POCKET COMPUTER NEWSLETTER

WWW.
PC-1500
.INFO

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

PART II

Issues 29 to 33

capability. (If you are using the LMD package, you should renumber the input routines so they are above line number 999. Then they will not be mistaken as lines of source code. Label the start of each routine so you can access it from the RUN mode using a labeled directive.)

If you do this you will note that while lines can be drawn considerably faster than when the ON routine was entirely in BASIC, the process is still relatively slow. In order to obtain impressive speed it will be necessary to also convert the LINE routine from BASIC to machine language. Are you ready to tackle that task? Why not give it a try. You can compare your method with that presented in the next installment (which will conclude this series).

A RENUMBERING UTILITY

The Sharp PC-1500 and Radio Shack PC-2 are tremendously powerful devices in the hands of even a modest programmer. However, the recent introduction of the long awaited 16K RAM modules has reminded me of the major stumbling block in writing long application programs on these PCs: the lack of an easy to use renumbering utility. The purpose of this article is to describe an extremely fast "renumbering" routine written in only three lines of BASIC code.

Why do I consider the availability of a renumbering utility to be so important? The BASIC programming language itself is at fault. When a program branches through a GOTO or GOSUB it is necessary to specify a destination line number. Unless you are one of the very few who completely plans out programs in advance on paper, you will find yourself needing to decide what line number your program should jump to as you create code.

Can this be a problem? Suppose that you are going to frequently be calling an as yet unwritten subroutine that will, say, evaluate the polynomial P(x). You make a note on paper that the subroutine will be at line 2000. Every time you need this routine you must either remember it or else look up the the subroutine's starting line number. If you come back to update the program at a later date to modify it you will find yourself in the unpleasant situation of having to trace each subroutine reference in order to understand the program (unless you have otherwise documented the program in a thorough fashion). Things can rapidly get out of a hand in a long program where there may be dozens of GOTOs and GOSUBs. Remember, six months after the fact, a reference to line number 2000 might not do much to remind you that it handles P(x).

Fortunately, Sharp designed the PC-1500 with an unusually powerful feature that is lacking in many home computers: labeling. Any line may be labeled. All that is needed is to place the label in quotes immediately after the line number. You do not even have to place a colon between the label and the first statement in a line.

To jump to the line, you only need to state GOTO or GOSUB followed by the label of the desired line. This label reference must be enclosed in quotation marks. Thus, GOSUB 2000 might be replaced by GOSUB "P(x)" which is much more descriptive than GOSUB 2000. In fact, it is a fair example of self-documenting code.

A programmer has a virtually unlimited range of choices for labels since a label on the PC-1500 can be up to 73 characters in length and include blanks, upper and lower case letters, numerals and any other symbols except the quotation mark.

Since the computer supports this powerful feature, I strongly recommend that PC-1500 programmers use labels for all references to other routines and subroutines.

If the previously cited reasons are not sufficient to convince you of the worthiness of using labels, I can offer one more tantalizing appeal: the following program can renumber a 300 line program in about one minute. This program resides in just three lines of code. But, it only operates on programs wherein all GOTO and GOSUB references are by labels, not line numbers!

Here is the magic routine:

```
65024 END
65025 "L"B=STATUS
      2-STATUS 1,L
      =0
65026 IF PEEK B>25
      3THEN END
65027 L=L+10:POKE
      B,L/256,L-25
      6*INT (L/256
      ):B=B+3+PEEK
      (B+2):GOTO 6
      5026
```

The program may be executed by typing DEF L or RUN 65025. The program then renumbers all lines lower than 65024, assigning the first line number the value of 10 and using an increment between lines of 10. Note that the program does *not* change GOTO and GOSUB statements! Those statements *must* refer to labeled lines.

Here is an outline of how the renumbering program operates:

The first line is just a precaution that prevents other programs from accidently dropping down to the renumbering program.

Line 65025 finds and stores the address of the first byte of your program. This address is stored in variable B. It utilizes the STATUS function so the procedure works regardless of the amount of memory installed in your PC. This line also initializes the variable L which is used to store new line numbers.

The best way to understand the rest of the routine is to recall how programs themselves are stored in the PC-1500. Each BASIC line starts with a two-byte value, most significant byte first, that represents its line number. The third byte is equal to N+1 bytes, where N is the number of bytes in the body of the tokenized BASIC line. The next N bytes are those comprising the tokenized line. The byte after these N bytes (N+1) is the ASCII code for a carriage return (13). All line numbers must be greater than zero. The most significant byte of a line number must be less than 255. This latter restriction limits the programmer to using line numbers that are less than 65280. The last user line in the computer is terminated by the code 255.

The renumber utility ignores all line numbers above 65023. Thus, you can store other commonly used utility routines, such as a base converter, above the renumbering routine, etc. Note that line 65026 in the program terminates its own operation when a line number greater than 65023 is found.

Line 65027 increments the line number (variable L) by 10. It then POKEs it into the proper two bytes as indicated by variable B. Variable B is then incremented by N+4 so that it points to the address of the next BASIC line number. The program then goes back to line 65026 to determine whether it is to continue renumbering.

You may customize this program. The line increment set in line 65027 by the statement L=L+10 may be replaced with whatever increment value suits you. And, instead of setting L=0 in line 65025, you may set L equal to whatever offset you desire, such that the first line number becomes that offset value *plus* the line increment value.

