inoperative while the program is running.

Using Cassette Tape

Following a BREAK, the CSAVE directive may be executed (in immediate mode) to record the text that is stored in memory. The recording may also be verified using CLOAD? (As usual, the use of a specific file name is optional.)

A previously-recorded file of text may be loaded into memory using CLOAD (again, in immediate mode) following a BREAK. Text already in memory (if any) may first be cleared by the initialization routine (executed by CALL starting address).

Executing MERGE will append recorded text to that currently in memory. (It does *not* prevent you from editing earlier portions, as is the case with BASIC programs.)

Centering

A heading may be centered by treating it as a paragraph and indenting an appropriate number of spaces. For example, to center a 12-character heading, note that (with a 42-character line) there will be 30 spaces left over. Therefore, indent 15 spaces, then type in the heading and follow it by ENTER. (It will look a little bit funny if this centered heading happens to be located at the bottom of a page -- you just have to take your chances!)

Note that these centered heading may occur only: (1) at the very beginning of text, or (2) immediately following the end of a paragraph that has been terminated using the ENTER key.

To Quit

When you are finished using the program and want to do something else with your computer, key BREAK and then execute NEW 0.

Summary of Operation

1. Load program with CLOAD M 256*PEEK &7863.
2. Execute C=&1D + 256 * PEEK &7863.
3. Initialize with CALL 256 * PEEK &7863.
4. Start typing. (Or, key BREAK and load in previously-recorded text file using CLOAD, then CALL C.)
5. Cursor controls operate with auto-repeat. Key ENTER to end a paragraph. Hold the RCL key to see how much memory is available.
6. To print, key DEF P.
7. To save on tape, key BREAK, execute CSAVE, return to program via CALL C.
8. To quit press BREAK then key NEW 0.

Do you know of any other word processor whose instructions are that simple? Perhaps we ought to try keeping things in perspective, however. This isn't really a *serious* word processor, let's admit it. (How *could* it be, when there isn't even an apostrophe?) In spite of that, many users may find it useful.

Thanks for this clever machine language program should be directed to: *Norlin Rober, 407 North 1st Avenue, Marshalltown, IA 50158*

PERSPECTIVE PLOT

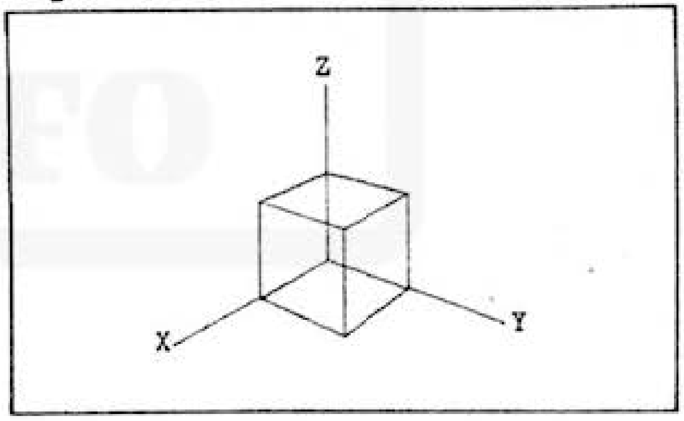
You can use this program on a Sharp PC-1500 or Radio Shack PC-2 to plot line segments and form a perspective drawing.

The line segments to be projected onto a plane of projection lie within a cube. The size of this cube is specified by the user in response to an INPUT prompt. The X, Y and Z coordinates of the end points of these segments must be specified (as positive numbers) in DATA statements starting at line 100. The letter S as a DATA item specifies the start of a new sequence of connected line segments. The final DATA item should be the letter E, which is interpreted as the end of the list. Here is an example illustrating the use of DATA statements to produce a cube:

```
100 DATA 0,0,1,1,0
    ,1,1,0,0,0,0,0
    ,0,0,1,0,1,1,1
    ,1,1,1,1,0
101 DATA 0,1,0,0,1
    ,1,S,1,1,1,1,0
    ,1,S,0,0,0,0,1
    ,0,S,1,1,0,1,0
    ,0,E
```

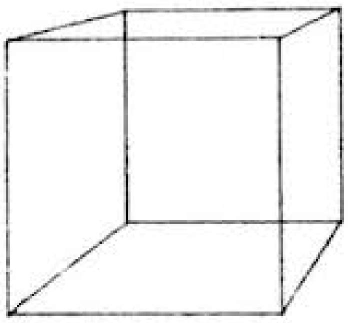
The relationship between the X, Y and Z coordinates is illustrated in the accompanying diagram.

Diagram *X, Y and Z Coordinates for Simple Cube.*

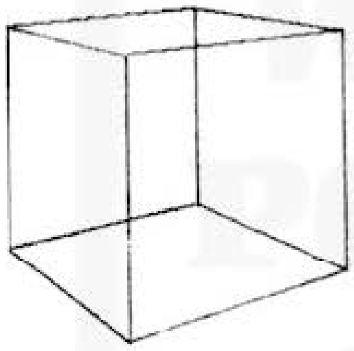


The point from which the cube (or object within a cube) is to be viewed is specified by the user in response to prompts. Its location, *relative to the center of the cube,* is L units in the X- direction, M in the Y-direction and N in the Z- direction. Note that L, M and N are measured in the same units as the X, Y and Z coordinates. The view point may be chosen in any direction from the center of the cube containing the object projected. However, it may

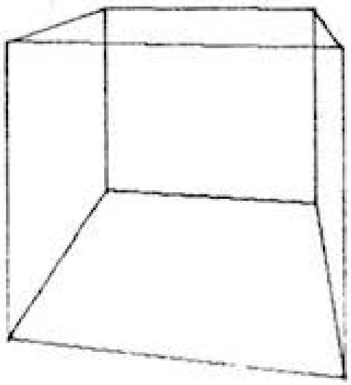
Diagram *Data and Several Sample Perspectives.*



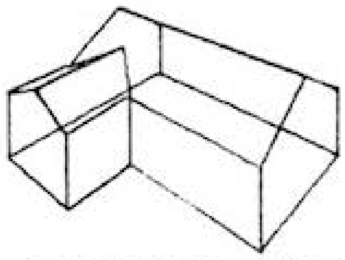
```
1-PT PERSP, FROM
4, 1.5, 1
```



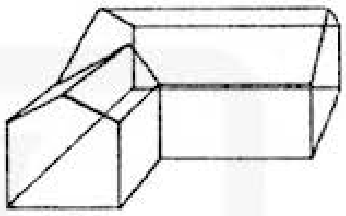
```
2-PT PERSP, FROM
3,-2, 1
```



```
2-PT PERSP, FROM
2, 0.3, 0.8
```



```
2-PT PERSP, FROM
75, 65, 50
```



```
1-PT PERSP, FROM
150, 60, 55
```

Diagram *Several Perspectives of a Cube.*

```
100:DATA 5,50,0,5,
    50,15,20,50,25
    ,35,50,15,35,5
    0,0,5,50,0
101:DATA 5,35,50,0
    ,35,20,0,50,20
    ,0,50,0,0,50,0
    ,15,50,10,20,5
    0,20,15
102:DATA 35,20,15,
    35,50,15,5,35,
    20,0,35,20,15,
    27.5,10,20,35,
    0,15,20,0,25,2
    0,50,25
103:DATA 5,50,0,0,
    5,0,0,5,50,0,5
    ,5,0,0,5,0,15,
    5,5,50,15,5,0,
    15,20,0,25
104:DATA 5,50,0,15
    ,35,0,15,5,27.
    5,10,20,50,10,
    20,50,20,15,50
    ,20,0,E
```

Program *Perspective Plot*

```
10:INPUT "VIEW PO
   INT: L? ";L,"M
   ? ";M,"N? ";N
11:INPUT "1 OR 2-
   POINT? ";T
12:INPUT "CUBE SI
   ZE? ";D:D=D/2
13:CLS :IF T=2
   THEN 30
20:IF ABS L<=D
   THEN 60
21:A=1/(1-D/ABS L
   ),B=1/(1+D/ABS
   L),C=1/(ABS L+
   ABS M)
22:K=108*C/D,HX=K
   *M,HY=K*L
23:UX=K*N*SGN L,U
   Y=0,UZ=K*ABS L
24:P=A*B:IF ABS M
   <DLET P=A*C*
   ABS L
25:PX=P/L,PY=0,PK
   =P+P*D/L
26:H0=108*A/P:IF
   L*M<0LET H0=21
   6-H0
27:K=-108*C/P:IF
   N>DLET U0=K*B*
   (ABS L+N),UB=2
   *U0*A:GOTO 40
28:U0=K*A*(ABS L-
   N):IF N>-DLET
   UB=2*K*A*ABS L
   :GOTO 40
29:UB=2*U0*B:GOTO
   40
30:IF ABS L<=DAND
   ABS M<=DTHEN 6
   0
31:A=D*(ABS L+ABS
   M),B=D*ABS (
   ABS L-ABS M),C
   =L*L+M*M
32:K=108/A,HX=K*M
   ,HY=K*L
33:K=K/√(C+N*N),U
   X=K*N*L,UY=K*N
   *M,UZ=K*C
34:IF ABS L<DOR
   ABS M<DLET P=(
   B/A/(C-A)+1/(C
   -B))/2:GOTO 36
35:P=(1/(C+B)+1/(
   C-B))/2
36:PX=P*L,PY=P*M,
   PK=P*(C+(L+M)*
   D)
37:H0=108/P/(C-B)
   :IF M*SGN L>L*
   SGN MLET H0=21
   6-H0
38:S=A*SGN (N-D):
   U0=-K/P*(C*D+N
   *S)/(C+S):IF
   ABS N>DLET UB=
   2*C*U0/(C-S):
   GOTO 40
39:UB=2*C*U0/(C-A
   *N/D)
40:HK=(HX-HY)*D,U
   K=(UX+UY-UZ)*D
   ,U0=U0-20,S=1E
   99,E=2E99,F=0:
   RESTORE 100:
   GRAPH :SORGN
41:READ X:IF X>=S
   THEN 45
42:READ Y,Z:P=PK-
   PX*X-PY*Y,H=H0
   +(HK-HX*X+HY*Y
   )/P,U=U0+(UK-U
   X*X-UY*Y+UZ*Z)
   /P
43:IF FLINE -(H,U
   ):GOTO 41
44:GLCURSOR (H,U)
   :F=1:GOTO 41
45:F=0:IF X=STHEN
   41
50:GLCURSOR (0,UB
   -60):TEXT :
   PRINT "KEY ENT
   ER TO PRINT"
51:LPRINT T;"-PT
   PERSP, FROM":
   LPRINT L;",";M
   ;",";N:END
60:PRINT "NO PLOT
    POSSIBLE":END

STATUS 1     1131
```

not be located *within* the cube. If the view point is chosen close to the cube, the resulting drawing will appear distorted. This is particularly true when N is large compared to L and M. In extreme cases the vertical range of the printer/plotter may be exceeded. For 1-point perspectives, the most pleasing views are obtained when L is at least twice as large as either M or N.

The plotted result will be scaled by the program to automatically produce a projection that spreads the cube over the full width of the plotting area.

The accompanying diagram shows examples of perspective views of a cube, plotted using the DATA statements listed earlier. The user should input 1 in response to the CUBE SIZE? prompt. Another figure illustrates a second example. In it, 50 should be input for the cube size. (For a rear view of the house, try -75, 65, 50 as L, M and N in a 2-point perspective. For an end view, try 0, 1000, 0.)

In a 1-point perspective projection, the plane of projection is parallel to one vertical face of the cube. In a 2-point perspective, the plane is perpendicular to a horizontal line connecting the center of the cube with the vertical line through the view point. In either case, the resulting projection is what a photograph would show, provided that the camera's film plane is parallel to the plane of projection.

Although entering DATA statements specifying coordinates can be a tedious process, once it has been done a wide variety of views of the object may be drawn. After you have determined the best view point, you wish to delete the DATA statements that involve hidden lines.

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

ALTERNATE CHARACTER SETS

This program, written entirely in machine language, provides for a convenient way of creating characters for display.

When you use this program there is no need to calculate binary codes. You simply use cursor controls and the decimal-point key to create the desired character in the display. Once characters have been defined they may be assigned to shifted keys for later recall.

Generation of Alternate ASCII Codes

In normal use, the keyboard input routine in ROM obtains an ASCII code from tables in ROM. The table for unshifted keys is located in addresses FE80 - FEBF. Shifted keys occupy FEC0 - FEFF.

When bit 2 of address 764E is set (as indicated by the Japanese Katakana annunciator), codes are obtained from alternate Key-to-ASCII tables. These tables are located at nn80 - nnBF and nnC0 through nnFF, where nn is one less than the byte stored in RAM address 785E. Keying SML twice will cancel the alternate keycode mode.

Alternate Display Characters

A ROM table containing codes for generating the display characters assigned to ASCII codes is located in FCA0 - FE7F. Each character is specified using 5 bytes. Normally, codes 80 - FF produce (by default) the same characters as the ASCII codes do. However, if RAM location 785D contains hexadecimal 80, an alternate table is used to obtain display codes for characters represented by codes greater than &80. Since codes above &80 are not ASCII codes, this table forms an extension of, rather than a replacement for, the set of available characters.

This alternate table begins at address nn00, where nn is the byte contained in address 785E.

Whenever the computer is powered up following a power down using the OFF key, the contents of 785D will be set to FF, thereby causing the alternate display mode to be cancelled.

An Alternate Character Set Program

Execution of CALL &7C72, when this program is in memory in a PC-1500A, will set both the alternate key code and alternate display modes.

Shifted alphabet and numeral keys will generate the codes A1 to C4 (rather than the usual ASCII codes for lower-case characters). These codes are displayed as characters that have been defined by the user. The ASC and CHR$ functions also apply to these codes.

After the alternate key code and alternate display modes have been set by CALL &7C72, execution of CALL &7D00 initiates a routine for formation of user-defined characters. The display will blank. Depress the key (alphabet or numeral) to which a character is to be defined. The symbol for that character will appear, followed by a 5 by 7 grid. The position of a "dot cursor" within this grid is indicated by an unlit dot.

The right and left cursor keys may be used to move the "dot cursor" through this grid. Keying the decimal-point key will enter a dot in the position indicated by the dot cursor, into the character being formed. Keying SPACE will clear a dot. The character being formed will be displayed in the position to the right of the dot cursor grid.

When the desired character has been completed, key ENTER to store it into memory. It will be displayed whenever the assigned key (following SHIFT) is depressed.

If BREAK is used to exit the character-creation mode, the character will not be assigned.

Using the Program with the PC-1500

The preceding instructions and listing of the machine language program apply to the Sharp PC-1500A. Since the earlier Sharp PC-1500 and the Radio Shack PC-2 do not contain RAM in the address range 7C00 - 7FFF, a different location is required. The program itself is relocatable without modification. However, its possible locations are restricted by the necessity to locate the Key-to-ASCII table and the Character Table at locations compatible with the PC's operating system routines.

The following locations are suggested for various memory configurations:

1. PC-1500 with CE-155 memory module. First assign the BASIC area with NEW &3B59. Enter the machine language program beginning at &3972, with it ending at &3AA3. (The character table will use the area &3AA5 - 3B58.) To execute the mode-setting routine, use CALL &3972. For the character-setting routine: CALL &3A00.
2. PC-1500 with CE-159 memory module. Begin with NEW &2359. Load the machine language program in the area &2172 - 22A3. The character table will occupy &22A5 - 2358. To execute the mode-setting routine use CALL &2172. For character creation, use CALL &2200.
3. PC-1500 with CE-161 memory module. Begin with NEW &0359. Load the machine language

program into the &0172 -02A3. The character table will occupy &02A5 - 0358. Set the mode using CALL &0172. For character creation use CALL &0200.

4. PC-1500 unexpanded. Use NEW &4359. Load the machine language program into &4172 - 42A3. The character table will occupy &42A5 - 4358. Set the mode using CALL &4172. For the character-setting routine: CALL &4200.

Location of Stored Character Codes

In the PC-1500A version, the codes that produce the characters defined by the user are stored in the area &7DA5 - 7E58. They may be viewed by using PEEK. These are the codes that can be used with GPRINT statements to produce the same characters.

The accompanying table provides the following information: 1) the address of the first of the five bytes that are used to generate the character, 2) the key to which that character is assigned, and 3) the code for the character, acting like an ASCII code.

Table *Character Codes.*

ADDR	KEY	ASCII
7DA5	A	A1
7DAA	B	A2
7DAF	C	A3
7DB4	D	A4
7DB9	E	A5
7DBE	F	A6
7DC3	G	A7
7DC8	H	A8
7DCD	I	A9
7DD2	J	AA
7DD7	K	AB
7DDC	L	AC
7DE1	M	AD
7DE6	N	AE
7DEB	O	AF
7DF0	P	B0
7DF5	Q	B1
7DFA	R	B2
7DFF	S	B3
7E04	T	B4
7E09	U	B5
7E0E	V	B6
7E13	W	B7
7E18	X	B8
7E1D	Y	B9
7E22	Z	BA
7E27	0	BB
7E2C	1	BC
7E31	2	BD
7E36	3	BE
7E3B	4	BF
7E40	5	C0
7E45	6	C1
7E4A	7	C2
7E4F	8	C3
7E54	9	C4

Program *Alternate Character Sets.*

```
7C72 FD 58 84 DD     7D0E FD C8 8E ED
7C76 0A 48 80 CA     7D12 57 EC 41 41
7C7A 5D EB 76 4E     7D16 41 4A 04 DF
7C7E 04 9A 0B 4E     7D1A 43 43 43 43
7C82 59 01 48 38     7D1E DF 0E B5 0A
7C86 35 32 09 58     7D22 AE 78 75 2A
7C8A 57 11 53 0F     7D26 58 7A 5A 00
7C8E 2D 2E 30 4D     7D2A 55 8E ED EF
7C92 55 15 4A 37     7D2E CD 8E 88 08
7C96 34 31 0D 28     7D32 8E E2 43 C3
7C9A 49 16 48 4F     7D36 42 CD 54 4A
7C9E 4C 29 19 43     7D3A 00 5A 06 6A
7CA2 45 12 44 2F     7D3E 04 87 2E 88
7CA6 2A 28 20 56     7D42 25 B7 20 88
7CAA 52 13 46 50     7D46 2A B7 0D 88
7CAE 08 3D 02 5A     7D4A 42 B7 0C 88
7CB2 51 18 41 18     7D4E 29 B7 08 99
7CB6 1F 0C 0A 42     7D52 21 45 87 FF
7CBA 54 14 47 39     7D56 98 05 46 48
7CBE 36 33 58 AE     7D5A FF 42 93 3F
7CC2 89 01 A8 C3     7D5E 4A 04 F8 D1
7CC6 C0 8D 09 B8     7D62 93 45 85 BF
7CCA B7 21 B3 0F     7D66 9E 49 45 BD
7CCE 2C 2E BB AD     7D6A FF 18 51 88
7CD2 B5 25 AA C2     7D6E 07 8E 05 45
7CD6 BF BC 0D 3C     7D72 19 51 88 05
7CDA A9 26 AB AF     7D76 4A AA 45 A7
7CDE AC 3E 19 A3     7D7A FF 98 05 46
7CE2 A5 22 A4 3F     7D7E 48 FF 44 4E
7CE6 3A 3B 5E B6     7D82 05 91 66 4A
7CEA B2 23 A6 B0     7D86 00 D8 9D 6B
7CEE 1D 40 02 BA     7D8A D9 9E 6E FD
7CF2 B1 18 A1 1A     7D8E 8A B1 41 83
7CF6 1E 1C 5D A2     7D92 02 83 28 FD
7CFA B4 24 A7 C4     7D96 58 4A A5 FD
7CFE C1 BE BE EC     7D9A CA D9 D9 FD
7D02 A2 BE E4 2C     7D9E CA 55 41 88
7D06 B7 30 91 07     7DA2 04 9A
7D0A B7 3D 9B 0B
```

SAVEing the Program on Tape

The program may be saved on tape using the CSAVE M statement. For the PC-1500A execute: CSAVE M "ALT CHAR SET",&7C72,&7DA3. (If you want to save the characters you have created, along with the program, then use the command: CSAVE M "ALTCHAR SET";&7C72,&7E58.)

If you are using the PC-1500 or PC-2, the addresses in the CSAVE M statement will need to be modified to coincide with the area of memory used. For example, when using the CE-155 memory module, save the programing using: CSAVE M "ALT CHAR SET";&3972,&3B58.

To reload the program, simply execute: CLOAD M.

A Suggestion

You can save a lot of typing by assigning the two CALL statements to a couple of REServe keys.

Norlin Rober, 407 North 1st Avenue, Marshalltown, IA 50158, provided this program.

INSIDE THE PC-1500

This is the fifth part of the current series on the Sharp PC-1500. (Most of this information is also applicable to the Radio Shack PC-2.) This valuable information is supplied through the efforts of *Norlin Rober, 407 North 1st Avenue, Marshalltown, IA 50158.* This issue continues from Issue 34 of *PCN.* Norlin has been describing a number of the ROM routines. Many of the functions they perform are of value to machine language programmers when called as machine language subroutines. Norlin has spent a great deal of effort deriving this information and taking the time to write about it. If you use it, try and find time to let him know how much you appreciate it!

TERMINATE WITH ERROR MESSAGE

The following routines will terminate a machine-language program with an ERROR message displayed in 'Ready' mode:

(a) Call E0 (CD88) displays ERROR n, where n is contained in UH

(b) Call E4 (CD89) displays ERROR 1

When the ERROR message is displayed, the up arrow key may be used to display a BASIC line at the address pointed to by Y. A meaningless display will result unless an effort has been made to load Y to point to the desired location.

Register Y can be made to point to the location in BASIC from which the machine-language program was called by either of the following:

(a) Provided that the stack has been kept balanced, POP Y twice.

(b) Load register S with 784A; then execute POP Y.

PRINTER ROUTINES

MOTOR OFF

Subroutine A769 is required following certain printer subroutines.

EQUIVALENTS OF CERTAIN BASIC STATEMENTS

TEXT	Subroutines ACBB, A9D5, A769
GRAPH	Subroutines A9D5, A769, ABEF; store FF into 79F0
CSIZE n	Store n (1 to 9) into 79F4
COLOR n	Subroutine A519. Precede by loading UL with n (0 to 3). Follow by subroutine A769 (Motor off)
LF n	Subroutine AA04. Precede by storing n (in signed binary) into 7B7F-7B80, and loading XH with 7B, XL with 7F.

	Follow by subroutine A769 (Motor off)
LCURSOR n	Subroutine AC52. Precede by loading XH with FF, and XL with n. Follow by subroutine A769 (Motor off)
SORGN	Subroutine AC97
ROTATE n	Store n (0 to 3) into 79F2

PRINTING IN TEXT MODE

(1) To print contents of R0 (or string pointed to by R0), using default format:

Store 04 into 788E; execute subroutine B3EA.

Numeric data is printed right-justified; strings, left-justified. Printer advances to next line if necessary. If CSIZE set is larger than 2, it will be changed to 2. TEXT mode will be set. A carriage return (with line feed) will follow printing.

(2) To print contents of R0 (or string pointed to), formatted:

(a) Store format specifications into memory, as follows:
 7BAA: Editing characters. 10, comma separation; 20, forced sign; 40, asterisk fill; 80, scientific format
 7BAB: Number of characters preceding decimal point, including spaces for sign and commas (when specified)
 7BAC: Maximum number of characters to be printed (in printing of character strings); if no limitation, zero
 7BAD: Number of characters including and following decimal point
(b) Store zero into 79EA and 79F5
(c) Execute subroutine B49A (Print); optionally, subroutine A9F1 (CR with line feed); subroutine A769 (Motor off)

(3) To print contents of R0 (or string pointed to by R0), formatted, starting at current pen location, with line advance when necessary:

(a) Store format specifications into 7BAA-AD, as described above
(b) Store 00 into 79EA, 08 into 79F5
(c) Execute subroutine B49A. Follow by subroutine A769 (Motor off)

(4) To print a single character (no carriage return or line feed):

(a) Store the ASCII code of the character to be printed into 7B7F
(b) Load XH with 7B, XL with 7F. Store zero into 79EA
(c) Execute subroutine A781. Follow by subroutine A769 (Motor off)

(5) To execute carriage return with line feed:

Load XH with 7B, XL with 7F. Execute subroutines A9F1, A769

PLOTTING (GRAPH MODE)

(1) To move pen, using given x and y increments:

(a) Specifying increments in two-byte signed binary, store the x-increment into 7B7E-7B7F, and the y-increment into 7B7C-7B7D

(b) Load XH with 7B, XL with 7C
(c) Execute subroutine AD78. Follow by subroutine A769 (Motor off)

(2) To move pen to location having coordinates (x,y):

(a) Specifying x and y in two-byte signed binary, store x into 7B7E-7B7F, and y into 7B7C-7B7D
(b) Load XH with 7B, XL with 7C
(c) Execute subroutine AC07. Follow by subroutine A769 (Motor off)

(3) To draw a line, starting from current pen location, using given increments in x and y:

(a) Specifying increments in two-byte signed binary, store the x-increment into 7B7E-7B7F, and the y-increment into 7B7C-7B7D
(b) Load XH with 7B, XL with 7C. Store FF into 79E9; store line type (0 to 9) into 79EA; store 01 into 7B84
(c) Execute subroutine AD07 (Draw line) and A769 (Motor off)

(4) To draw a square, taking current pen location as lower left corner:

(a) Load A with length of a side of the square
(b) Load XH with 7B, XL with 7F. Store line type (0-9) into 79EA
(c) Execute subroutine AC2F (Draw square) and A769 (Motor off)

(5) To draw a rectangle, beginning at current pen location:

(a) Specifying dimensions in two-byte signed binary, store the x-dimension into 7B7E-7B7F, and the y-dimension into 7B7C-7B7D
(b) Load XH with 7B, XL with 7C; store FF into 79E9; store line type (0 to 9) into 79EA
(c) Execute subroutine AD64 (Draw rectangle) and A769 (Motor off)

PRINTING IN GRAPH MODE

(1) To print contents of R0 (or string pointed to by R0), formatted, starting from present pen location, in direction specified by ROTATE:

(a) Store format specifications into 7BAA-AD, as in TEXT mode
(b) Store zero into 79EA and 79F5
(c) Execute subroutine B433. Follow by subroutine A769 (Motor off)

(2) To print a single character, in direction specified by ROTATE:

(a) Store the ASCII code of the character to be printed into 7B7F
(b) Load XH with 7B, XL with 7F. Store zero into 79EA
(c) Execute subroutine A781. Follow by subroutine A769 (Motor off)

PRINTING PROGRAM LINES (TEXT MODE)

(1) To place a program line into the Input Buffer prior to printing:

The line to be printed must be put into the Input Buffer, starting with its two-byte line number, omitting the link byte, and continuing with the codes for the BASIC statements in the line. A line may be

Put into the Input Buffer in that form in any of the following ways:

(a) Subroutine D26F may be used to place the first BASIC line into the Input Buffer; the SEARCH pointer is set to point to that line.

FROM THE WATCH POCKET

In a recent column in the magazine InfoWorld, John Gantz lamented the demise of the pocket and notebook computer field. Much of his pessimism apparently evolved from a personal experience wherein he tried to dispose of some 200 Texas Instrument Compact 40 handheld computers. We don't blame him for possibly being a little disappointed with the CC-40, but we do not think that the pocket and portable computer field is about to fade away.

Indeed, the signs are evident that all is well in the PC business despite recent withdrawals from the market by some suppliers. It does seem to currently appear that we are in a consolidation phase. Suppliers who can't match consumer's tastes are dropping out. Products are improving. New genre are being identified. The recently announced Sharp PC-1260 and 1261 models are good examples of this forward progress. The 1261, with its 10K of user RAM and 40K of built-in ROM software, points to the future. Here is a pocket computer that fits easily in a shirt pocket, yet contains enough power to really ease one's daily task. 10K of user memory is more than enough to hold several practical programs for use on a daily basis. The automatic calculation capability of the 1260 and 1261 are examples of the types of features that will be provided as standard in pocket computers of the future.

Pocket computers, now entering their fifth year of existence, are still in the pioneer stages. Manufacturers are beginning to learn just what it is that users want from their PC's. Models recently or about to be announced will provide more and more of what users seek. For example, the recently announced Casio FX-750P model and the upcoming Sharp PC-1350 allow for wafer-thin memory modules, complete with battery backup, to be easily plugged into the pocket computer. These new modules will make it easier for users to access a large variety of programs. They represent a significant step in the development of the PC towards becoming a practical tool for users from all walks of life.

Opportunities abound for visionaries and entrepreneurs who understand the vistas that are opening. While the society as a whole is still struggling with the legal concepts that need to be in place to protect both the consumer and producer in areas such as software copyright and protection, new technologies will make it practical to distribute information processing tools. For instance, the Sharp PC-1260 or 1261 can be password protected. This means that those with specialized knowledge or techniques can program this information into PC's and sell the entire package as a ready-to-use application tool. This can be accomplished with relatively low risk of having the source code and methodology comprised unless authorized by the program author. In the case of newer models, such as the PC-1350, application programs can be distributed installed directly in the battery backed RAM modules. Once again, various levels of source code protection can be provided at the author's discretion.

It should be noted that each generation of pocket computer products provides benefits in several areas. Not only is the hardware itself more powerful, but deficiencies in operating modes and capabilities are being corrected as a result of feedback from earlier users. And these early users have been speaking out. Hence, multiple line displays, which provide for the easier comprehension of data, will become the norm. Feedback from users indicates they would rather have several lines of information, even though the characters may be smaller, than having a single line that forces prompts to be remembered while data is being entered or leaves the user guessing about the interpretation of an output value.

In case you haven't heard, Sharp now produces a low cost combination thermal printer and cassette interface called the Model CE-126P. The list price of this unit is $94.95. The machine contains a 24 digit thermal printer as well as a standard audio-cassette interface. Originally designed for the EL-5500 and EL-5510 calculator units, this

FOR PC-1500 & PC-2 USERS

MACHINE LANGUAGE PROGRAMMING

Readers of the series *Machine Language Programming the Sharp PC-1500 and Radio Shack PC-2* (produced in 1983 as a separate publication by *PCN*) will especially appreciate the following article. As such readers know, the series was abruptly terminated due to serious illness on the part of the author.

What follows is a sample of how the author had planned to continue the series by building upon the instruction set foundation that had been laid in the early installments. Enjoy!

MACHINE LANGUAGE PROGRAMMING THE SHARP PC-1500 AND RADIO SHACK PC-2 POCKET COMPUTERS

Once a truly interested ML fan acquires an understanding of the types of instructions that are available on a machine, he or she soon develops an itch to do some real ML programming. Let's satisfy that urge right now.

Clearing Memory

When batteries are installed in a PC, the individual bits in memory will "come up" in random states. Some set to the logic 1 state, some cleared to the 0 state. Sometimes it is not desirable to have areas of RAM in such a chaotic condition. One way for a ML programmer to set an area in memory to a known conditon is to fill it with zero bytes. That is, load bytes set to the value zero into a specific range of memory addresses. Sounds like a pretty simple procedure, right? It is! And it can be done in a whole lot of different ways.

Suppose, for example, that you wanted to clear out the section of memory normally used to store the fixed string variables A$ through D$. These are stored in memory addresses &78C0 - &78FF.

One way to accomplish this job would be to load CPU register A with the value zero. Next, the starting address of the area to be cleared might be set up in CPU register X. Once this had been done, STAI (X) directives could be used to stuff the contents of the accumulator (register A) into successive locations in memory as pointed to by the contents of CPU register X. Remember, the STAI (X) instruction automatically advances the value in X each time it is executed. There is just one more parameter to consider. How many times must the STAI (X) directive be repeated in order to clear out the desired block of memory?

Since the size of the memory block is known in this example, a counter could be established, say in register UL. Each time the STAI (X) command was performed, the count in UL could be decremented. When this count reached zero, it would be time to stop stuffing zeros into memory. Thus, one could use the sequence of directives (after the STAI (X) instruction) consisting of: DEUL and RBNZ #nn. That is, decrease the count in register UL and if it is still non-zero, then loop back to repeat the STAI (X) directive. If this method was used, in this example, then register UL would initially have to be set to the decimal count of 64 (hexadecimal 40), representing the number of bytes in memory (from &78C0 through &78FF) that were to be cleared. See the accompanying listing for a detailed example of this method.

If you were wide awake when you read the previous section of this series, then you might remember that this type of situation is ideal for the use of the BNZD #nn instruction. Since, however, the BNZD directive tests for zero *before* it decrements the contents in UL, then the count initially placed in UL must be one less than the number of loops (locations to be cleared). Thus, in this specific example, if BNZD was used, register UL would need to start out with a count of decimal 63 (which is hexadecimal &3F). An accompanying listing illustrates this method, too.

Yet another way of determining when to stop looping would be to check for an appropriate address value in CPU register X. In this case, when the value in X exceeded &78FF (i.e., reached &7900), then it would be time to discontinue the clearing operation. There are several ways this type of procedure might be implemented.