Thanks for this utility procedure and routine go to: *Norman E. Beam, 21051 Gresham Street #103, Canoga Park, CA 91304.*

STAR TARGETS

This is a graphics game for the Radio Shack PC-2 and Sharp PC-1500. It was designed by *David A. Cloutier, Bullard Road, North Brookfield, MA 01535.* You need at least a 4K RAM module in your PC in order to use this program.

To play the game, imagine that you are a ball turret gunner in a space freighter. Your job is to eliminate all enemy fighters trying to destroy the freighter. When the title flashes on the screen, press ENTER *twice* and you will be all set for battle. Press the left arrow key to rotate your turret to the left. Use the right arrow key to rotate it to the right. (Note that the enemy will shift in the opposite direction.) Use the keys to position the enemy into the sights. When the sights start blinking, press the spacebar to fire!

```
3005:REM Star Tar
     gets
3010:CLEAR :CLS :
     DIM S$(0)*18
     ,SU$(0)*18,S
     D$(0)*18,C$(
     0)*18
3015:DIM D1$(0)*2
     2,D2$(0)*22,
     V$(0)*22
3020:S$(0)="120C1
     A2E2A2E1A0C1
     2"
3025:D1$(0)="0008
     2255361D6E10
     244B3A"
3030:SU$(0)="0906
     0F1517150F06
     09"
3035:D2$(0)="0002
     28120D3E0C32
     040012"
3040:SD$(0)="6418
     3C343C343C18
     64"
3045:V$(0)="63410
     000081C08000
     04163"
3047:GOSUB 3300
3050:T=TIME :WAIT
     0:G=60:
     GCURSOR G:
     GPRINT S$(0)
     ;
3060:C=0:RANDOM :
     R=RND 3:IF R
     =1THEN LET C
     $(0)=S$(0)
3070:IF R=2THEN
     LET C$(0)=SU
     $(0)
3080:IF R=3THEN
     LET C$(0)=SD
     $(0)
3090:BEEP 1,15,50
     :GR=(RND 8)-
     4:IF GR=0
     THEN LET GR=
     -4
3095:G=G+GR:IF G>
     155THEN LET
     G=0
3097:IF G<0THEN
     LET G=155
3098:GOSUB 3500:
     GOSUB 3400
3100:BEEP 1,20,50
     :CLS :
     GCURSOR G:
     GPRINT C$(0)
     :GOTO 3060
3300:WAIT 0:PRINT
     CHR$ 127;" S
     tar Targets
      V1.8 !";:
     CURSOR 25:
     PRINT CHR$ 1
     27
3302:IF INKEY$ =
     CHR$ 13THEN
     3302
3304:A=RND 200:B=
     RND 50
3306:GCURSOR 1:
     BEEP 1,A,B:
     GPRINT A,B;:
     GCURSOR 151:
     GPRINT A,B;
3308:IF INKEY$ <>
     CHR$ 13THEN
     3304
3310:CLS :PRINT "
     By David A.
     Cloutier":
     GOSUB 3850
3320:RETURN
3400:P=RND 7:IF P
     <>1THEN
     RETURN
3410:CLS :GCURSOR
     G:GPRINT "08
     04";C$(0);"0
     408";
3420:BEEP 5,7,200
     :P=RND 10:IF
     P<>1THEN
```

Program *Star Targets.*

```
     RETURN
3430:G1=78:G2=79
3440:BEEP 1,G2-70
     ,25:GCURSOR
     G1:GPRINT 25
     5-POINT G1:
     GCURSOR G2:
     GPRINT 255-
     POINT G2
3445:G1=G1-1:G2=G
     2+1
3450:IF G1<0OR G2
     >155THEN 345
     5
3452:GOTO 3440
3455:PRINT "You w
     ere blown to
      pieces!":
     GOSUB 3850
3460:PRINT "Score
      =";SC:GOSUB
     3850
3470:PRINT "Ships
      =";WS:GOSUB
     3850
3480:END
3500:GCURSOR 73:
     GPRINT "41";
     :GCURSOR 82:
     GPRINT "41"
3505:A$=INKEY$ :
     GOSUB 3590
3510:IF G<>74THEN
     RETURN
3520:CLS :GCURSOR
     73:GPRINT U$
     (0);:BEEP 1,
     8,100:A$=
     INKEY$
3530:IF A$=" "
     THEN 3600
3540:CLS :GCURSOR
     G:GPRINT S$(
     0);:BEEP 1,8
     ,200:C=C+1
3550:IF C=3THEN
     RETURN
3560:GOTO 3520
3590:IF A$=CHR$ 1
     2THEN LET G=
     G-5
3592:IF G<0THEN
     LET G=155
3595:IF A$=CHR$ 8
     THEN LET G=G
     +5
3596:IF G>155THEN
     LET G=0
3597:IF A$=" "
     THEN 3599
3598:RETURN
3599:A$=INKEY$ :
     GOTO 3597
3600:GCURSOR 73:
     GPRINT D2$(0
     );:BEEP 5,9,
     200
3610:CLS :GCURSOR
     73:GPRINT D1
     $(0);:BEEP 5
     ,20,50
3620:G1=72:G2=84
3630:GCURSOR G1:
     GPRINT (RND
     128)-1:
     GCURSOR G2:
     GPRINT (RND
     128)-1:BEEP
     1,G2,5
3640:G1=G1-4:G2=G
     2+4:IF G1<=0
     OR G2>=155
     THEN LET WS=
     WS+1:GOSUB 3
     700:GOTO 305
     0
3650:GOTO 3630
3700:SC=INT (SC+1
     00-(100*(
     TIME -T)))
3720:PRINT "Score
      =";SC:GOSUB
     3850
3730:PRINT WS;" e
     nemy ships d
     estroyed":
     GOSUB 3850
3840:GOTO 3860
3850:IF INKEY$ =
     CHR$ 13THEN
     3850
3855:IF INKEY$ <>
     CHR$ 13THEN
     3855
3857:IF INKEY$ <>
     CHR$ 13THEN
     3857
3860:CLS :RETURN
```