One way would be to set the ending value (say &78FF in this case) into another CPU register. CPU register Y would be available for such use in this example. A comparison could then be made between the contents of X and Y each time that X was advanced by the STAI directive. When the value in X exceeded that in Y, then it would be appropriate to stop the clearing operation. See the

Program *Routine for Clearing Memory Using RBNZ Instruction.*

PG	LC	B1	B2	B3	B4	B5	LABELS	MNEMONICS	COMMENTS
00	00	B5	00				CLRMM1	LDA #00	Load register A with zero.
00	02	48	78					LDXH #78	Set up register X to point to 16-bit address &78C0.
00	04	4A	C0					LDXL #C0	Set up register X to point to 16-bit address &78C0.
00	06	6A	40					LDUL #40	Set up counter (64 decimal here) in register UL.
00	08	41					CLRM1	STAI (X)	Stuff contents of A into memory, then increment pointer.
00	09	62						DEUL	Decrement the counter value in UL (UL=UL-1).
00	0A	99	04					RBNZ CLRM1	If counter is not zero, jump back to stuff another byte.

Program *Routine for Clearing Memory Using BNZD Instruction.*

PG	LC	B1	B2	B3	B4	B5	LABELS	MNEMONICS	COMMENTS
00	00	B5	00				CLRMM2	LDA #00	Load register A with zero.
00	02	48	78					LDXH #78	Set up register X to point to 16-bit address &78C0.
00	04	4A	C0					LDXL #C0	Set up register X to point to 16-bit address &78C0.
00	06	6A	3F					LDUL #3F	Set up counter (63 decimal here) to count-1 (64-1=63).
00	08	41					CLRM2	STAI (X)	Stuff zero byte into memory, advance memory pointer in X.
00	09	88	03					BNZD CLRM2	Check UL for zero, loop back if non-zero, (UL=UL-1).

Program *Routine for Clearing Memory Using Compare Operation to Terminate Loop.*

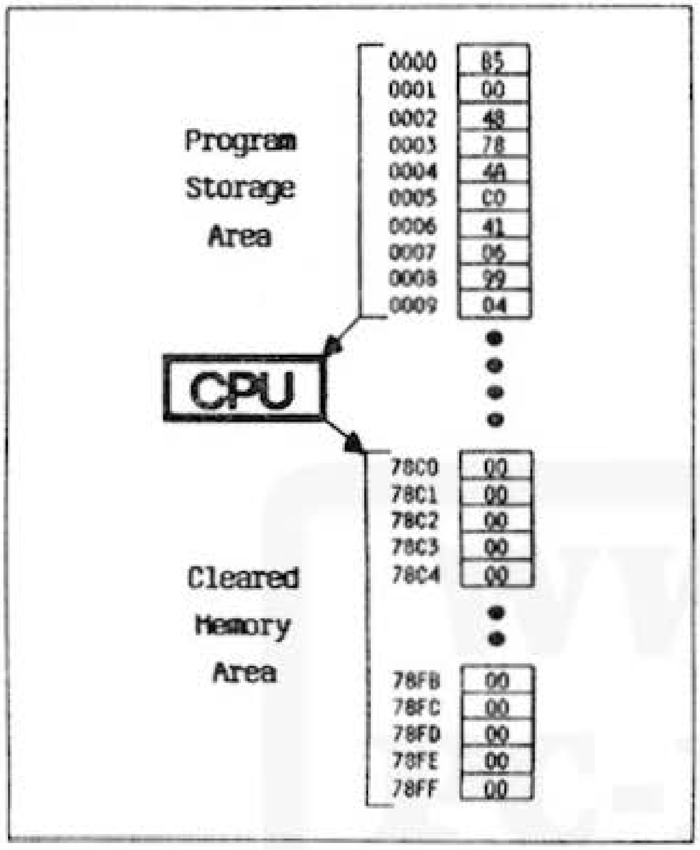
PG	LC	B1	B2	B3	B4	B5	LABELS	MNEMONICS	COMMENTS
00	00	B5	00				CLRMM3	LDA #00	Load register A with zero.
00	02	48	78					LDXH #78	Set up register X to point to 16-bit address &78C0.
00	04	4A	C0					LDXL #C0	Set up register X to point to 16-bit address &78C0.
00	06	41					CLRM3	STAI (X)	Stuff zero byte into memory, advance memory pointer in X.
00	07	06						CPA XL	Check XL for zero (indicating address &7900 reached).
00	08	99	04					RBNZ CLRM3	Loop back to stuff next location if XL is not zero.

example program listing.

An even easier way, in this particular case, would merely be to test for the value in XL going to zero. That is because, when XL exceeds FF (because X reaches 78FF), it will go to the value 00 as X advances to the value 7900. That just happens to be the point at which we want to stop clearing memory in this particular example! A listing of this method is also provided for examination.

By now you should be convinced that ML programming is not an exact science. There are usually countless ways to approach a particular objective. Sometimes you can customize your approach depending on specific parameters. For instance, if you are trying to conserve your use of memory, you might try to devise a sequence of directives that uses a minimum amount of space. Or, you might be interested in maximizing speed of program execution. In that case, you could work towards selecting directives that could be performed in the minimum amount of time. (Do *not* make the assumption that the fastest program is the one with the fewest instructions! Various classes of instructions require different times to complete their execution. It may require a very detailed study in order to find a sequence of instructions that accomplishes an objective in a minimum amount of time!) In most practical applications, however, the actual sequence of instructions selected is not at all critical. Select or create a method you like and try it out!

Diagram *Memory Map of Clearing Operation.*



A Practical Memory Clearing Application

Have you ever developed a BASIC program that used a temporary variable array? That is, an array that you needed to continuously clear out in between operations with other arrays? If so, you know that you had to create a special program loop that would initialize all the elements in that specific temporary array. You could not use a BASIC command such as CLEAR as that would wipe out *all* of the other variables and array elements used in the program, something that we assume for this example, would not be desired.

Of course, there is nothing wrong with creating a BASIC program loop to clear out the elements in an array. It is just that, if the number of elements in the array is large, the process can take some time. If the clearing operation has to be done frequently, the amount of time devoted to this one aspect can become quite exasperating.

However, as a ML programmer, it is possible to devise a scheme to clear out the elements of an array in the proverbial "blink of an eye." Let us see how this could be done.

First, it is necessary to know a few facts about the operation of the BASIC interpreter provided in the PC-1500 (and Radio Shack PC-2). As you may be aware, the interpreter program stored on ROM organizes RAM memory in a specific fashion to serve its purposes. Thus, the lower range of available user memory is assigned for use by the PC's "softkeys" as the REServe memory area. Immediately above this area (in terms of memory addresses) is where user program statements for a BASIC program (or multiple programs) are stored. Finally, the "top" or highest address value locations in RAM are used for the storage of user-defined arrays and variables.

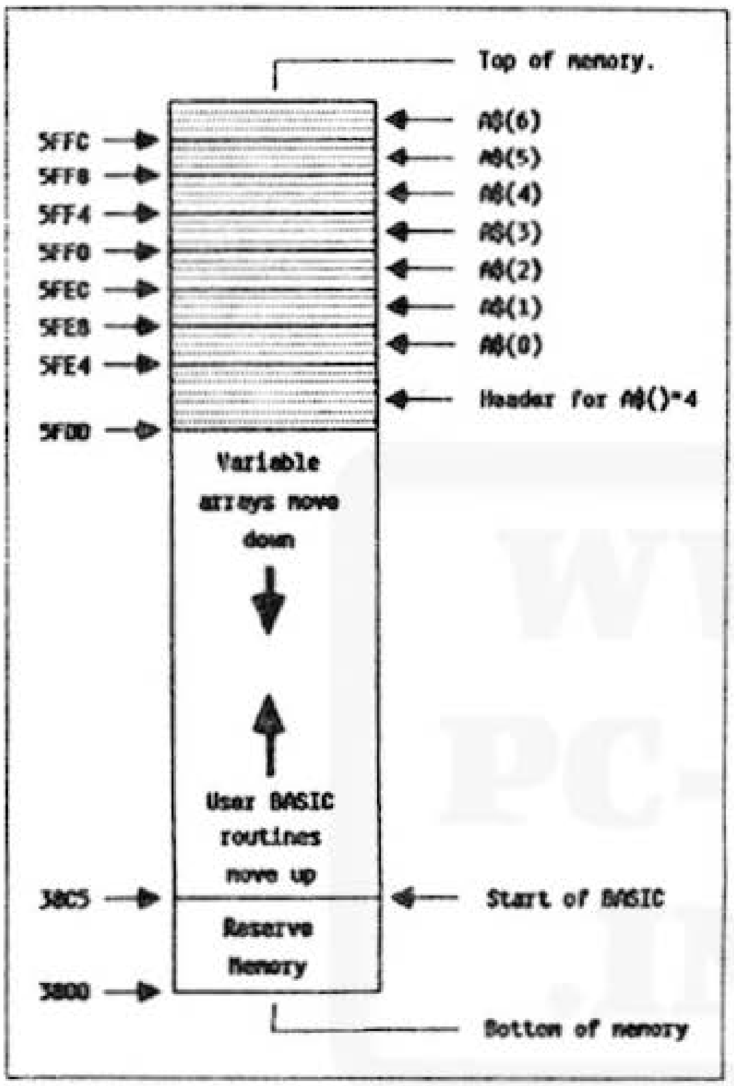
When the pocket computer is first turned on, the ROM program determines the bottom and top addresses of user RAM. The "page" (high order 8 bits) value of these locations are stored in the system RAM at addresses &7863 and &7864 respectively. This procedure is necessary as the range of user RAM can vary depending on which RAM expansion module (such as the CE-151, CE-155 or CE-161), if any, is installed in the PC. Thus, for example, if an 8K module was installed, location &7863 would contain &38 (which is the page portion of the address &3800 where RAM memory would begin). Location &7864 would contain &60. This is the page portion of the address &6000 which is one more than the top user RAM address (&5FFF) that is available in such a system.

(Knowing this information makes it possible to design a procedure that will automatically take account of the amount of memory in a user's system when it is used.)

When a BASIC programmer wants to create a variable array, it is necessary to issue a DIMension statement. This statement essentially blocks out space in user RAM for storage of the array elements. For purposes of illustration let us assume that the DIMension statement is the first statement contained in the user's BASIC program. Let us further assume that it is expressed as follows: DIM A$(6)*4. This means that a string array named A$() is being specified, that there are a total of 7 array elements (numbered 0 – 6) being reserved and that each element is to have room for 4 characters.

When the BASIC interpreter encounters this DIMension statement it will do the following: determine the top of memory (by examining system RAM location &7864), reserve 28 bytes of memory immediately below this location for storage of the array elements (7 elements with 4 characters reserved for each one), immediately below this it will record the array "header" information, (using 7 bytes of storage for this purpose). A summary of this process is shown in the accompanying diagram.

Diagram *Memory Map for Clearing an Array.*



The reason why it is necessary to assume that the DIMension statement is the first statement in the program in this description, is because any other arrays (or user-created variables) will be stored "beneath" this initial array. Thus, locating other arrays can become complicated and such complications are not needed at this point in the ML programming educational process. Right?

It will be worth knowing, so that you might customize such a routine to your own specific purposes, the following additional information about array elements: (1) The number of characters reserved for each element of a string array is precisely the number specified after the asterisk in the DIMension statement. If an asterisk is not used to specify such a value (which is limited to the range 1 to 80), then a value of 16 is assumed by the BASIC interpreter. (2) The number of characters reserved for each element of a numeric array is *always* eight! This is because all numeric values stored in such array elements are assumed to be in floating-point BCD format.

An Array-Clearing ML Routine

If we assume an array size of seven elements (numbered 0 through 6), with an element length of 4 characters, then we can calculate that the space needed by the array elements is exactly 28 bytes. (Note that this excludes the 7 bytes needed by the array "header." This is just as well, however, as we do not want to erase the contents of the array header!)

By examining the contents of memory location &7864, we can locate the start of memory. Actually, this location yields the first page address that lacks RAM, so user memory really begins on the highest possible address (&FF) on the next

Program *Routine for Clearing Memory that Terminates When Address Value Reached.*

PG	LC	B1	B2	B3	B4	B5	LABELS	MNEMONICS	COMMENTS
00	00	48	78				CLRMM4	LDXH #78	Set up register X to point to 16-bit address &78C0.
00	02	4A	C0					LDXL #C0	Set up register X to point to 16-bit address &78C0.
00	04	58	79					LDYH #79	Set up register Y to point to 16-bit address &7900.
00	06	5A	00					LDYL #00	Set up register Y to point to 16-bit address &7900.
00	08	B5	00				CLRM4	LDA #00	Load register A with zero.
00	0A	41						STAI (X)	Stuff zero into memory, advance memory pointer.
00	0B	94						LDA YH	Put contents of YH into the accumulator.
00	0C	86						CPA XH	Compare (YH-XH) to see if pointer page values are same.
00	0D	99	07					RBNZ CLRM4	Loop back if page values are not the same.
00	0F	14						LDA YL	Put contents of YL into the accumulator.
00	10	06						CPA XL	Compare (YL-XL) to see if pointer values are same.
00	11	99	0B					RBNZ CLRM4	Loop back if page values are not the same.

Program *Routine for Clearing Array Elements.*

PG	LC	B1	B2	B3	B4	B5	LABELS	MNEMONICS	COMMENTS
71	50	58	78				CARRAY	LDYH #78	Set up register Y to point to 16-bit address &7864.
71	52	5A	64					LDYL #64	This is where top of memory value stored by ROM routines.
71	54	15						LDA (Y)	Fetch top of memory value into accumulator.
71	55	DF						DEA	Decrement page value by one to point to next lower page.
71	56	08						STA XH	Store top of memory (adjusted) page value in register XH.
71	57	4A	FF					LDXL #FF	Store low portion of top of memory address in XL.
71	59	6A	1C				CARRA1	LDUL #1C	Set up counter (28 decimal here) in register UL.
71	5B	B5	00					LDA #00	Load the accumulator with zero.
71	5D	43					CARRA2	STAD (X)	Stuff accumulator into memory, then decrement pointer.
71	5E	62						DEUL	Decrement the counter value in UL (UL=UL-1).
71	5F	99	04					RBNZ CARRA2	If counter not zero, jump back to stuff another byte.
71	61	9A						RTS	Else, exit back to caller when counter equals zero.

lower page. Thus, if location &7864 contains the value &60 (indicating page &60 in the address &6000), the last location in RAM is at location &FF in the next lower page (&5F) or at address &5FFF. (This is precisely what would be encountered if a PC-1500 had an 8K RAM module such as the CE-155 installed.)

From there it is a simple matter to set up pointers and a byte counter within the CPU in order to erase the desired block of memory. However, now it will be appropriate to decrement the memory pointer as locations are cleared (instead of incrementing it as was done in previous routines). See the accompanying listing for the actual series of instructions that can accomplish this objective.

Check It Out With A Hybrid Program

You can test the operation of the array clearing routine by combining it with a BASIC program. The BASIC portion of the program can be used to DIMension the array, load the machine language routine into memory using POKE directives (since it is fairly short), and initialize the array with a set of known values.

Next, the initial values placed into the array can be displayed for checking purposes. The array-clearing ML routine can then be "called" using the CALL statement provided in BASIC. The contents of the array can then be displayed again to verify that the elements were properly cleared. This process may be repeated as long as desired for observational purposes. Here is the listing for such a *hybrid* program:

```
1000 "AA"DIM A$(6
     )*4
1010 GOSUB "CC"
1020 "BB"GOSUB "D
     D"
1030 GOSUB "EE"
1040 CALL &7150
1050 GOSUB "EE"
1060 GOTO "BB"
1090 END
1100 "CC"POKE &71
     50,&58,&78,&
     5A,&64,&15,&
     DF,8,&4A
1110 POKE &7158,&
     FF,&6A,&1C,&
     B5,0,&43,&62
     ,&99
1120 POKE &7160,4
     ,&9A
1130 RETURN
1200 "DD"FOR A=0
     TO 6
1210 A$(A)=STR$ (
     A)+STR$ (A)+
     STR$ (A)+
     STR$ (A)
1220 NEXT A
1230 RETURN
1500 "EE"WAIT 20
1510 FOR A=0TO 6
1520 PRINT A;"
     !";A$(A);"!
     ":PRINT " "
1530 NEXT A
1540 "FF"RETURN
```

(The nomenclature *hybrid* in this text refers to programs that combine BASIC and machine language methods within one general program, such as illustrated by this array-clearing example.)

Note that the ML routine is tucked into memory locations that are normally used for storing the string variables P$ and O$. Keep this in mind if you intend to meld this array-clearing capability into some other program!

INSIDE THE PC-1500

This is the sixth and concluding portion of the current series on the Sharp PC-1500. (Most of this information is also applicable to the Radio Shack PC-2.) *Norlin Rober, 407 North 1st Avenue, Marshalltown, IA 50158,* has been the provider of this extremely valuable information. We at *PCN* on behalf of all our readers, would like to extend our thanks to him for a most informative treatise.

This final section picks up from where we left off in Issue 35, with a continuation of information regarding printing program lines....