REDEFINING THE PC-1500 KEYBOARD

If you have POKEd around in the Sharp PC-1500 you may have noticed a little Japanese symbol appear in the display. This occurs when bit 2 of location &764E is set. (You can see it by entering: POKE &764E,PEEK &764E OR 4.) When the PC is in this mode, keyboard codes may be fetched from a user-built table. To do this, a pointer stored at &785D must be set to &80 when using a CE-150 or to &00 when using a CE-158 (RS-232) interface.

In this discussion, the use of a CE-150 will be assumed. For this case the user-built table must start at an address: &nn80. The high byte of this address (nn) is indicated to ROM by setting the pointer at &785E to (nn+1).

The first keyboard table contains 128 bytes. Sixty-four are used for the regular keyboard and 64 for when the shift key is utilized. Keys to be redefined must have ASCII codes greater than A0.

A second table holds GPRINT codes -- five per symbol -- that are used to define each character. This table is located at address &(nn+1)A0. The length of this second table depends on the number of keys redefined. It takes five bytes to define each key.

Thus, the directive POKE &785D,&80,&3A will indicate to ROM that a keyboard table starts at &3980 (going to &39FF) and that there is a GPRINT table starting at &3AA0.

(Keys having ASCII codes less than &80 will be fetched from the regular ROM table at addresses &FCA0 through &FE7F.)

The program shown in the accompanying listing capitalizes on this information. It *adds* a set of Greek alphabet characters to the repertoire of the PC-1500. Note that I said *adds* and not *substitutes.* All other capabilities remain. Thus, by a single keystroke, you may switch from one keyboard representation to another.

Actually, the program only redefines 35 keys. This provides 24 lowercase Greek letters, 10 uppercase Greek symbols (as the other 14 uppercase Greek characters are identical to English ones), and the apostrophe which Sharp did not bother to provide. To facilitate using these new symbols within BASIC programs, only lowercase keyboard characters were redefined.

The program consist of two parts. One POKEs the two necessary tables into the low portion of memory. The other is a little machine language routine that activates the Greek mode whenever desired.

Of course, these keyboard tables must be protected by an appropriate NEW statement to isolate them from the user's BASIC storage area. However, the program works with any RAM memory expansion (4K, 8K or 16K modules). This is because the first available address is computed after examining the SOM (Start Of Memory) pointer that is maintained in address &7863.

Installation and Use

The first step you must take is to protect a section of low memory where the table will be stored. Do this by executing the command:

```
NEW(PEEK &7863+2)*256+336
```

(If you are using an 8K RAM module, then you could just enter: NEW &3850.)

Next, load the entire BASIC program shown in the accompanying listing into memory.

Execute this program by typing RUN in the RUN mode. This causes the two tables to be placed into the protected area of low memory. After a few seconds the > prompt will come back up on the LCD. Now you can eliminate all of the BASIC lines *except for line 300.*

To save the keyboard tables on tape for future use, set up two variables by executing these statements:

```
A=(PEEK &7863+1)*256+128
B=(PEEK &7863+2)*256+335
```

Now place the tape unit in the record mode and execute the command:

```
CSAVE M "GREEK KB TABLE";A,B
```

(Again, if you are using an 8K RAM module, then use: CSAVE M "GREEK KB TABLE",&3980,&3B4F.)

Then, in the future you need only reserve the protected area in memory and load the keyboard table directly into memory using the command:

```
CLOAD M "GREEK KB TABLE";(PEEK &7863+2)*
256+128
```

You also need to restore line 300 (but you may renumber that as desired). Alternately, you may assign the contents of that line to a Reserved key:

```
(CALL(PEEK&7863+2)*256)
```

However, make sure you erase this assignation when the Greek table is not in memory!

That is all there is to it. Press the DEF/G key (or the Reserved key if you set it up as suggested) when you want to use Greek symbols. The Japanese

Diagram *Symbols Assigned to Numeric Keys.*

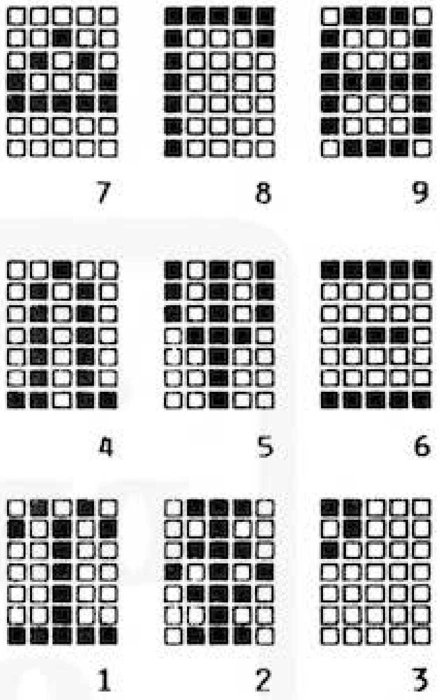
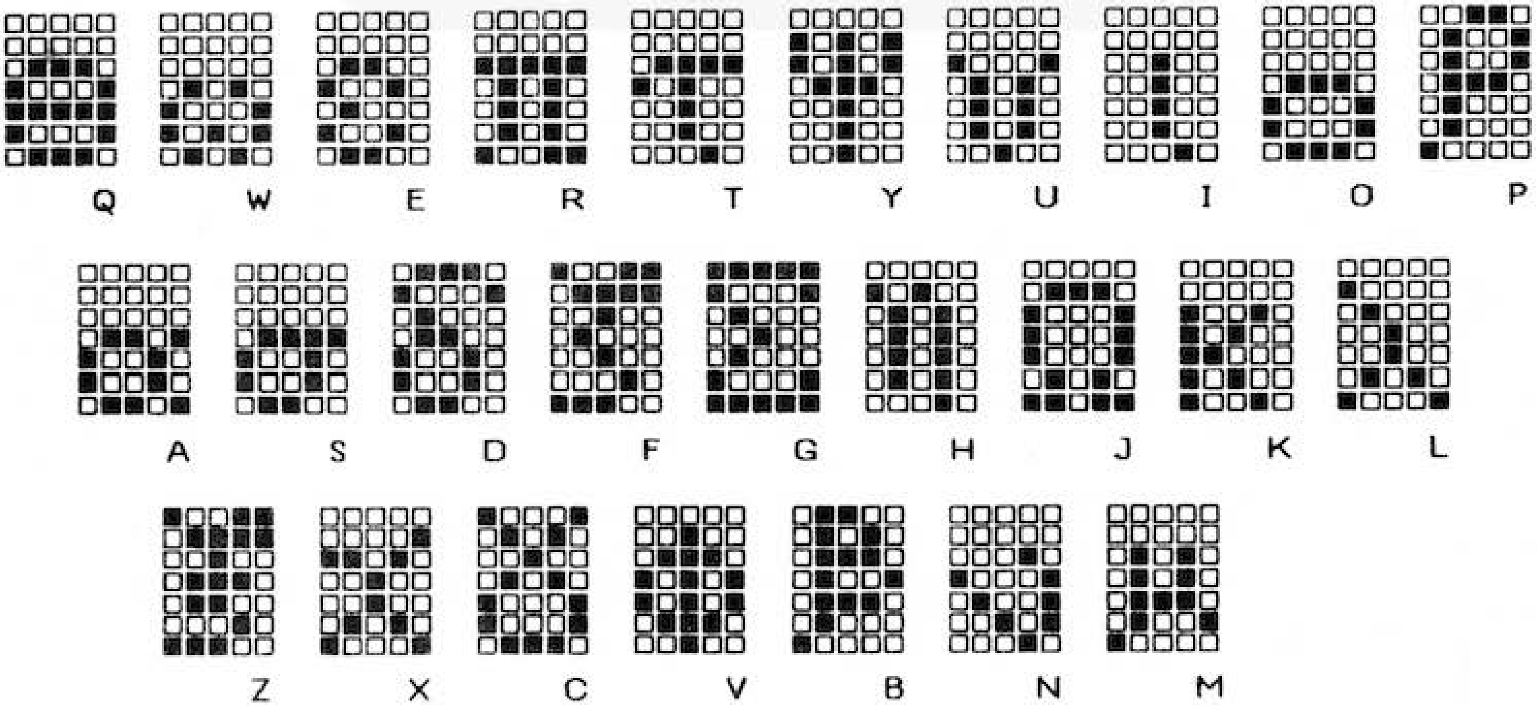


Diagram *Greek Symbols Assigned to Alphabetical Keys.*



symbol will come up on the display (to the left of the SMALL legend) to indicate that you are in this special keyboard mode. Remember these symbols are accessed using lowercase, so press the SHIFT key before pressing the desired alphabetical or numeric key. An accompanying diagram shows the Greek keyboard layout.

To switch back to the regular keyboard, press the SML key twice.