(b) Subroutine D2EA may be used to locate the line whose number has been loaded into Register U. (A data byte, which will be added to the return address if the specified line does not exist, must follow the instruction calling this subroutine.) After execution of subroutine D2EA, load XH and XL with the contents of 78A6 and 78A7; decrement X twice; and execute subroutine D2D0. The specified line will be in the Input Buffer, and the SEARCH pointer will point to that line.

(c) The program line that follows the one currently pointed to by SEARCH may be placed into the Input Buffer, with automatic update of the SEARCH pointer, by execution of subroutine D2B3.

(2) To print the line stored in the Input Buffer:

(a) Execute subroutine B73F to store the line size into 79F5. (If the CSIZE set is larger than 2, it will be set to 2.)

(b) Subroutine AE3A will perform a line feed with carriage return, then print the line contained in the Input Buffer. Follow by subroutine A769 (Motor off)

MISCELLANEOUS (MECHANICAL)

(1) Paper Feed Key Disable/Enable

Disable: Subroutine A306
Enable: Subroutine A30A

(2) To move paper (no update of coordinates, GRAPH mode):

With n a two-byte signed binary number, subroutine AA0E feeds the paper by n graphic units. (Negative n will feed in reverse.) Precede by loading XH with 7B, XL with 7F, Y with n, and UL with 01.

(3) Pen Up/Down Control:

Pen Up: Subroutine AB08
Pen Down: Subroutine AAFA
Pen Up/Down: Subroutine AAE3 sets pen up if 79E9 contains zero, down if 79E9 contains FF

(4) To move pen (no update of coordinates, GRAPH mode):

(a) Move right, n graphic units, with subroutine A28E. Precede by loading XL with n. Follow by subroutine A769 (Motor off)

(b) Move left, n graphic units, with subroutine A28B. Precede by loading XL with n. Follow by subroutine A769 (Motor off)

(c) Return pen to left end, without line feed, with subroutine A9D5. Follow by subroutine A769 (Motor off).

(5) To cycle pen to next color:

Execute subroutine A629. Follow by subroutine A769 (Motor off)

CASSETTE ROUTINES

TO SAVE A FILE ON CASSETTE

(1) Construct Header in Output Buffer area (7B60-7BAF)

(a) Execute subroutine BBD6 to form Lead-in, including file type code. Precede by loading A with file type code: 00=ML; 01=BASIC; 02=Reserve memory; 04=data file; others definable by user
(b) File name is optional; if to be included, its character codes must be stored into addresses 7B69-7B78 (maximum, 16 characters)
(c) Store beginning address of file being saved, into 7B82-7B83; store 1 less than the number of bytes in the file, into 7B84-7B85
(d) If a ML Program (file type 00) is being recorded, store the beginning execution address (or default value FFFF) into 7B86-87

(2) Record Header onto cassette

(a) Store value of Cassette Parameter into 7879: 00=RMT 0, 10=RMT 1
(b) Execute Call B0 (BCE8)

(3) Record file

(a) Load X with beginning address (obtainable from 7B82-83); load U with 1 less than number of bytes (obtainable from 7B84-85)
(b) Execute Call AA (BD3C); terminate operation with Call B4 (BBF5)

TO LOAD A FILE FROM CASSETTE

(1) Construct Header in Output Buffer area, 7B60-7BAF

(a) Load A with code for file type sought
(b) Execute subroutine BBD6
(c) Store (optional) file name into 7B69-78

(2) Find file on cassette

(a) Store value of Cassette Parameter into 7879: 80=RMT 0, 90=RMT 1
(b) Execute CALL B0 (BCE8). (A search will begin for a previously-recorded Header having the file type specified. If found, its file name is displayed and stored into 7B91-A0. If this name differs from the one specified, the search continues)
(c) If flag C is set, there was a checksum error; end with Call B4

(3) Load file into computer

(a) Load X with the address into which the file is to be loaded, and load U with 1 less than the number of bytes to be loaded. (The values for these parameters that were originally recorded are now obtainable, respectively, from 7BAA-AB and 7BAC-AD)
(b) Execute Call AA (BD3C). Following loading, Register X will contain the address immediately following the last address loaded
(c) If flag C is set, there was a checksum error; end with Call B4

(4) Terminate operation of the recorder with Call B4 (BBF5)

CONTROL OF RELAYS

Subroutine BF28 may be used to operate the remote control relays. The effect of this subroutine is determined by the contents of A.

A=03:	Close relay, REM 0
A=05:	Open relay, REM 0
A=09:	Close relay, REM 1
A=11:	Open relay, REM 1

RECORD, LOAD, OR VERIFY BLOCK OF DATA

Register X must be loaded with the beginning address of the block of memory involved, and Register U must contain 1 less than the number of bytes in that block. Then Call AA (BD3C) will perform the operation specified by the contents of the Cassette Parameter (7879), as follows:

When 7879 contains 00, the block of data is recorded onto tape.

When 7879 contains 80, the data read from tape is transferred to memory. If a checksum error occurs, there will be a return, with flag C set.

When 7879 contains 40, data read from tape is compared to the data in the block of memory, without any transfer of data. If a discrepancy is found, there is a return, with flags C and V set; if a checksum error occurs, a return with flags C and H set.

In each of the above cases, Call B4 (BBF5) may be used to terminate the operation; it will open the relay, shutting off the recorder.

RECORD OR LOAD A SINGLE BYTE

Subroutine BDCC will record the byte contained in the accumulator.

Call A4 (BDF0) will read one byte from tape, and load it into the accumulator. This routine also checks whether the BREAK key has been used; if it has, flag C is set.

1984 Index – Ordered By Type of Article

Article	Author	Issue	Page
Announcement: "The Commuter" Portable	PCN, Staff	31	1
Announcement: Casio FX-750P	PCN, Staff	35	1
Announcement: Casio PB-700	PCN, Staff	33	1
Announcement: Flight Computer Uses PC-1500	PCN, Staff	32	8
Announcement: Graphics Package for PC-2	PCN, Staff	32	8
Announcement: Hewlett-Packard HP-110 Portable	PCN, Staff	34	1
Announcement: Hewlett-Packard HP-71B	PCN, Staff	32	1
Announcement: Sharp PC-1350	PCN, Staff	36	1
Commentary: Watch Pocket	PCN, Staff	35	15
Commentary: Watch Pocket	PCN, Staff	36	16
Program: Addressing Pixels (PC-1500/PC-2)	Pollet, Patrick L.	33	9
Program: Alarm Clock (PC-1250/51/60/61 & PC-1211/PC-1)	Auersbacher, Emerich	36	3
Program: Alternate Character Sets (PC-1500/PC-2)	Rober, Norlin	35	10
Program: Appointment Calendar (PC-1500/PC-2)	Eichman, Steve	32	4
Program: Binary Magic (PC-1500/PC-2)	Gibson, John R.	33	11
Program: Formatted Mortgage (PC-1500/PC-2)	Hartmann, Peter V.	32	6
Program: Improved Vertical Lister (PC-1500/PC-2)	Rober, Norlin	34	8
Program: Isometric Drawings (PC-1500/PC-2)	Chrystie, Richard H.	34	10
Program: Linear Systems (PC-1500/PC-2)	Rober, Norlin	34	8
Program: Little Editor (PC-1500/PC-2)	Jackson, H. David	31	5
Program: Perspective Plot (PC-1500/PC-2)	Rober, Norlin	35	7
Program: Pocket Wordpro (PC-1500/PC-2)	Rober, Norlin	35	4
Program: Schedule Plotter (PC-1500/PC-2)	Chrystie, Richard H.	32	2
Program: Shell Game (PC-1500/PC-2)	Gibson, John R.	33	10
Program: Weather Forecaster (HP-71B)	PCN, Staff (Conversion)	36	5
Review: Casio PB-700	Woodhead, Jeffrey	36	2
Review: CE-501A, CE-501B, CE-505A Sharp ROM Modules	Sutliff, Robert	33	5
Review: Ce-502A, -502B, -504A, -507A Sharp ROM Modules	Sutliff, Robert	34	3
Review: Sharp PC-1261	PCN, Staff	35	2
Review: The Hewlett-Packard HP-71B	PCN, Staff	33	3
Review: The Sharp PC-5000 Portable	PCN, Staff	31	2
Technical Info: Clearing Memory and Arrays (PC-1500/PC-2)	PCN, Staff	36	7
Technical Info: Inside the PC-1500 (Part 1)	Rober, Norlin	31	11
Technical Info: Inside the PC-1500 (Part 2)	Rober, Norlin	32	9
Technical Info: Inside the PC-1500 (Part 3)	Rober, Norlin	33	12
Technical Info: Inside the PC-1500 (Part 4)	Rober, Norlin	34	12
Technical Info: Inside the PC-1500 (Part 5)	Rober, Norlin	35	12
Technical Info: Inside the PC-1500 (Part 6)	Rober, Norlin	36	12
Technical Info: Updates on Inside the PC-1500	Rober, Norlin	36	15

NORLIN UPDATES

Norlin Rober has noted a few errors in the ongoing *Inside the PC-1500* series that has been appearing in 1984 issues of *PCN*. Please make the following corrections:

Issue 31 – page 12 – line 785D should read: If 80, Katakana displayed; 00, displayed and . . . etc.

Addresses 786C to 786F *are* used by ROM, to temporarily save the contents of 787C-7F while the Display Buffer is temporarily saved in the String and Output Buffers.

Issue 31 – page 13 – 788E should read: TRACE parameter: 00, **ending** execution of **previous** line; 01, starting execution of **new line**;... etc.

Issue 31 – page 14 – location 79DA: . . . is obtained from 79DB – DC.

Issue 32 – page 11: In the middle of the page, the Output Buffer addresses for storage of tape header information are incorrectly given as beginning with 7A. . . the 7A should be replaced by **7B**.

Issue 32 – page 15: Under DECIMAL-TO-BINARY CONVERSION, the memory address of CALL D0 is incorrectly stated as D0F9; it should be **D5F9**.

Note too, that system RAM addresses that are "**unused**" by the PC-1500 actually are used by the CE-158. These are **7850 to 7858 and 79FA to 79FE**.

Norlin also notes that there is a mistake in his article in Issue 34 on page 8. The example given is not correct and its use can cause readers to think that the program is not operating correctly. The example printed should be changed so that the constant term in the third equation should be 3 (not 1) and the constant in the fourth equation should be 4 (not 3). Additionally, when used with an *old* PC-1500 (having early versions of ROM), the list of coefficients will not be LPRINTed correctly. To remedy this situation, change the last part of line 32 to read:

```
LPRINT "C  ";A(I,N+1): . . .
```

instead of:

```
LPRINT "C  ";A(I,J): . . .
```

Finally, Norlin has a few general tips to pass on the PC-1500 users:

1. A BASIC program inadvertently cleared by NEW can be recovered as follows:

(1) POKE &7750, 204, 101, 202, 105, 73, 0, 69, 221, 153, 4, 70, 202, 103, 154

(2) CALL &7750

Note: If the number of the first line was larger than 255, it will be changed by this procedure. It may be corrected with the usual editing procedures. You can use the above routine, in some cases, to recover a program that has been lost by a crash. Key ALL RESET and execute NEW 0 *before* using this routine for crash recovery!

2. It is possible to make a BASIC program halt with automatic display of the RESERVE template (for whichever of groups I, II or III is currently set). This technique adds a nice touch to a menu-driven program. The program instructions for doing it are simply: POKE &7880,0:CALL &CB8B

3. Here is a way to clear RAM completely. Every byte of RAM will be cleared to zero, including system RAM, after which the power-up routine will automatically be executed.

(1) POKE &4000, 40, 36, 8, 10, 170, 255, 253, 186, 211, 197

(2) CALL &4000

(3) Key CL and execute NEW 0

Finally, although the instruction manual does not mention it, the BASIC statement ON ERROR GOTO 0 will cancel an ON ERROR directive!

Available Only by Prepaid Subscription for a Calendar Year Period (January - December). You are sent back issues for the calendar year to which you subscribe, at the time you enroll.

__ I am interested. Please send more information. I have the following Model Pocket Computer: ______________

__ Enroll me as a 1985 Subscriber (Issue numbers 37-44). $24.00 in U.S. (U.S. $30.00 to Canada/Mexico. Elsewhere U.S. $40.00 payable in U.S. funds against a U.S. bank.)

__ Enroll me as a 1984-85 Subscriber (Issue numbers 31-44). $42.00 in U.S. (U.S. $51.00 to Canada/Mexico. Elsewhere U.S. $70.00 payable in U.S. funds against a U.S. bank.)

__ Enroll me as a 1983-85 Subscriber (Issue numbers 21-44). $78.00 in U.S. (U.S. $93.00 to Canada/Mexico. Elsewhere U.S. $120.00 payable in U.S. funds against a U.S. bank.)

__ Enroll me as a 1982-85 Subscriber (Issue numbers 11-44). $102.00 in U.S. ($125.00 to Canada/Mexico. Elsewhere U.S. $160.00 payable in U.S. funds against a U.S. bank.)

__ Check here if paying by MasterCard or VISA. Please give credit card information below.

Orders must be accompanied by payment in full. All checks must be magnetically encoded, payable in U.S. funds and drawn against a U.S. bank.

Name: ______________________

Addr: ______________________

City: ____________ State: ____ Zip: ______

MC/VISA #: ______________________

Signature: ____________ Exp. Date: ______

mail this order form to:
POCKET COMPUTER NEWSLETTER
P.O. BOX 232, SEYMOUR, CT 06483

POCKET COMPUTER NEWSLETTER
P.O. Box 232, Seymour, CT 06483

FROM THE WATCH POCKET

As we close-out our fourth year of publishing this unique newsletter, it is interesting to note that Sharp Electronics continues to spearhead the effort to produce good quality, low cost PCs. Of course, many of the other pocket computer pioneers are still hanging in there. Casio continues to put up a good fight. HP is doing OK. Even Panasonic continues to push its HHC, though primarily to specialized markets.

There are probably well over a million PCs in use in the United States today. That is only about one for every 250 persons. It took about 20 years for calculators to reach their present ubiquitous status. Chances are it won't take as long for PCs to reach the same level of distribution. Based on that projection, the growth of the industry should be steep during the coming years. *PCN* looks forward to covering this advancing, exciting field.

By the way, during 1985 we will increase the frequency of publication to eight times (instead of the current six issues-per-year). However, rising postal and publishing costs dictate that we revert back to a smaller type style and our previous 8-page format. The combination of steps being taken will yield continued quality coverage of the pocket computing field in a timely manner. We all thank you for your continued support and extend our best wishes for pocket computing happiness during 1985, our fifth straight year of publication!

PAYROLL DEDUCTIONS PROGRAM FOR HP-71B

This program is a customization of the Payroll program (described for other models of PCs in this issue) specifically for the HP-71B. [Refer to other articles in this issue for additional background information concerning this program.]

The program takes advantage of the 71B's advanced capability in handling inputs. For instance, the ability to "cue" a default input value. Using this capability means that default values do not have to be set up by using a separate "let" statement. Thus, for example, in line 20 of the program, instead of preceding the input statement with a statement setting G=0 (as a default value), that default value is made a part of the INPUT statement. You might want to customize this default value to your own application. For example, if you have a number of employees earning $350.00 gross per week, you could set the default value to that figure (instead of the 0 used in the program listing).

This HP version also utilizes labels in the tax lookup table routines (beginning at lines 600 and 800). Observe how line 60 obtains the label of the appropriate lookup table from the string M$ (which is inputted by the user in response to line 40).

If you are a newcomer to the use of the HP-71B, then you may want an explanation about the use of the statements A$=KEY$ in lines 20 and 100. As you may be aware, the -71B has a lot of features, including a "type-ahead" input buffer. This capability allows a user to start entering data before a prompt even appears on the screen. However, sometimes this advanced capability can trip a new programmer up! For instance, when using a delay of infinity (which is what you have when the line-scrolling parameter in a DELAY statement is 8 or more, as in this program), the use of the END LINE key causes that character to be stored in the type-ahead buffer. Whatever is in the type-ahead buffer is then used as the data for the next INPUT statement. That means, the next INPUT statement picks up an END LINE character as its initial character, thereby terminating the input operation and effectively causing the input to be a "null" character. That is usually a rather undesirable manner of operation. One way to get around this situation is to effectively empty out the type-ahead buffer prior to executing an INPUT statement. The use of the KEY$ statement does just that! Thus, the simple implied let statement used in lines 20 and 100 (A$=KEY$) uses a "dummy" variable (A$) as a means of dumping out the type-ahead buffer prior to using the INPUT statements in those lines. If you find this procedure difficult to understand, just try operating the program a few times with those statements removed and watch what happens!

Be sure to make use of the line editing capability of the HP-71B when building the lookup tables that start at lines 600 and 800. Once you have entered line 600, just use the FETCH command to get it back. Then, use the cursor controls to change the line number and modify the remainder of the line appropriately. You should find this method somewhat faster than having to type in each and every line in its entirety!

If you enter the program exactly as shown in the listing, a CAT should indicate that it is taking up 789 bytes.

For some fun, test the program by entering an amount of $5000.00 for hypothetical weekly wages, assume 4 dependents, and that you are married. You should get a FWT value of 1684.26, FICA witholdings of 352.50. With no other witholdings, the net pay would be 2963.24. If everything works out to these figures, then you can figure that at least the heart of the program is working O.K. (However, you should recheck all your table entries to make sure they are accurate before utilizing the program for serious purposes.)

FOR PC-1500 & PC-2 USERS

PC-1500 POTPOURRI

The two programs presented in the PC-1250/51/60/61 section of this issue, can, of course, be adapted to run on the PC-1500 with relative ease.

Rather than present a specific program for the PC-1500 in this issue, we are going to use this issue's column space to discuss several matters that are frequently raised by readers.

We often receive inquiries from program developers who want to safeguard their programs. "How?", they ask, "Can I lock up a program so that it cannot be listed, edited or saved on a cassette?" In other words, how can they "protect" a program so that they can sell it while preventing others from copying their work.

First of all, it is unlikely that any technique one might try to apply could be 100 percent effective. No matter what you do, the chances are that somebody else, having similar knowledge and skills, can undo your work. Thus, protection really becomes more of a game between the developer or "code maker" and the prospective "code breaker" or unauthorized duplicator. In the final analysis, it becomes a matter of wits and perseverance, knowledge and skill. For instance, virtually anyone who reads this article, is going to have knowledge about the techniques that are discussed and can use that information for or against copy protection. The information contained in *PCN* may be disseminated to perhaps five percent of all pocket computer users. Does that mean that 95% of PC users won't know about the techniques discussed here? Not necessarily. It is really impossible to determine how effective a technique might be since the "game" is really between those that "know" and those that do not! And, there is no way of knowing who has or who does not have the requisite knowledge.

One of the first things to decide when considering copy protection is just what is it you plan on protecting. Do you want to prevent duplication of the program and unauthorized distribution? Or, do you want to protect the algorithms or program methodologies? If your primary interest is the former, then you should consider the standard legal avenues such as copyright and trademark protection along with any technical anti-duplication techniques you might apply. If you are attempting to guard algorithms or methodologies, then your task may be quite different. Here, you may want to concentrate on making it difficult for someone else to figure out just exactly how you performed a particular operation. (Remember, while a copyright can protect a particular implementation of a program, it cannot stop someone else from using the same methodology or idea. A simple re-write of your program using different variable names, labels, text and some re-arrangement of the materials is all that is needed to legally circumvent the copyright laws.)

If you want to prevent outright duplication of your program, then you had better not plan on publishing it on magnetic tape. Anyone with a duplicating tape recorder unit can duplicate what is on a magnetic tape. If someone is serious about distributing your software (say, in a foreign country), all they need is one copy of your program recorded on tape and they are in business. Presto! They can be making copies in a matter of seconds. If someone is going to break the copyright laws in this manner (assuming your work has been copyrighted), it is certainly not hard to do from a technical viewpoint. Remember, we are talking here about tape-to-tape copying. The program never has to be loaded into a computer! No amount of protection within the program or computer is going to be able to prevent this simple outright duplication of the information contained on the tape cassette! By placing their own labels on the copied cassettes, program pirates can easily end up selling your program (particularly in a foreign country) with little chance of being detected.

OK, you say, you are going to distribute the program on a memory module (such as the CE-160) instead of tape cassette. Now what can you do? Well, the CE-160 programming system has an option that is said to prevent the listing, editing or saving of the program. Just select that option when you place a program into the module! Does that mean your program is now safe from duplication? Hardly.

Oh sure, it means the casual user cannot simply invoke a LLIST to view the program. Nor can that user invoke a CSAVE command to make a copy on a tape recorder. However, anyone familiar with machine language techniques (such as a reader of *PCN*) would be capable of accessing the information in the module. A simple machine language memory dump could be used to obtain the contents of the module. The information obtained from such a memory dump could be loaded into another PC without the protection. Once such a copy had been made, the program could be viewed, edited and duplicated. From there, mass duplication would be no more difficult than simply making copies on a magnetic tape cassette duplicator.

So what is the poor, defenseless, pocket computer programmer to do? Well, for one, stop being paranoid about duplication per se. Believe this: if your program is so valuable that people want to duplicate it for the sake of being able to generate some "underground" sales, they are going to find a way of doing it regardless of what you do. The only salvation is that you will probably make some money too, since you will have the lead in terms of promotion, supply, supporting materials, etc. Remember, however, that the outright duplicator (copier) who distributes your copyrighted material without your permission, is breaking the law. In other words, they are viewed as crooks. Thus, they take a certain amount of risk. And, if you can prove that they are duplicating your material, you may be able to put them out of business.

Frankly, you would probably be better off to assume that there is very little, from a practical viewpoint, you can do to stop someone bent on being a crook. It is better, perhaps, to be more concerned about slowing down someone who wants to legitimately compete with you by "re-writing" your program. Remember, a good many people do not want to risk being caught and branded as outright thieves. They might, however, view "competing" with you as quite another matter. They might even define "competing" as simply doing what you have done a little differently or a little better. All they need in order to provide this legal competition is to figure out how your program operates. In other words, they need to know how you did what you did.

There are things you can do to make this difficult on their part. One thing is to put key sections of your program in machine language. Once placed in the PC as plain object code, your competition is going to have to spend a lot of time figuring out what you did. First, they have to find out where you stored the key sections in memory. Then, they will need to disassemble your code and study it in detail. This process could take months. It is subject to various kinds of interpretation and error. (You might encourage misinterpretation by deliberately inserting nonsense sections in you code.) It may involve so much work that the perpetrators may find it easier to simply design their own version of your program from scratch. Or, they may decide it is not worth competing with you!

But, will it absolutely protect you from having your program copied? Of course not! Anybody with the same skills and training as you (for instance, in machine language) will still be capable of eventually figuring things out if that is their ultimate goal. The point now is, if they are so skilled and cunning, why would they bother? Chances are they would rather be devoting their time and energy to producing their own original creations.

Notice that protection of a program (preventing it from being copied or duplicated) is a different matter than securing an individual PC from unauthorized use. The former problem assumes that the program must be distributed by some practical means (tape or module). The latter assumes that the program is installed in the PC and that the use of that particular PC (and hence the program(s) it contains) is what is being secured. Several people have submitted programs to *PCN* that are claimed to provide protection along these lines for the PC-1500. Perhaps, if reader interest warrants, we can provide a sample of this type of security program in a future issue?

Code Breaking

A number of readers have asked for information on how to approach deciphering the machine code for a computer. That is, how did *PCN's* authors break the code for the LH-5801 CPU? We suspect that much of the recent expressions of interest in this matter have something to do with people wanting to work on discovering the machine language of some of the new PCs!

Whatever the reason for the interest, here are a few comments on the subject:

First, work on mapping memory. Determine which sections are devoted to RAM and ROM. Perform experiments (using BASIC or whatever language is available on the machine) to further define how RAM is used. That is, what addresses are used for program storage, where variables are stored, what parts of RAM appear to be used as "system" resources. This type of information should be arranged in orderly fashion, such as by memory address value.

Next, print out a dump of ROM memory. Preferably, each line of the dump should show raw machine code (in whatever number base you prefer) and the character produced by ASCII values. Using this technique can help you pick out various kinds of lookup and conversion tables within the ROM.

Then, obtain a histogram on the contents of ROM. That is, for each possible value (0 - 255 in an 8-bit machine), find out how many times the value occurs throughout ROM. It is fairly easy to do this using a BASIC program that counts the number of occurrences for each possible value and stores this in a 256-element array.

From this point on, start analyzing the data while calling on your own personal knowledge. The more experience you have in terms of computer science or mathematics, the higher the chances for immediate success. Use any data that has relevance for you. For instance, when Norlin Rober first started working on the PC-1500 ROM, he (being a skilled mathematician) was quickly able to spot values used to make mathematical conversions and tables used to calculate mathematical functions within the ROM.

Start making assumptions about the ROM code and playing hunches. For instance, in a large machine language program such as a BASIC interpreter, chances are good that some of the most frequently used instructions will be "loads" or "stores," "jumps," "branches" or "calls" to subroutines and "return" instructions. Use the histogram you made of ROM, the dump of memory, and your knowledge of the use of addresses in RAM to try and put some of your hypothesis together. For instance, you might assume that most call and jump instructions would be to locations within the ROM. This means that the call or jump opcode would be followed by address bytes having values within the range utilized by ROM. Take the values having the ten highest occurrences from the histogram of the ROM code that you compiled. Find occurrences of those values within ROM and see if any of them appear to be followed by reasonable address values. If so, mark the memory dump at those locations. When you get a handful of prospective jump or call locations, start examining the code in the vicinity of those addresses. See if you find the same code value immediately preceding each prospective jump or call address. Any luck? If so, you may have found the opcode for a subroutine "return" instruction!

On a machine such as the PC-1500, once a solid "guess-estimate" had been made on an opcode such as return, it was easy to confirm the finding. All that had to be done was POKE the suspected return opcode into a specific address in RAM and then attempt a CALL (using the BASIC statement) to that location. Finding that the PC immediately returned from the machine mode when the CALL was executed (from BASIC), regardless of where it was placed in RAM, quickly confirmed the tentative hypothesis. Once the thrill of confirming your first "find" has subsided, you can continue your investigations in a number of ways.

One way is to look for code in ROM that contains RAM addresses. Chances are good these involve instructions that are "pointing" to RAM. These may be "loads" or "stores." Try CALLing a ROM address that you have potentially identified as the entry point to a routine or subroutine. Ideally, the routine(s) you are trying at this point contain code that points to RAM locations. Observe what happens to these RAM locations. (Note, be prepared to do a lot of system resetting. You are likely to "bomb" the PC on almost every trial at this stage in the sleuthing process!) Do the RAM location(s) change? If so, you may be executing some "store" directives. Keep a record of your suspicions, correlate your findings with the ROM histogram and memory dump, try to confirm each hypothesis by repeating the test at different locations in ROM (that have similar opcode/data patterns).

It is great fun for some people. Time-consuming, that is for sure, but it can be exciting. How much work to break the code for a CPU such as the LH-5801? Perhaps as much as 300 to 400 hours to find just about all the directives. It is sort of like working on a giant puzzle. Bring on the PC-1350!

FOR PC-1350 USERS

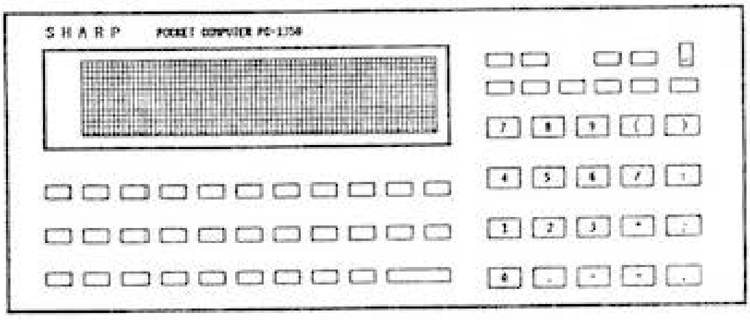
WELCOME ABOARD

Be sure to read the review of the PC-1350 elsewhere in this issue of *PCN*. This is the first issue having a column specifically devoted to this unit, so let's take a look at some of the most exciting news relating to this machine.

First of all, the BASIC language installed in the PC-1350 is virtually identical to that used in the PC-1260/61 models. Thus, the BASIC programs listed under that section (in *PCN*) should work without modification. So, presto! You can use the payroll and memory dump programs that are in this issue on your new PC-1350. Indeed, the compatability between the 1260 and 1350 is so good that you can load programs from tapes made with the 1260 directly into the 1350. (You cannot, however, go the other way. As the Sharp manual points out, the PC-1350 is "upwards" compatible. You can load from an earlier model into the new 1350. You cannot use tapes to go the other way.)

Programs designed for the operation on the PC-1250 (also the Radio Shack PC-3) and even for the original PC-1211 (also the Radio Shack PC-1) will work in the new 1350 with little or no alteration. (Unless you used programming tricks such as implied multiplication or left off closing parenthesis. In such cases, you will have to do some editing.)

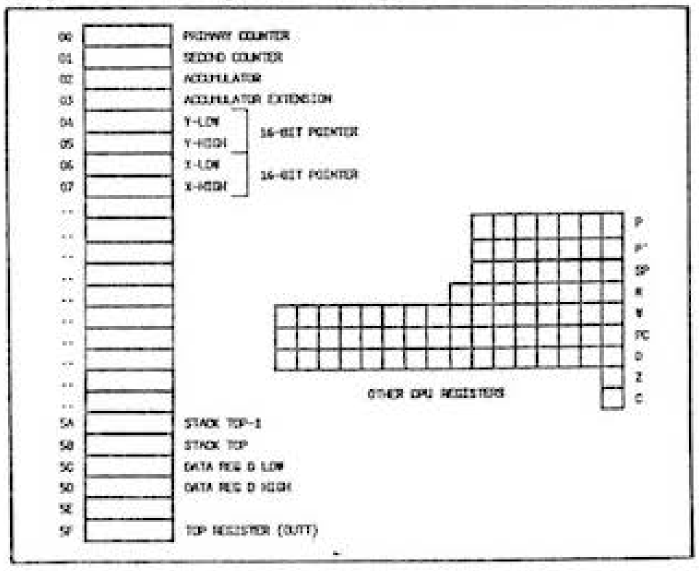
Now for the next hot piece of news. Preliminary investigation indicates that the PC-1350 utilizes the same CPU as that in the PC-1250 and PC-1260 series! And wouldn't you know it, a number of *PCN* fans have been hard at work busting the PC-1250 machine language code. In fact, next issue we will be publishing a comprehensive report on the work of a new CPU sleuth, *Rick Wenger*, who has pretty well decoded the instruction set for the CPU used in the 1250 (and consequently that of the 1260 and, it appears, the PC-1350).



If you have been doing any exploring on your own, an accompanying diagram summarizes the internal registers associated with the unit's CPU. You can use the memory dump program described for the 1250 to begin mapping out the 1350. It appears that there is some RAM in the 1350 beginning at hexadecimal address &6000. This should be a fun machine to explore. Imagine what kind of machine language programming tools you could tuck into one of those 16K RAM cards?

The PC-1350 is considerably easier to program from the keyboard than other Sharp models. This is primarily due to the re-arranged keyboard which eliminates having to use the shift key in order to input a lot of the punctuation marks used in BASIC programming. Symbols such as the comma, colon, semicolon, etc., have their own separate keys. Also, the 4-line display, capable of displaying 96 characters at a time, makes it a lot easier to review and edit programs. Editing is also aided by the fact that you can insert or delete characters using special keys for those operations. No more constant use of the extra shift key.

Diagram ***Probable Layout of CPU Registers in the PC-1350.***



POCKET COMPUTER NEWSLETTER

(c) Copyright 1985 POCKET COMPUTER NEWSLETTER

Issue 38 - February

REVIEW OF TRAMSOFT TOOLKIT

Recently *PCN* had the opportunity to review the operation of a device designed to assist PC-1500 owners. The unit is named the TRAMsoft Toolkit. It is currently being manufactured and distributed in Europe. This device proved interesting for several reasons, not the least of which is its ability to save and restore programs and/or data about 25 times faster than a standard PC-1500 CSAVE or CLOAD operation. What is more, this amazing increase in speed is accomplished through the use of a standard audio cassette recorder! (A Radio Shack model Minisette-9 was used in tests conducted by *PCN*.)

The TRAMsoft Toolkit consists of several electronic integrated chips and supporting components mounted inside a small shielded box, (measuring approximately 3 by 3 by 3/4 inches). A 60-pin male connector mounted on one end of the box is used to connect the unit directly to a PC-1500 or to the rear port of a CE-150 printer/cassette unit. There are two jacks on a side of the box, used to connect with an audio tape recorder.

One of the ICs inside the unit is a ROM containing software (apparently written in machine language) that controls overall operation of the unit. This software is used to "hook" onto the BASIC operating package in the PC-1500 and give it a number of new capabilities by providing new BASIC keywords.

For instance, there are four new keywords associated with using the enhanced magnetic tape storage system: FCHAIN, FLOAD, FSAVE and VERIFY. As you might surmise, FCHAIN permits you to chain in a new program segment using the faster tape technique, similar to what is done with a standard CHAIN statement. FLOAD and FSAVE invoke the faster tape system in the same fashion as CLOAD and CSAVE. These commands also provide the extensions that permit saving and loading machine language programs. The VERIFY command is similar to CLOAD?, but is intended verify data saved using the enhanced method. Furthermore, the enhanced tape system is able to verify machine language programs, something that is not possible using the standard CLOAD? option.