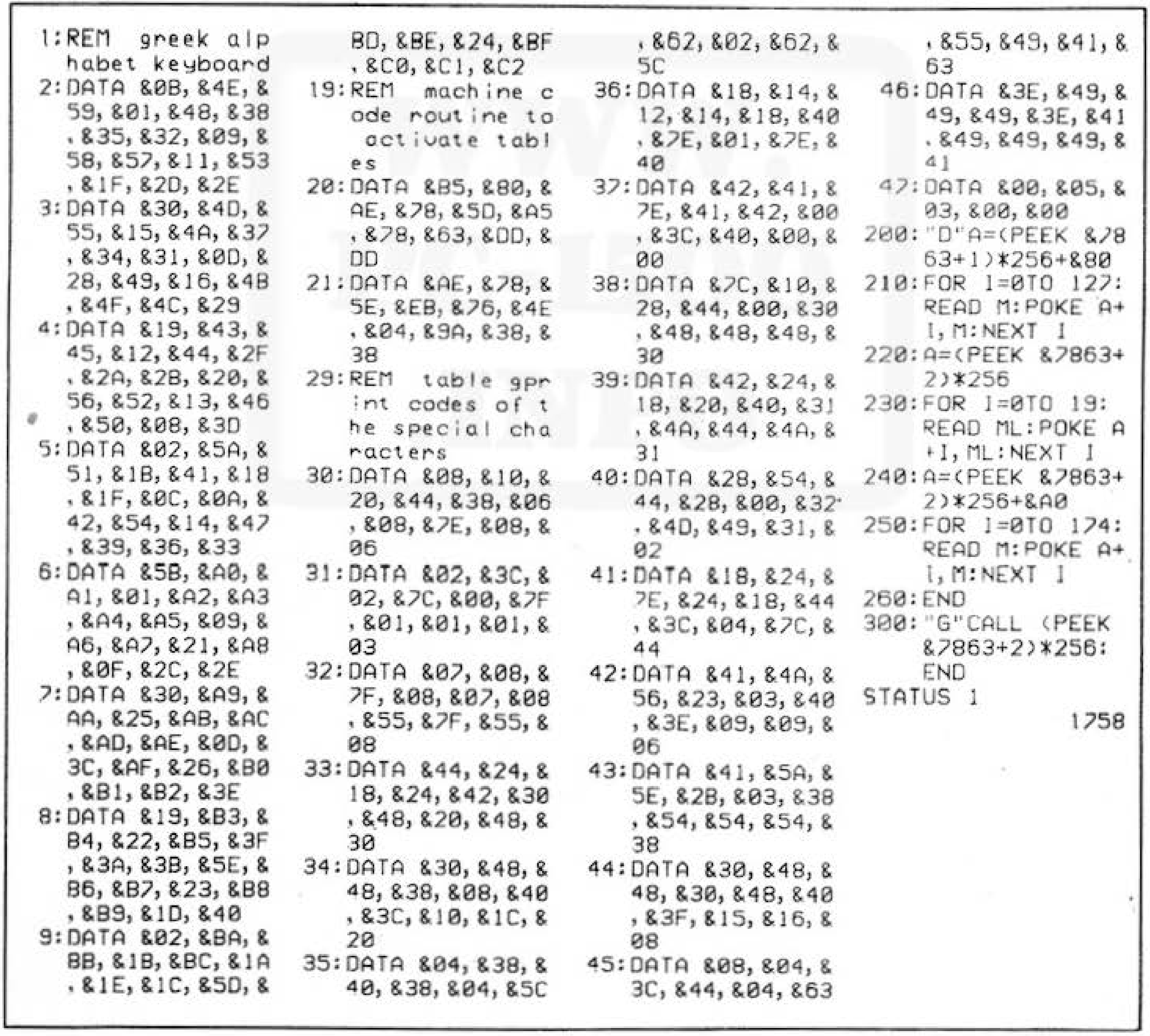
If you turn the PC off using the BREAK key, you will automatically quit the Greek mode. However, if the PC turns off by powering-down after the 7 minute time-out, you will remain in whatever mode you were using.

Of course, you may use these new Greek symbols just as you would any other characters: they can serve in labels, PRINT, PAUSE and INPUT statements, as well as file names.

Naturally the technique demonstrated here may be modified to enable your PC to communicate in whatever language you desire, be it Russian, Hebrew, or Sanscript. How about converting your PC to French, complete with accentuated vowels?

This powerful technique for customizing your PC keyboard was provided by: *Patrick L. Pollet, P.O. Box 144, UPM Box 298, Dhahran International Airport, Saudi Arabia.*

Program *Greek Keyboard*



```
  1:REM  greek alp
    habet keyboard
  2:DATA &0B,&4E,&
    59,&01,&48,&38
    ,&35,&32,&09,&
    58,&57,&11,&53
    ,&1F,&2D,&2E
  3:DATA &30,&4D,&
    55,&15,&4A,&37
    ,&34,&31,&0D,&
    28,&49,&16,&4B
    ,&4F,&4C,&29
  4:DATA &19,&43,&
    45,&12,&44,&2F
    ,&2A,&2B,&20,&
    56,&52,&13,&46
    ,&50,&08,&3D
  5:DATA &02,&5A,&
    51,&1B,&41,&18
    ,&1F,&0C,&0A,&
    42,&54,&14,&47
    ,&39,&36,&33
  6:DATA &5B,&A0,&
    A1,&01,&A2,&A3
    ,&A4,&A5,&09,&
    A6,&A7,&21,&A8
    ,&0F,&2C,&2E
  7:DATA &30,&A9,&
    AA,&25,&AB,&AC
    ,&AD,&AE,&0D,&
    3C,&AF,&26,&B0
    ,&B1,&B2,&3E
  8:DATA &19,&B3,&
    B4,&22,&B5,&3F
    ,&3A,&3B,&5E,&
    B6,&B7,&23,&B8
    ,&B9,&1D,&40
  9:DATA &02,&BA,&
    BB,&1B,&BC,&1A
    ,&1E,&1C,&5D,&
    BD,&BE,&24,&BF
    ,&C0,&C1,&C2
 19:REM  machine c
    ode routine to
     activate tabl
    es
 20:DATA &B5,&80,&
    AE,&78,&5D,&A5
    ,&78,&63,&DD,&
    DD
 21:DATA &AE,&78,&
    5E,&EB,&76,&4E
    ,&04,&9A,&38,&
    38
 29:REM  table gpr
    int codes of t
    he special cha
    racters
 30:DATA &08,&10,&
    20,&44,&38,&06
    ,&08,&7E,&08,&
    06
 31:DATA &02,&3C,&
    02,&7C,&00,&7F
    ,&01,&01,&01,&
    03
 32:DATA &07,&08,&
    7F,&08,&07,&08
    ,&55,&7F,&55,&
    08
 33:DATA &44,&24,&
    18,&24,&42,&30
    ,&48,&20,&48,&
    30
 34:DATA &30,&48,&
    48,&38,&08,&40
    ,&3C,&10,&1C,&
    20
 35:DATA &04,&38,&
    40,&38,&04,&5C
    ,&62,&02,&62,&
    5C
 36:DATA &18,&14,&
    12,&14,&18,&40
    ,&7E,&01,&7E,&
    40
 37:DATA &42,&41,&
    7E,&41,&42,&00
    ,&3C,&40,&00,&
    00
 38:DATA &7C,&10,&
    28,&44,&00,&30
    ,&48,&48,&48,&
    30
 39:DATA &42,&24,&
    18,&20,&40,&31
    ,&4A,&44,&4A,&
    31
 40:DATA &28,&54,&
    44,&28,&00,&32
    ,&4D,&49,&31,&
    02
 41:DATA &18,&24,&
    7E,&24,&18,&44
    ,&3C,&04,&7C,&
    44
 42:DATA &41,&4A,&
    56,&23,&03,&40
    ,&3E,&09,&09,&
    06
 43:DATA &41,&5A,&
    5E,&2B,&03,&38
    ,&54,&54,&54,&
    38
 44:DATA &30,&48,&
    48,&30,&48,&40
    ,&3F,&15,&16,&
    08
 45:DATA &08,&04,&
    3C,&44,&04,&63
    ,&55,&49,&41,&
    63
 46:DATA &3E,&49,&
    49,&49,&3E,&41
    ,&49,&49,&49,&
    41
 47:DATA &00,&05,&
    03,&00,&00
200:"D"A=(PEEK &78
    63+1)*256+&80
210:FOR I=0TO 127:
    READ M:POKE A+
    I,M:NEXT I
220:A=(PEEK &7863+
    2)*256
230:FOR I=0TO 19:
    READ ML:POKE A
    +I,ML:NEXT I
240:A=(PEEK &7863+
    2)*256+&A0
250:FOR I=0TO 174:
    READ M:POKE A+
    I,M:NEXT I
260:END
300:"G"CALL (PEEK
    &7863+2)*256:
    END
STATUS 1
                1758
```

POCKET
COMPUTER
NEWSLETTER

Issue 30 - December

CARLETON 16K EPROM MODULES

Carleton Micro Systems has announced the availability of a 16K EPROM Module for the Sharp PC-1500 and PC-1500A or the Radio Shack PC-2 computers. Using this new module, program source codes can be stored permanently. Furthermore, Carleton states that the source code in these modules can be protected against any kind of viewing, copying or editing. This is accomplished by disabling the LIST, LLIST, CSAVE and editing functions.

The firm claims that these modules are much more reliable than battery backup RAM modules. Using these modules, complete software packages may be quickly interchanged by simply plugging them into the computer hardware.

Using a Carleton EPROM Module with a Sharp PC-1500A results in a system containing 16K of program ROM and 8K of data RAM.

The company can either burn a program into the module or assist software vendors in programming the modules on their own. Normally, software

received on cassette or battery backup RAM is transferred to the Carleton ROM Modules. Carleton works under standard non-disclosure terms to protect vendors software. Orders for as many as 250 ROM Modules can be processed in less than a week.

The service is primarily designed to support software vendors who wish to produce modules in quantity. A minimum module order is normally 25 pieces. At this quantity, a 16K EPROM Module is priced at $150.00 each, plus an initial programming setup fee. The price declines to the range of $100.00 per module for higher quantities.

For further information contact: **Carleton Micro Systems**, *1801 Commerce Drive, South Bend, IN 46624*. The phone number is (219) 236-4630.

HEWLETT-PACKARD ANNOUNCES HP-41CX

Hewlett-Packard Corporation keeps striving to bring pocket computer capabilities to its line of Series 40 advanced programmable calculators. Now the firm has added clock and calendar functions, plus text-file editing and extended memory to its HP-41 line in its new HP-41CX.

The new model has all the features of its predecessor, the HP-41CV, plus those mentioned in the preceding paragraph, 20 new commands, and over 3,100 bytes of memory.

The Time Module in the HP-41CX enables the operator to use the highly portable unit as a time-based controller, an alarm clock, an appointment reminder, a timer or a stopwatch.

The Extended Functions/Memory Module in the HP-41CX (which was an option in the 41CV) provides 868 bytes of extended memory, memory-management functions, programmable versions of several HP-41 functions, plus several register and flag manipulation functions.