In tests conducted by *PCN*, using FSAVE to store a program on tape and FLOAD to recover it was many times faster than using CSAVE/CLOAD. For instance, a program that took about six minutes to load with Sharp's methodology, took barely 15 seconds with the TRAMsoft Toolkit. All-in-all, the system seems to be about 25 times faster. This can make a big difference in how you approach using your PC. While six minutes seems interminable (and hence frequently not worth bothering with!), 15 seconds to load a decent-sized program into memory is relatively easy to bear.

The TRAMsoft Toolkit would seem a pretty valuable device even if only considered for its enhanced program storage capability. But, the package contains a number of other features that will be of value to those who like to use their PC a lot. These other capabilities are referred to as "programming aids." In essence, what this system enables you to do is "partition" user memory into modules, much as what occurs when you "merge" programs. The difference is that you have full, instantaneous control over the "modularization" and "merge" process.

Thus, you may have in effect, up to 255 "modules" in memory at a time. Only one module is "active" at any given moment. (The active module is the one in which any "editing" operations will take place.) This manipulation of modules is accomplished through the following types of new "keywords" that have been hooked into the BASIC operating system.

APPEND -- closes an active program module and enables you to create ("open") another module.

CHANGE -- is a powerful editing function that enables you to "search and replace" over a given range of line numbers or for a specified number of instances.

DELETE -- gives the capability to remove a range of line numbers from the current "active" module.

ERASE -- allows a user to eliminate an entire module.

FIND -- searches for a text string or BASIC token.

KEEP -- provides the ability to save the loadable part of cassette input following a loading error (partial load).

LINK -- combines program modules.

PLIST -- lists the current "active" program module.

PLAST -- gives last line number in the active program module.

PROGRAM -- activates a designated module.

RENUMBER -- provides capability to renumber the active module over any range and by any increment

SPLIT -- allows user to split one module into two modules (opposite of the LINK command).

It should be noted that this "modularization" is essentially an "editing" feature. That is, you use it when in the PROgram mode. When in the RUN mode, the PC operates in its normal fashion and must be directed to execute the desired "module" by reference to an appropriate label. Thus, the system does not provide true multiple file capability (such as is found in Hewlett-Packard's HP-71B). However, it does go a long way towards making it easier to develop complex programs or customize a PC by loading (and editing) essentially independent program modules.

Anyone seriously developing programs for the PC-1500 (and, probably, the Radio Shack PC-2), would undoubtably appreciate this tool.

Alas, the TRAMsoft Toolkit is not yet available in the United States. However, if sufficient genuine interest was expressed, it might be possible to have the tool imported and distributed. (Pricing in Europe is apparently equivalent to the price of a PC-1500.) If you would like to see this type of device made available in the U.S., *PCN* suggests you drop a note expressing your interest to: *Chris Mallner, 150 Yantic Street - Apt. 159, Norwich, CT 06360.*

Another *Personal Information*(TM) product.

FOR PC-1500 & PC-2 USERS

PASSWORD PROTECTION

Last issue, this column discussed some of the ramifications of attempting to protect programs from unauthorized use or duplication. As indicated in that article, such protection is generally "breakable" by those that have knowledge of the system being utilized. However, they can be effective against the "casual" user or those that do not make a concerted effort to defeat the mechanism(s).

The program described here, submitted by *Eric Bowman, PEA 81, Exeter, NH 03833,* is claimed to be a "fairly effective password protection system." The protection program is also capable of disabling use of the printer by those not authorized. The program has been designed (as presented here) to reside in the machine language program area of a PC-1500A. However, the necessary changes for relocation of the program, so that it might be used in a standard PC-1500 (or Radio Shack PC-2) have been included in the article.

Operating instructions are as follows. After typing in the program, immediately execute a CALL &7D33. The computer will then prompt the user with "SET PASSWORD:". At this point, the user should decide on a password of five letters and type it in. As the fifth letter is entered, the password will be displayed, along with a question mark. If the password is the desired one, "Y" should be pressed to install the password and terminate the program. If the displayed password is not appropriate, responding with an "N" will restart the program, thus enabling the user to try again. Pressing BREAK at any time will terminate the program leaving the previous password in effect.

To actually use the password, execute a CALL &7C01. At this point the computer should turn off by itself. Upon power-up, the display will read: PC-1500A ONLINE:. Pressing any key, including BREAK, should cause PASSWORD: to be displayed. At this point, the user must enter the previously defined keyword. As the fifth letter of the password is entered, the computer will either turn itself off, indicating an invalid password, or it will display the query PRINT (Y/N)? The latter indicates that the user has access to the PC and now has the option of enabling or disabling the printer. If the user answers "Y" at this time, the printer will be available for normal use. Responding with a "N" will result in the PC simulating the low-battery condition if any attempt is made at using printer commands. (This yields an ERROR 78 message.) This state may be altered by either running the program again and answering "Y" to the printer prompt or by a POKE &79F1,0 directive.

(Note: during the password entry process, the LF key is disabled.)

Program Registers

The program uses various memory locations throughout memory

Program ***Password Protection.***

```
7C01 B5 40 AE 76    7C71 FD 88 B5 36    7CE1 50 58 78 5A    7D51 52 44 3A BE
7C05 4E 85 43 AE    7C75 AE 78 75 BE    7CE5 60 6A 0A F5    7D55 ED 3B 85 4E
7C09 76 4F EB 79    7C79 E2 43 83 5A    7CE9 88 03 F2 BA    7D59 AE 78 75 BE
7C0D F1 FF FD BE    7C7D FD C8 B5 2A    7CED 7C 01 F2 FD    7D5D E2 43 C3 42
7C11 E9 79 D5 00    7C81 BE ED 4D FD    7CF1 58 B5 0B FD    7D61 FD C8 BE ED
7C15 FD 58 B5 0A    7C85 8A FD 0A 41    7CF5 CA FD 6A 4A    7D65 4D FD 8A FD
7C19 FD CA FD 6A    7C89 A5 79 D5 DD    7CF9 0C BA 7D 09    7D69 0A 41 4E 05
7C1D 4A 10 BE 10    7C8D B7 05 88 07    7CFD 50 52 49 4E    7D6D 98 04 FD 88
7C21 50 43 2D 31    7C91 AE 79 D5 FD    7D01 54 20 28 59    7D71 9E 17 F2 68
7C25 35 30 30 41    7C95 88 9E 20 B5    7D05 2F 4E 29 3F    7D75 7F 6A 00 4A
7C29 20 4F 4E 4C    7C99 D0 AE 7A 04    7D09 BE ED 3B BE    7D79 05 BE ED 3B
7C2D 49 4E 45 3A    7C9D B5 76 AE 7A    7D0D E2 43 83 10    7D7D B5 24 AE 78
7C31 BE ED 3B 48    7CA1 05 B5 50 AE    7D11 B7 59 88 13    7D81 75 B5 3F BE
7C35 78 4A 60 58    7CA5 7A 06 B5 05    7D15 B5 FF AE 79    7D85 ED 57 BE E2
7C39 78 5A 50 6A    7CA9 AE 7A 07 B5    7D19 F1 E9 79 FF    7D89 43 C3 42 B7
7C3D 0A F5 88 03    7CAD D0 AE 7A 14    7D1D 00 FD 81 9A    7D8D 59 88 07 B7
7C41 B5 00 AE 79    7CB1 B5 79 AE 7A    7D21 FD E9 F0 0B    7D91 4E 98 11 BA
7C45 FF 48 6F 4A    7CB5 15 B5 FA AE    7D25 FD 9E 39 E9    7D95 7D 5C 48 7F
7C49 FF CA 65 CA    7CB9 7A 16 B5 05    7D29 79 F1 00 E9    7D99 4A 00 58 79
7C4D 67 CA 69 BE    7CBD AE 7A 17 B5    7D2D 79 FF 00 FD    7D9D 5A FA 6A 04
7C51 E3 3F F2 FD    7CC1 04 BE D0 F9    7D31 81 9A 48 7F    7DA1 F5 88 03 9A
7C55 58 B5 0B FD    7CC5 8B 17 48 78    7D35 4A 00 FD 98    7DA5 F2 48 7F 4A
7C59 CA FD 6A 4A    7CC9 4A 50 58 78    7D39 F2 FD 58 B5    7DA9 00 FD 98 BA
7C5D 09 BA 7C 6A    7CCD 5A 60 6A 0A    7D3D 0B FD CA FD    7DAD 7D 39 CC 65
7C61 50 41 53 53    7CD1 F5 88 03 BA    7D41 6A 4A 0D BA    7DB1 CA 69 49 00
7C65 57 4F 52 44    7CD5 7C EF FD E9    7D45 7D 54 53 45    7DB5 45 DD 99 04
7C69 3A BE ED 3B    7CD9 F0 08 FD 9E    7D49 54 20 50 41    7DB9 46 CA 67 9A
7C6D 48 76 4A 50    7CDD 66 48 78 4A    7D4D 53 53 57 4F
```

to store parameters. Most of these locations are, it is believed, used by the CE-158 interface. Thus, it is not advisable to attempt using this program when a CE-158 is connected without first making sure that the following locations will not be disturbed:

79D5	Counts the number of characters entered
7650-4	Stores the characters being typed
79FA-E	Location of actual password
7F00-4	Stores the characters being typed in the SET PASSWORD routine

Several characteristics of the program are worth mentioning. Once access has been gained, the computer is set to DEG, RUN, softkey mode I, and LOCK. If this is not desirable, everything except the setting of LOCK may be changed by replacing the first ten bytes of the program with NOPs. Note too, that the password is not in effect unless the machine is turned off by using CALL &7C01. (The OFF key functions normally.) You might want to place the CALL &7C01 directive under control of a softkey. Also, if the machine sits idle for more than seven minutes during the password entry process, it will automatically shut off. However, upon power-up, nothing will have changed and the user can continue entering a password.

Relocating the Program

To simplify the process, it is suggested that you relocate the program so that it starts at an address ending with 01. Thus, if you elected to start it at address 3901, the following changed would need to be made:

385F to 39
3905 to 39
39ED to 39
39FB to 3A
3A45 to 3A
3A95 to 3A
3AAD to 3A

The program uses &7F00-4 as a brief stack. If the user does not have a PC-1500A (or has another program stored there), this stack can be relocated to 7650-4 by making the following alterations:

7D34 (3A34) to 76
7D36 (3A36) to 50
7D75 (3A75) to 76
7D77 (3A77) to 50
7D98 (3A98) to 76
7D9A (3A9A) to 50

Of course, by storing the appropriate high and low bytes at these locations, the stack could be relocated anywhere in memory. However, 7650 seems a good place.

Memory Map

The main sections of the program are located as follows:

7C01	Start of password routine. This section turns the computer off, then asks for the password upon power-up.
7D33	Start of routine to set the password. Asks for the password, verifies it and stores it.
7DAF	"Renew" program originally contributed by Norlin Rober (*PCN* Issue 36, page 16).
79FA-FE	Location of password.

In the event that the password is ever forgotten, you can perform the following procedure. First, turn the machine on while pressing the "all reset" button (on the back of the PC) and holding the ON key. Press the clear key and then enter and execute the following brief machine language program. (It may be located at any convenient place in memory).

48 78 4A 50 58 78 5A 60 6A 0A F5 88 03 9A

This routine will set the BASIC parameters back to their original values and will allow you to re-use whatever BASIC program was in memory.

So there you have it! Why not give it a try. How difficult do you think it would be to break this protection scheme (assuming you didn't have the information given in this article)?

Available Only by Prepaid Subscription for a Calendar Year Period (January - December). You are sent back issues for the calendar year to which you subscribe, at the time you enroll.

__ Enroll me as a 1985 Subscriber (Issue numbers 37-44). $24.00 in U.S. (U.S. $30.00 to Canada/Mexico. Elsewhere U.S. $40.00 payable in U.S. funds against a U.S. bank.)

__ Enroll me as a 1984-85 Subscriber (Issue numbers 31-44). $42.00 in U.S. (U.S. $51.00 to Canada/Mexico. Elsewhere U.S. $70.00 payable in U.S. funds against a U.S. bank.)

__ Enroll me as a 1983-85 Subscriber (Issue numbers 21-44). $78.00 in U.S. (U.S. $93.00 to Canada/Mexico. Elsewhere U.S. $120.00 payable in U.S. funds against a U.S. bank.)

__ Enroll me as a 1982-85 Subscriber (Issue numbers 11-44). $102.00 in U.S. ($125.00 to Canada/Mexico. Elsewhere U.S. $160.00 payable in U.S. funds against a U.S. bank.)

__ Check here if paying by MasterCard or VISA. Please give credit card information below.

Orders must be accompanied by payment in full. All checks must be magnetically encoded, payable in U.S. funds and drawn against a U.S. bank.

Name: ____________________

Addr: ____________________

City: __________ State: ____ Zip: ______

MC/VISA #: ____________________

Signature: __________ Exp. Date: ______

mail this order form to:

POCKET COMPUTER NEWSLETTER
P.O. Box 232, Seymour, CT 06483

FOR PC-1350 USERS

TITLE

If you think this column is a little short this issue, go back and take a look at the PC-1250 column. See what it says there? The CPU used in the Sharp PC-1350 is the same one used in the 1250! That means the instruction set that has been so nicely decoded by *Rick Wenger* (and presented in this issue) for the 1250, also works on the 1350!

Want to verify that is indeed the case? Try loading the following short machine language routine into your PC-1350:

```
POKE &6800,&12,&5F,2,&20,&DB,&DF,&37
```

Then, place the following BASIC statement in memory:

```
100 "B" CALL &6800 : WAIT 1 : PRINT "*" : GOTO "B"
```

Executing the BASIC routine using DEF/B should result in your hearing a string of "beeps" coming from your PC and a series of asterisks appearing on the display. The beeps are being generated by the machine language routine.

Can you decode the five machine language instructions that make up this routine (using the information provided in this issue under the 1250 section)? Hint: the first one is LDP # 5F. Once you do this, you can try some experimenting on your own.

Hopefully, we will have some PC-1350 memory maps soon!

```
400 NEXT Z @ W=0 @ DISP "BEST FIT WAS #";U
410 ON U GOTO 180,230,280,330
420 DISP "ESTIMATE OF X" @ INPUT "Y= ","?";V @ ON U GOTO 440,450,460,470
430 DELAY 8,0 @ DISP "X, Y OR BOTH NEGATIVE" @ GOTO 10
440 L=(V-A)/B @ GOTO 630
450 L=(LOG(V)-LOG(A))/B @ GOTO 630
460 L=EXP((V-A)/B) @ GOTO 630
470 L=EXP((LOG(V)-LOG(A))/B) @ GOTO 630
480 INPUT "DEL. LAST PT (Y/N)?","N";Z$ @ IF Z$="Y" THEN 510
490 DISP "PT# ";N @ INPUT "X= ";L @ IF L<=0 THEN J=1
500 X=L @ INPUT "Y= ";Y @ GOTO 530
510 DISP "X=";X;"Y=";Y @ INPUT "OK TO DELETE (Y/N)? ","N";Z$
520 IF Z$="N" THEN 10
530 C=-1 @ IF Y<=0 THEN T=1
540 GOSUB 130 @ GOTO 10
550 DISP "ESTIMATE OF Y" @ INPUT "X= ";L @ ON U GOTO 560,570,580,590
560 V=B*L+A @ GOTO 630
570 V=A*EXP(B*L)-@ GOTO 630
580 V=B*LOG(L)+A @ GOTO 630
590 V=A*L^B @ GOTO 630
600 DELAY 8,0 @ DISP "A="; @ DISP USING 700;A @ DISP "B=";
610 DISP USING 700;B @ DISP "R="; @ DISP USING 700;R @ C=R*R
620 DISP "R**2="; @ DISP USING 700;C @ GOTO 10
630 DELAY 8,0 @ DISP "X=";L;"Y=";V @ GOTO 10
640 DISP "X=";X;"Y=";Y @ RETURN
650 U=Z @ V=C @ RETURN
700 IMAGE 5D.DDD
```

the program returns to the main menu.

If you want to see how other curves fit, by examining equation terms and coefficient, select menu item number 4. This will bring up a secondary menu that enables you to select any of the four curves: linear, exponential, power or logarithmic. Select the curve desired. The terms and coefficients will then be presented in response to presses of the END LINE key.

Items 5 and 6 of the main menu permit you to obtain projected values of x and y based on the best fit. Respond to the prompts to obtain the desired information.