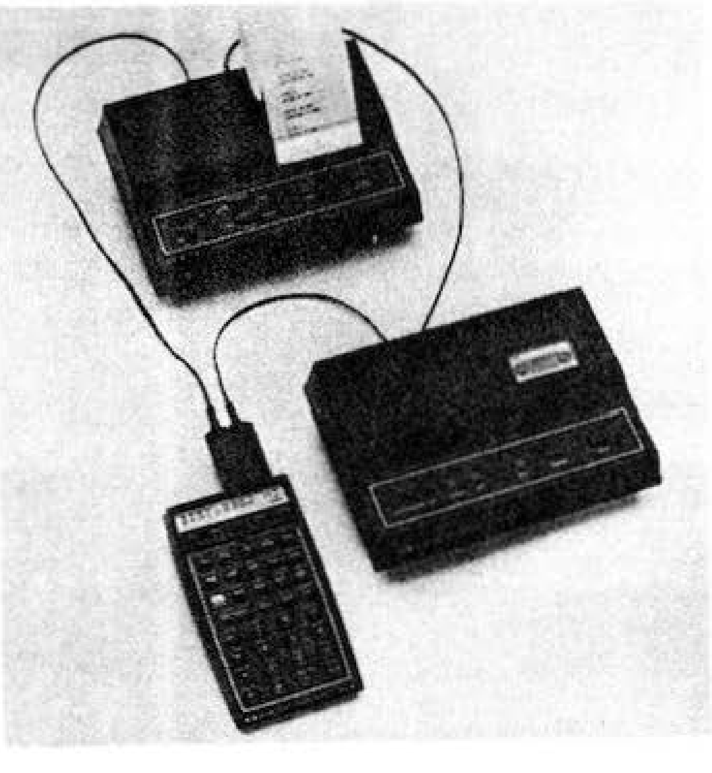
The 24K bytes of ROM in the HP-41CX features an RPN operating system. This sytem permits users to view intermediate results as well as recover easily from error conditions. The RAM in the HP-41CX retains its contents when the unit is turned off. An alphanumeric LCD screen permits viewing of both numeric and alphabetical characters plus special symbols and operators. The alphanumeric keyboard is redefinable. Thus, users may assign frequently used programs or functions to a single key to customize operation. Keyboard overlays are available for users who want to create their own labeled keyboards.

The HP-41CX can be expanded, via the HP-IL (Hewlett-Packard Interface Loop) . Peripherals include printers, plotters and instruments. All of these can be maintained as a battery-operated system allowing complete portability.

For more information on the $325.00 HP-41CX, contact your local HP sales office.

Photo *HP-41CX Programmable Calculator.*

Photo *HP Peripherals Connected Via HP-IL.*



ENHANCED FILE MAINTENANCE PROGRAM

One of the satisfying aspects of being the Editor of *PCN* is seeing the improvements that come about over time as a consequence of publishing hard to obtain information. As you know, during the past year (and more), *PCN* has published a great deal of information on machine language programming for the Sharp PC-1500 and Radio Shack PC-2. We are now seeing the results of having made this type of information widely available.

The program presented here is an excellent example of the types of improvements that can be obtained through the judicious application of ML techniques. This is an enhanced version of the File Maintenance program that originally appeared in Issue 24 of *PCN*. It was developed by the originator of the program, *Stephen Tomback, 19 Maplewood Way, Pleasantville, NY 10570*. Through the use of machine language routines, Mr. Tomback has vastly improved the overall operating speed of the program. Additionally, he has removed the former barrier (imposed by BASIC) that limited record lengths to 80 characters. If you found the original version useful, you will be ecstatic over this one!

Overview

The File Maintenance program will allow the set up of a filing system to a user's specific needs. Think of a file box filled with 3 x 5 cards. The box is called a *file*, the cards are called *records*, and the lines on the cards are called *fields*.

This programs allows the user to define the fields on the cards, store information in those fields, and have the ability to retrieve cards by searching fields for specific information. Thus, one could utilize this file maintenance program as a filing system for such things as birthdays, anniversaries, magazine subscriptions, or investments. A periodic search (such as monthly) would serve as a reminder as to which items were coming due.

Equipment Required

This program operates in a Radio Shack PC-2 or a Sharp PC-1500 that is equipped with an 8K memory expansion module. In order to make use of the program you will also need a printer/cassette interface such as the Sharp CE-150.

About the File

Here is important information for you to know about the structure of the file:

Number of Records: The program utilizes just under 3K of memory. The remainder is used for storage of the file. Using an 8K memory expansion module gives the PC a total memory capacity of about 10 kilobytes. Thus, about 7 kilobytes would be available for data storage. The number of records that can be stored will depend on the total number of bytes assigned to a record. For instance, a file utilizing 100 bytes per record would allow for the storing of approximately 70 records.

File Name: When a file is created, it must be assigned a name. This name may not exceed 15 characters. The assigned name is saved with the file. (Using a file name that is shorter than 15 bytes will not increase the amount of data that can be stored.)

Field Name: A record may contain up to 12 fields. (A field is simply an item within a record. A name, an address, and a telephone number might each constitute a field.) When a file is created, the user must assign a name to each field. That name must not exceed 15 characters. Again, assigning less than 15 characters to a field name will not make additional room for the storage of data.

Field Length: The length of a field may be set at from 1 to 40 bytes.

Record Length: The record length is a function of the number of field(s) defined and their length(s). Since a field may have a maximum of 40 bytes and a record may contain up to 12 fields, the maximum record length is 480 bytes. (A record this size would not be practical in most situations because it would only allow approximately 14 records in a file.)

Fields Per Record: A record may have from 1 to 12 fields.

Field Search: A record may be searched by 1 to 4 fields. If you set up a file of birthdays as well as anniversaries, it would be beneficial to establish a field called TYPE. This field would be used to distinguish between the two categories (say B and A, respectively). Another field might be named MONTH DUE. Finding all the birthdays that occurred in a particular month could then be accomplished by a two field search.