To test the program, try entering the following data points: PT# 1 X=1 Y=1, PT# 2 X=3 Y=3, PT# 3 X=5 Y=5. Remember, enter PT# 0 to end the data entering process. After returning to the main menu, select menu item number 3. Observe that the program displays four sets of data and then announces that "BEST FIT WAS # 1". It then displays the type of curve (Linear) and its equation. Finally, you should see that it reports the terms and coefficients as: A= 0.000, B= 1.000, R= 1.000 and R**R= 1.000. When back in the main menu, try items 5 and 6. Positive values for either axis (such as X= 9) should yield an identical figure for the other (i.e., Y= 9).

Note: when selecting from the flashing menu, hold the desired item number key down for about one-half a second.

The HP-71B is especially suited for scientists and engineers. This program can help people in those disciplines utilize some of its special capabilities.

FOR PC-1500 & PC-2 USERS

SPEED COMPARISONS -- PC-1500 STILL TOPS

Norlin Rober, 407 North 1st Avenue, Marshalltown, IA 50158, has run some comparisons amongst the Sharp and Radio Shack PCs. They show that the Sharp PC-1500, on the whole, outperforms all other compared models in terms of overall typical operating speeds. Here is his complete report.

The accompanying table shows the execution times (in milliseconds) for various types of operations performed on Sharp and Radio Shack models of pocket computers. To make the comparisons meaningful, the same input arguments for a particular operation were used on all computers.

Here is how the tests were conducted. First the simple line: 10 FOR X=1 TO 10000: NEXT X was executed. In the PC-1350 this took 69 seconds. Then, to check division for example, the statement A=B/C was inserted into that line. On the PC-1350, this was timed at 447 seconds. This indicated that

Table *Comparisons of PC Execution Times.*

Statement Executed	PC-1211 PC-1	PC-1250 PC-3	PC-1260 PC-1261	PC-1350	PC-1500 PC-2
A=B	75	15	12.6	12.0	6.4
A=B+C	105	27	22.4	21.4	11.1
A=B*C	145	47	33.0	31.8	21.4
A=B/C	169	58	39.0	37.8	30.6
A=√B	171	67	43.4	42.5	39.8
A=B^C	630	408	252.3	257.6	205.9
Loop Cycle	192	42	7.5	6.9	14.0
GOTO (next line)	80	20	9.0	8.5	8.3
GOTO (later line)	2420	161	26.1	25.6	16.0

NOTE: The "GOTO (later line)" required searching past fifty 25-byte lines.

378 seconds was required for 10,000 division operations. Of course, division time will vary depending on what numbers are being divided.

Some observations that are of interest: 1. The newest computers such as the PC-1260 and PC-1350, surpassed the PC-1500 in executing loops, but they still *lagged behind the PC-1500 in all other tests!* 2. The PC-1350 outdid the PC-1260 by a small margin, except for one interesting case, when it handled the statement A=B^C slightly faster.

The "Benchmark" program listed in *PCN* Issue 26, page 4, was also tried. The results were as follows: PC-1211 = 693 seconds, PC-1250 = 218 seconds, PC-1260 = 119 seconds, PC-1350 = 118 seconds and the PC-1500 = 107 seconds. Again, the PC-1500 came in the overall winner.

In another comparison between the PC-1500 and PC-1350, using the same program in each PC to solve the same system of 8 linear equations, the PC-1500 took 22 seconds, the PC-1350 consumed 35 seconds.

Cassette operations, however, show a marked improvement in the PC-1260 and PC-1350. The accompanying table indicates the measurements made during tape operations (times are in seconds).

Table *Comparisons of PC Tape Operation Times.*

	PC-1211	PC-1250	PC-1260	PC-1350	PC-1500
Time for "Header"	6	8	8	8	11.7
Time, per kilobyte	93	44	27	27	72.5

A ROUTINE TO VERIFY MACHINE LANGUAGE SAVES

This program, CLOAD M?, is to machine language programming what CLOAD? is to BASIC -- a means of verifying that a program has been saved correctly on tape.

The program, supplied in machine language itself, is fully relocatable. Once you have keyed it into memory, be sure to save it on tape before using it.

To use the program, proceed as follows. First, save the machine language program you want to verify using the standard CLOAD M command. Rewind the tape to the start of that program. Place the remote switch on and press the "play" button on the recorder. Now, call the starting address of the CLOAD M? program. Respond to the TITLE: prompt with the name of the machine language program that you just saved. Press the ENTER key. The tape recorder should activate.

When the program has been read by the program, one of three displays will appear. VERIFICATION COMPLETE is shown if there were no discrepancies between the tape and the original code. DISCREPANCY will appear if there was any disagreement. In this case, you should try re-saving the code. The message CHECKSUM will appear if there was a read error. This indicates that the tape is bad. Re-record the code with a new tape.

This CLOAD M? program is supplied courtesy of: *Eric Bowman, PEA Box 81, Exeter, NH 03833.*

Program ***CLOAD M? for the PC-1500/PC-1500A.***

```
7DBD F2 BE D0 2B        7E49 7B AB 0A A5
7DC1 FD 58 B5 08        7E4D 7B AC 28 A5
7DC5 FD CA 6A 05        7E51 7B AD 2A CD
7DC9 BE 06 54 49        7E55 AA 83 2C CD
7DCD 54 4C 45 3A        7E59 B4 F2 FD 58
7DD1 58 7B 5A B0        7E5D B5 0A FD CA
7DD5 F5 88 03 B5        7E61 FD 6A 4A 15
7DD9 40 AE 78 B0        7E65 BE 15 56 45
7DDD BE E8 CA BE        7E69 52 49 46 49
7DE1 E2 43 C3 42        7E6D 43 41 54 49
7DE5 B7 18 9B 2C        7E71 4F 4E 20 43
7DE9 EB 7B 0E 40        7E75 4F 4D 50 4C
7DED B7 08 89 07        7E79 45 54 45 BE
7DF1 5E B6 9B 15        7E7D ED 3B BE D0
7DF5 56 9E 1B B7        7E81 2B CD 46 87
7DF9 0C 89 08 B5        7E85 23 F2 BE D0
7DFD 0D 17 9B 21        7E89 2B CD B4 F2
7E01 54 9E 27 B7        7E8D FD 58 B5 0A
7E05 0D 8B 1D E9        7E91 FD CA FD 6A
7E09 7B 0E 8F B7        7E95 4A 0B BE 0B
7E0D 1C 89 05 BE        7E99 44 49 53 43
7E11 CD E6 9E 3B        7E9D 52 45 50 41
7E15 B7 1D 89 05        7EA1 4E 43 59 BE
7E19 BE CE 38 9E        7EA5 ED 3B CD 46
7E1D 41 B7 20 91        7EA9 BE D0 2B CD
7E21 42 51 9E 48        7EAD B4 F2 FD 58
7E25 B5 00 BE 8B        7EB1 B5 0A FD CA
7E29 D6 48 7B 4A        7EB5 FD 6A 4A 0E
7E2D B6 58 7B 5A        7EB9 BE 0E 43 48
7E31 69 05 B7 0D        7EBD 45 43 4B 53
7E35 88 04 51 44        7EC1 55 4D 20 45
7E39 9E 09 B5 C0        7EC5 52 52 4F 52
7E3D AE 78 79 CD        7EC9 BE ED 3B BE
7E41 B0 83 40 A5        7ECD E2 43 CD 46
7E45 7B AA 0B A5
```

FOR PC-1350 USERS

USING POKES ON THE PC-1350

The capability of the PC-1350 may be extended by the use of various POKES. A few of these techniques are discussed here.

Memory Reallocation

In an unexpanded PC-1350, BASIC normally starts at address &6030. This address is stored in a START OF BASIC pointer. This pointer is located at &6F01-02. Note that &6F01 contains &30, the low byte of this starting address. &6F02 contains &60, the high byte of this starting address.

BASIC can be relocated by changing the contents of this pointer. For example, execution of POKE &6F01,&30,&64, followed by execution of NEW (in the PRO mode) will set the START OF BASIC to &6430. Now any BASIC program, whether entered by CLOAD or from the keyboard, will be stored beginning at &6430. The area from 6030 to 642F is then available for machine language. BASIC will not interfere with it.

Execution of CALL 0 will restore the START OF BASIC pointer to its normal value, clearing programs and variables.

Those familiar with the PC-1500 and PC-2 will note that the two operations described above are equivalent to using NEW <address> and NEW 0.

Correction of the Printer Bug

If printing with the CE-126P is interrupted by the use of the BRK key, the computer does not automatically reset the

A DISASSEMBLER PROGRAM

This program decodes sequences of machine language instructions and converts them to assembly language using *Rober Mnemonics*. It may be used to print listings of hand-assembled machine language programs written by the user. Or, it may be used to list the instructions stored in ROM in the Sharp PC-1500 or Radio Shack PC-2.

You need at least a 4K RAM module in your PC in order to utilize this disassembler program.

Introductory Notes

To begin using the disassembler, execute RUN and respond to the initial prompts by entering the beginning and ending addresses of the code that is to be disassembled.

The disassembler uses the printer to produce a list of the assembly language instructions. As provided, output uses CSIZE 2 characters. It will, however, work with CSIZE 1 printing if desired.

Relative branches are calculated and the branch address is printed alongside the instruction.

As an option, a sequence of stored codes (machine codes) may be printed (using hexadecimal notation) by selecting DEF A. Enter the beginning and ending addresses when prompted.

Disassembling ROM

Before you can successfully disassemble ROM, you need to know where the instruction code sequences are located! Remember, not all of ROM contains instructions. Some of it holds various lookup tables. Attempting to disassemble these areas will simply yield nonsense. The following is a rudimentary map of ROM in the PC-1500, showing major instruction code and non-code areas:

C001 – C01C	Code
C01D – C3FF	Non-code (mostly tables)
C400 – D6AC	Code
D6AD – D6BE	Short lookup table
D5BF – DCAD	Code
DCAE – DCB5	Non-code in PC–1500, however, in the PC-2 addresses DCAE – DCB2 contain code.
DCB6 – E167	Code
E168 – E170	Short lookup table
E171 – F950	Code
F951 – F956	Non-code (???)
F957 – FBDF	Code
FBE0 – FFFF	Non-code (tables)

A similar map of ROM in the CE-150 printer/cassette interface is provided next:

A000 – A28A	Non-code (table used in printing)
A28B –AFF9	Code
AFFA – B009	Non-code
B00A – B015	? ? ? ?
B016 – B0EA	Non-code
B0EB – B7FF	Code
B800 – B809	Non-code
B80A – B81C	Code (some questionable areas)
B81D – B887	Non-code
B888 – BB55	Code
BB56 – BB69	Short lookup table
BB6A – BFFC	Code

The disassembler does not print the addresses of base-page calls. I did have it do so in an earlier version of the program, but it proved to be more of a hindrance than a help in attempting to follow the operation of a ROM routine. The accompanying table provides the addresses of the routines referred to by base-page calls. (The base page is FF.)

It must be noted that some routines in the ROM pass parameters to subroutines. Typically, one or more of the bytes immediately following a JSR or CALL instruction are used for that purpose. (When this occurs then the called subroutine modifies the return address to skip over these bytes.)

The disassembler program takes care of parameter-passing for base-page subroutine calls. However, some of the subroutines called by JSR instructions also pass parameters. When a JSR to a subroutine that may pass parameters is encountered, the PC will stop and display the prompt:

PASS?

The user must then enter the number of bytes to be passed. I am including a list of subroutines that are known to pass 1 byte. When the program requests "PASS?" information, check this list. If the subroutine (whose address will have just been printed) is in this list, enter the digit 1. If it is not, enter the digit 0 or just press the ENTER key.

List *Addresses of Subroutines Passing 1 Byte*

CC86	D2EC	DAB4
CC8B	D407	DB95
CC9C	D40D	DBB3
D14C	D52A	DD2F
D14F	D6D9	DF9B
D2E0	D7CA	DFA0
D2EA	DAB2	DFA1

Tips on Interpreting Disassembled ROM

A subroutine may end with the CALL codes 48, 4A, 4C or 4E. If so, it is a subroutine that passes a parameter.

In many cases, a passed parameter is used to *increment* the return address, depending on the results of tests made within the subroutine. In the base-page calls having the base-page addresses: 00, 02, 04, 08, 0E, 1A, 28, 2C, 2E, C2, C4, C8, CE, D0, D2 and DE, the last byte passed may (or may not) be added to the return address.

The base-page subroutine called through base-page address 34 may select one of several passed values to modify the return address.

Additionally, each of the byte-passing subroutines *other than* the base-page ones, may or may not used the passed byte to increment the return address. I know of one exception: The subroutine at DD2F will *not* do this.

PC-1500/PC-2 Comparison

It is interesting to note that the ROM in the Radio Shack PC-2 is practically the same as that in the Sharp PC-1500. The few differences that do exist are probably minor revisions.

Table *Address of Subroutines Accessible through Base-Page FF.*

CALL:	ADDRESS:						
00	DCB7	40	C401	80	F707	C0	DD00
02	DCB8	42	CA58	82	F729	C2	DCD4
04	DCC5	44	CA7A	84	EF00	C4	DCD5
06	D065	46	CA80	86	E840	C6	DD13
08	D0D9	48	DCF9	88	EDF6	C8	DCC5
0A	DE5E	4A	DCFD	8A	ED50	CA	C001
0C	DE97	4C	DCE9	8C	EE1F	CC	DDC0
0E	D401	4E	DCED	8E	ED01	CE	D450
10	D020	50	DA71	90	EDA0	D0	D5F9
12	DF93	52	F663	92	ED00	D2	DD1A
14	DFFA	54	F780	94	EC5C	D4	DEE3
16	DFF5	56	F73D	96	EA70	D6	DE01
18	DF80	58	F004	98	EC74	D8	DF30
1A	D2E5	5A	E573	9A	ECE0	DA	C00E
1C	FA89	5C	F618	9C	EC87	DC	DEBC
1E	F82A	5E	F7A7	9E	E4A0	DE	D6DF
20	DF72	60	F604	A0	E234	E0	C088
22	DF63	62	F008	A2	E655	E2	C400
24	DEAF	64	F783	A4	8888	E4	[illegible]
26	D087	66	F709	A6	E451	E6	F78D
28	D081	68	F715	A8	8888	E8	F661
2A	D03E	6A	F80F	AA	888E	EA	F75C
2C	DCA6	6C	F6FB	AC	E88C	EC	F757
2E	D6C0	6E	F060	AE	8891	EE	F7CC
30	DC16	70	F747	B0	8894	F0	EF0A
32	D071	72	F7CE	B2	8897	F2	EE71
34	DF23	74	F775	B4	889A	F4	D80C
36	DF0F	76	F75F	B6	889D	F6	DD85
38	CE9F	78	F72F	B8	88A0	F8	E171
3A	CFF0	7A	F700	BA	F763	FA	E22C
3C	FA74	7C	F6E6	BC	E487	FC	E220
3E	F890	7E	F01A	BE	E4A0	FE	E000

```
10:INPUT "STARTIN
   G ADDRESS? ";A
11:INPUT "ENDING
   ADDRESS? ";B
12:CLS :ON ERROR
   GOTO 80
14:C=A:GOSUB 20:
   TAB 5:GOSUB 10
   0+PEEK A:A=A+1
   :LPRINT :IF A<
   =BGOTO 14
16:END
20:D=INT (C/256):
   GOSUB 24:D=C-2
   56*D:GOTO 24
22:A=A+1:D=PEEK A
24:E=INT (D/16):F
   =DAND 15:
   LPRINT CHR$ (E
   +48+7*(E>9));
   CHR$ (F+48+7*(
   F>9));:RETURN
26:GOSUB 22:GOTO
   22
30:TAB 10:LPRINT
   "#";:GOTO 22
32:TAB 15:LPRINT
   "#";:GOTO 22
34:TAB 10:GOTO 26
36:TAB 10:GOSUB 2
   6:GOTO 32
40:GOSUB 34:C=0:F
   =PEEK (A-1):IF
   F>208AND F<224
   OR F=204INPUT
   "PASS? ";C
42:CLS :IF C=0
   RETURN
44:LPRINT :TAB 5:
   LPRINT "PASS";
46:FOR I=1TO C:
   LPRINT " ";:
   GOSUB 22:NEXT
   I:RETURN
50:LPRINT "CALL "
   ;:D=PEEK A:
   GOSUB 24:GOTO
   54
52:GOSUB 30
54:C=0:IF D<47LET
   C=VAL MID$ ("3
   3101002100001 1
   000011211",D/2
   +1,1)
55:IF D=52LET C=3
   +2*PEEK (A+1)
56:IF D=194OR D=1
   96LET C=2+(
   PEEK (A+1)>223
   )
57:IF D>199LET C=
   VAL MID$ ("111
   2221100010000
   00000220000",D
   /2-99,1)
58:GOTO 42
60:C=1:GOTO 64
62:C=-1
64:GOSUB 30:
   LPRINT ",";:C=
   A+1+C*D:GOTO 2
   0
70:LPRINT "ALT BU
   FFER:":TAB 5:
   GOTO 100+PEEK
   A
80:IF PEEK (A-1)=
   253LPRINT "FD
   ";
82:D=PEEK A:GOSUB
   24:LPRINT "??"
   :A=A+1:GOTO 14
90:"A"INPUT "STAR
   TING ADDRESS?
   ";A
91:INPUT "ENDING
   ADDRESS? ";B
92:CLS :LPRINT "B
   EGINNING ";:C=
   A:GOSUB 20:
   LPRINT ":":A=A
   -1:C=B-A:GOSUB
   46:LPRINT :END
100:LPRINT "SUBA X
    L";:RETURN
101:LPRINT "SUBA (
    X)";:RETURN
102:LPRINT "ADDA X
    L";:RETURN
103:LPRINT "ADDA (
    X)";:RETURN
104:LPRINT "LDA  X
    L";:RETURN
105:LPRINT "LDA  (
    X)";:RETURN
106:LPRINT "CPA  X
    L";:RETURN
107:LPRINT "CPA  (
    X)";:RETURN
108:LPRINT "STA  X
    H";:RETURN
109:LPRINT "ANDA (
    X)";:RETURN
110:LPRINT "STA  X
    L";:RETURN
111:LPRINT "ORA  (
    X)";:RETURN
112:LPRINT "DSBA (
    X)";:RETURN
113:LPRINT "EORA (
    X)";:RETURN
114:LPRINT "STA  (
    X)";:RETURN
115:LPRINT "BITA (
    X)";:RETURN
116:LPRINT "SUBA Y
    L";:RETURN
117:LPRINT "SUBA (
    Y)";:RETURN
118:LPRINT "ADDA Y
    L";:RETURN
119:LPRINT "ADDA (
    Y)";:RETURN
120:LPRINT "LDA  Y
    L";:RETURN
121:LPRINT "LDA  (
    Y)";:RETURN
122:LPRINT "CPA  Y
    L";:RETURN
123:LPRINT "CPA  (
    Y)";:RETURN
124:LPRINT "STA  Y
    H";:RETURN
125:LPRINT "ANDA (
    Y)";:RETURN
126:LPRINT "STA  Y
    L";:RETURN
127:LPRINT "ORA  (
    Y)";:RETURN
128:LPRINT "DSBA (
    Y)";:RETURN
129:LPRINT "EORA (
    Y)";:RETURN
130:LPRINT "STA  (
    Y)";:RETURN
131:LPRINT "BITA (
    Y)";:RETURN
132:LPRINT "SUBA U
    L";:RETURN
133:LPRINT "SUBA (
    U)";:RETURN
134:LPRINT "ADDA U
    L";:RETURN
135:LPRINT "ADDA (
    U)";:RETURN
136:LPRINT "LDA  U
    L";:RETURN
137:LPRINT "LDA  (
    U)";:RETURN
138:LPRINT "CPA  U
    L";:RETURN
```

```
139:LPRINT "CPA  (
    U)";:RETURN
140:LPRINT "STA  U
    H";:RETURN
141:LPRINT "ANDA (
    U)";:RETURN
142:LPRINT "STA  U
    L";:RETURN
143:LPRINT "ORA  (
    U)";:RETURN
144:LPRINT "DSBA (
    U)";:RETURN
145:LPRINT "EORA (
    U)";:RETURN
146:LPRINT "STA  (
    U)";:RETURN
147:LPRINT "BITA (
    U)";:RETURN
156:LPRINT "NOP";:
    RETURN
164:LPRINT "INXL";
    :RETURN
165:LPRINT "STAI (
    X)";:RETURN
166:LPRINT "DEXL";
    :RETURN
167:LPRINT "STAD (
    X)";:RETURN
168:LPRINT "INX";:
    RETURN
169:LPRINT "LDAI (
    X)";:RETURN
170:LPRINT "DEX";:
    RETURN
171:LPRINT "LDAD (
    X)";:RETURN
172:LPRINT "LDXH";
    :GOTO 30
173:LPRINT "AND  (
    X)";:GOTO 32
174:LPRINT "LDXL";
    :GOTO 30
175:LPRINT "OR   (
    X)";:GOTO 32
176:LPRINT "CPXH";
    :GOTO 30
177:LPRINT "BIT  (
    X)";:GOTO 32
178:LPRINT "CPXL";
    :GOTO 30
179:LPRINT "ADD  (
    X)";:GOTO 32
180:LPRINT "INYL";
    :RETURN
181:LPRINT "STAI (
    Y)";:RETURN
182:LPRINT "DEYL";
    :RETURN
183:LPRINT "STAD (
    Y)";:RETURN
184:LPRINT "INY";:
    RETURN
185:LPRINT "LDAI (
    Y)";:RETURN
186:LPRINT "DEY";:
    RETURN
187:LPRINT "LDAD (
    Y)";:RETURN
188:LPRINT "LDYH";
    :GOTO 30
189:LPRINT "AND  (
    Y)";:GOTO 32
190:LPRINT "LDYL";
    :GOTO 30
191:LPRINT "OR   (
    Y)";:GOTO 32
192:LPRINT "CPYH";
    :GOTO 30
193:LPRINT "BIT  (
    Y)";:GOTO 32
194:LPRINT "CPYL";
    :GOTO 30
195:LPRINT "ADD  (
    Y)";:GOTO 32
196:LPRINT "INUL";
    :RETURN
197:LPRINT "STAI (
    U)";:RETURN
198:LPRINT "DEUL";
    :RETURN
199:LPRINT "STAD (
    U)";:RETURN
200:LPRINT "INU";:
    RETURN
201:LPRINT "LDAI (
    U)";:RETURN
202:LPRINT "DEU";:
    RETURN
203:LPRINT "LDAD (
    U)";:RETURN
204:LPRINT "LDUH";
    :GOTO 30
205:LPRINT "AND  (
    U)";:GOTO 32
206:LPRINT "LDUL";
    :GOTO 30
207:LPRINT "OR   (
    U)";:GOTO 32
208:LPRINT "CPUH";
    :GOTO 30
209:LPRINT "BIT  (
    U)";:GOTO 32
210:LPRINT "CPUL";
    :GOTO 30
211:LPRINT "ADD  (
    U)";:GOTO 32
228:LPRINT "SUBA X
    H";:RETURN
229:LPRINT "FBNC";
    :GOTO 60
230:LPRINT "ADDA X
    H";:RETURN
231:LPRINT "FBC";:
    GOTO 60
232:LPRINT "LDA  X
    H";:RETURN
233:LPRINT "FBNH";
    :GOTO 60
234:LPRINT "CPA  X
    H";:RETURN
235:LPRINT "FBH";:
    GOTO 60
236:LPRINT "BNZD";
    :GOTO 62
237:LPRINT "FBNZ";
    :GOTO 60
238:LPRINT "RTI";:
    RETURN
239:LPRINT "FBZ";:
    GOTO 60
240:LPRINT "DADA (
    X)";:RETURN
241:LPRINT "FBNV";
    :GOTO 60
242:LPRINT "FB";:
    GOTO 60
243:LPRINT "FBV";:
    GOTO 60
244:LPRINT "SUBA Y
    H";:RETURN
245:LPRINT "RBNC";
    :GOTO 62
246:LPRINT "ADDA Y
    H";:RETURN
247:LPRINT "RBC";:
    GOTO 62
248:LPRINT "LDA  Y
    H";:RETURN
249:LPRINT "RBNH";
    :GOTO 62
250:LPRINT "CPA  Y
    H";:RETURN
251:LPRINT "RBH";:
    GOTO 62
253:LPRINT "RBNZ";
    :GOTO 62
254:LPRINT "RTS";:
    RETURN
255:LPRINT "RBZ";:
    GOTO 62
256:LPRINT "DADA (
    Y)";:RETURN
257:LPRINT "RBNV";
    :GOTO 62
258:LPRINT "RB";:
    GOTO 62
259:LPRINT "RBV";:
    GOTO 62
260:LPRINT "SUBA U
    H";:RETURN
261:LPRINT "SUBA";
    :GOTO 34
262:LPRINT "ADDA U
    H";:RETURN
263:LPRINT "ADDA";
    :GOTO 34
264:LPRINT "LDA  U
    H";:RETURN
265:LPRINT "LDA";:
    GOTO 34
266:LPRINT "CPA  U
    H";:RETURN
267:LPRINT "CPA";:
    GOTO 34
268:LPRINT "(A8)";
    :RETURN
269:LPRINT "ANDA";
    :GOTO 34
270:LPRINT "LDS  #
    ";:GOTO 26
271:LPRINT "ORA";:
    GOTO 34
272:LPRINT "DADA (
    U)";:RETURN
273:LPRINT "EORA";
    :GOTO 34
274:LPRINT "STA";:
    GOTO 34
275:LPRINT "BITA";
    :GOTO 34
277:LPRINT "SUBA";
    :GOTO 30
279:LPRINT "ADDA";
    :GOTO 30
281:LPRINT "LDA";:
    GOTO 30
283:LPRINT "CPA";:
    GOTO 30
284:LPRINT "(B8)";
    :RETURN
285:LPRINT "ANDA";
    :GOTO 30
286:LPRINT "JMP";:
    GOTO 34
287:LPRINT "ORA";:
    GOTO 30
289:LPRINT "EORA";
    :GOTO 30
290:LPRINT "JSR";:
    GOTO 40
291:LPRINT "BITA";
    :GOTO 30
292:GOTO 50
293:LPRINT "CANC";
    :GOTO 52
294:GOTO 50
295:LPRINT "CAC";:
    GOTO 52
296:GOTO 50
297:LPRINT "CANH";
    :GOTO 52
298:GOTO 50
299:LPRINT "CAH";:
    GOTO 52
300:GOTO 50
301:LPRINT "CANZ";
    :GOTO 52
302:GOTO 50
303:LPRINT "CAZ";:
    GOTO 52
304:GOTO 50
305:LPRINT "CALL";
    :GOTO 52
306:GOTO 50
307:LPRINT "CAV";:
    GOTO 52
308:GOTO 50
309:LPRINT "RRCA";
    :RETURN
310:GOTO 50
311:LPRINT "RDR  (
    X)";:RETURN
312:GOTO 50
313:LPRINT "RRA";:
    RETURN
314:GOTO 50
315:LPRINT "RDL  (
    X)";:RETURN
316:GOTO 50
317:LPRINT "RLA";:
    RETURN
318:GOTO 50
319:LPRINT "RLCA";
    :RETURN
320:GOTO 50
321:LPRINT "INA";:
    RETURN
322:GOTO 50
323:LPRINT "DEA";:
    RETURN
324:GOTO 50
326:GOTO 50
328:GOTO 50
330:GOTO 50
332:GOTO 50
333:LPRINT "AND";:
    GOTO 36
334:GOTO 50
335:LPRINT "OR";:
    GOTO 36
336:GOTO 50
337:LPRINT "BIT";:
    GOTO 36
338:GOTO 50
339:LPRINT "ADD";:
    GOTO 36
340:GOTO 50
341:LPRINT "RDA";:
    RETURN
342:GOTO 50
344:GOTO 50
345:LPRINT "INXY";
    :RETURN
346:GOTO 50
347:LPRINT "CPAI (
    X)";:RETURN
348:GOTO 50
349:LPRINT "CLRC";
    :RETURN
350:GOTO 50
351:LPRINT "SETC";
    :RETURN
352:GOTO 50
353:A=A+1:GOTO 400
    +PEEK A
354:GOTO 50
401:GOTO 70
403:GOTO 70
405:GOTO 70
407:GOTO 70
409:GOTO 70
410:LPRINT "POPX";
    :RETURN
411:GOTO 70
412:GOTO 70
413:GOTO 70
414:GOTO 70
415:GOTO 70
417:GOTO 70
419:GOTO 70
421:GOTO 70
423:GOTO 70
424:LPRINT "LDX  Y
    ";:RETURN
425:GOTO 70
426:LPRINT "POPY";
    :RETURN
427:GOTO 70
428:GOTO 70
429:GOTO 70
430:GOTO 70
431:GOTO 70
```

```
433:GOTO 70
435:GOTO 70
437:GOTO 70
439:GOTO 70
440:LPRINT "LDX  U
    ";:RETURN
441:GOTO 70
442:LPRINT "POPU";
    :RETURN
443:GOTO 70
444:GOTO 70
445:GOTO 70
446:GOTO 70
447:GOTO 70
464:LPRINT "INXH";
    :RETURN
466:LPRINT "DEXH";
    :RETURN
472:LPRINT "LDX  S
    ";:RETURN
473:GOTO 70
475:GOTO 70
476:LPRINT "(FD 4C
    )";:RETURN
477:GOTO 70
478:LPRINT "STX  S
    ";:RETURN
479:GOTO 70
480:LPRINT "INYH";
    :RETURN
482:LPRINT "DEYH";
    :RETURN
488:LPRINT "LDX  P
    ";:RETURN
489:GOTO 70
490:LPRINT "STX  Y
    ";:RETURN
491:GOTO 70
493:GOTO 70
494:LPRINT "STX  P
    ";:RETURN
495:GOTO 70
496:LPRINT "INUH";
    :RETURN
498:LPRINT "DEUH";
    :RETURN
505:GOTO 70
506:LPRINT "STX  U
    ";:RETURN
507:GOTO 70
509:GOTO 70
511:GOTO 70
529:LPRINT "SETI";
    :RETURN
536:LPRINT "PSHX";
    :RETURN
538:LPRINT "POPA";
    :RETURN
540:GOTO 70
552:LPRINT "PSHY";
    :RETURN
556:GOTO 70
561:GOTO 70
563:GOTO 70
565:GOTO 70
567:GOTO 70
568:LPRINT "PSHU";
    :RETURN
569:GOTO 70
570:LPRINT "LDA  E
    ";:RETURN
571:GOTO 70
572:GOTO 70
573:GOTO 70
574:GOTO 70
575:GOTO 70
577:LPRINT "WAI";:
    RETURN
586:LPRINT "LDA  K
    B";:RETURN
590:LPRINT "CLRI";
    :RETURN
592:LPRINT "(FD C0
    )";:RETURN
593:LPRINT "(FD C1
    )";:RETURN
600:LPRINT "PSHA";
    :RETURN
602:LPRINT "ADDX A
    ";:RETURN
606:LPRINT "(FD CE
    )";:RETURN
611:GOTO 70
615:GOTO 70
618:LPRINT "ADDY A
    ";:RETURN
633:GOTO 70
634:LPRINT "ADDU A
    ";:RETURN
635:GOTO 70
636:LPRINT "STA  E
    ";:RETURN
637:GOTO 70
639:GOTO 70

STATUS 1
                    5595
```

The POCKET COMPUTER NEWSLETTER Is Available!
By Subscription Only: for a calendar year period (January – December). You get all issues published to date for the calendar year in which you subscribe, at the time you subscribe.
MC/VISA Phone Subscriptions: (203) 888-1946

- ☐ 1982 Regular Subscriber (Issues 11 – 20). $30.00 in U.S. (U.S. $36.00 to Canada. U.S. $45.00 elsewhere.)
- ☐ 1982/83 Subscriber (Issues 11 – 30). $60.00 in U.S. (U.S. $72.00 to Canada. U.S. $90.00 elsewhere.)
- ☐ 1983 Regular Subscriber (Issues 21 – 30). $36.00 in U.S. (U.S. $42.00 to Canada. U.S. $50.00 elsewhere.)

Orders must be accompanied by payment in full. We do not issue invoices for the POCKET COMPUTER NEWSLETTER.
Thank you for your remittance.

Name: ______________________________
Addr: ______________________________
City: ______________ State: ___ Zip: ______
MC/VISA #: ______________ Expires: ________
Signature: ______________________________

P.O. Box 232, Seymour, CT 06483

LATEST FINDINGS

As this Special Edition of *PCN* went to press, Norlin reported that he had ascertained the coding for several more instructions. They are associated with the processing of interrupts and Input/Output operations as described here. The instructions with the mnemonics RTI, WAI and LDA KB have been incorporated into the disassembler listing provided on the preceeding pages.

Rober Mnemonic	Machine Code	Description
SETI	FD 81	Flag I set, interrupt enabled.
CLRI	FD BE	Flag I cleared, interrupt disabled.
RTI	8A	Return from interrupt.
WAI	FD B1	Wait for interrupt.
LDA KB	FD BA	Load accumulator with input from keyboard.

The final instruction in the list, LDA KB, results in a byte being placed in the accumulator. The byte loaded is determined by the row of the key matrix in which a key is depressed. The bit in the position corresponding to that row is 0, the other bits are 1.

What is Left?

Norlin reports that the following machine codes, which appear related to I/O operations, have not yet been fully defined. These codes do, however, appear in the PC-1500's ROM: A8, B8, FD 4C, FD C0, FD C1 and FD CE. Any ideas? Let Norlin know!