Before Starting

The first time the RUN command is issued, the program POKES several machine language routines into memory. It then erases those POKE commands from the BASIC program in order to save memory. Attempting to copy the program after it has been started should thus be avoided.

Also, using the RUN command will cause any file currently in memory to be erased. Do *not* use RUN to restart the program once you have any part of a file stored in memory. Always use the directive GOTO 15 to restart the program without loss of data. (You might want to place such a statement under control of a softkey.)

The program uses the black pen of the printer. Make sure you have a good pen in that position.

When selecting from the menu, remember that

Program *File Maintenance.*

```
 10:GOSUB 995:LOCK
    :TEXT :CSIZE 1
    :COLOR 0
 15:CLS :WAIT 0:D=
    9:H=0:RESTORE
    :FOR I=1TO D:
    READ I$(0):
    PRINT I$(0)
 20:FOR J=1TO 30:I
    $=INKEY$ :IF I
    $<>""LET J=60
 25:NEXT J
 30:IF I$=""THEN 5
    0
 35:FOR J=1TO D:IF
    MID$ ("NOI1AVP
    FC",J,1)=I$LET
    H=J:J=D
 40:NEXT J
 45:IF HCLS :ON H
    GOSUB 100,200,
    300,400,500,70
    0,700,800,900:
    GOTO 15
 50:NEXT I:GOTO 15
 55:DATA "N - NEW
    FILE","O - OUT
    PUT/TAPE","I -
     INPUT/TAPE"
 60:DATA "1 - FILE
     PARAMETERS","
    A - ADD RECORD
    ","P - PRINT R
    ECORD"
 65:DATA "V - VIEW
     RECORD","F -
    FIND RECORD(S)
    ","C - CHANGE
    RECORD"
100:IF LEN N$=0
    THEN 115
105:BEEP 3:INPUT "
    SURE? ";I$
110:IF I$<>"Y"
    RETURN
115:GOSUB 995
120:INPUT "NEW FIL
    E NAME: ";N$
125:IF LEN N$<1OR
    LEN N$>15THEN
    120
130:INPUT "FIELDS
    PER RECORD: ";
    K
135:IF K<1OR K>12
    THEN 130
140:FOR I=1TO K
145:CLS :PRINT "FI
    ELD";I;" NAME:
    ":CURSOR 14:
    INPUT "";@$(I+
    14)
150:IF LEN @$(I+14
    )<1OR LEN @$(I
    +14)>15THEN 14
    5
155:CLS :INPUT "HO
    W MANY BYTES:
    ";@(I+14)
160:IF @(I+14)<1OR
    @(I+14)>40THEN
    155
165:L=L+@(I+14):
    NEXT I
170:GOSUB 190
175:PRINT "MAXIMUM
     RECORDS";N:
    BEEP 15:RETURN
190:N=INT ((STATUS
    3-STATUS 2-63)
    /L):RETURN
200:IF LEN N$=0
    GOSUB 960:
    RETURN
205:INPUT "TAPE RE
    ADY? ";I$
210:IF I$<>"Y"
    RETURN
215:PRINT #N$;@$(*
    ),@(*)
220:I$=N$+"*":
    CSAVE MI$;
    STATUS 2,
    STATUS 2+(M*L)
    :RETURN
300:IF LEN N$=0
    THEN 315
305:BEEP 3:INPUT "
    SURE? ";I$
310:IF I$<>"Y"
    RETURN
315:GOSUB 995:
    INPUT "FILE NA
    ME: ";N$
320:INPUT #N$;@$(*
    ),@(*)
325:GOSUB 190
330:IF N<MPRINT "F
    ILE SPACE EXCE
    EDED!":BEEP 15
    :GOSUB 995:
    RETURN
335:I$=N$+"*":
    CLOAD MI$;
    STATUS 2:
    RETURN
400:IF LEN N$=0
    GOSUB 960:
    RETURN
405:LF 5:LPRINT "F
    ILENAME: ";N$
410:LPRINT "MAXIMU
    M RECORDS:";N
415:LPRINT "RECORD
    S USED:";M
420:LPRINT "BYTES
    PER RECORD:";L
    :LF 1
425:FOR I=1TO K:
    LPRINT "FIELD"
    ;I;" ";@$(I+14
    );TAB 28;USING
    "###";"BYTES";
    @(I+14):USING
430:NEXT I:LF 6:
    RETURN
500:IF LEN N$=0
    GOSUB 960:
    RETURN
505:IF M=NPRINT "O
    UT OF RECORDS!
    ":BEEP 15:
    RETURN
510:F=STATUS 2+(M*
    L):FOR I=1TO K
    :G=I+14
515:CLS :I$(0)="":
    PRINT @$(G);":
    ":CURSOR LEN @
    $(G)+2:INPUT "
    ";I$(0)
520:IF LEN I$(0)>@
    (G)GOSUB 985:
    GOTO 515
525:GOSUB 955:F=F+
    @(G):NEXT I:
    CLS :M=M+1:
    PRINT "ADD REC
    ORD";M:BEEP 15
    :RETURN
600:LF 1:LPRINT "R
    ECORD:";I:LF 1
605:G=STATUS 2+((I
    -1)*L):FOR F=1
    TO K:E=F+14
610:CALL STATUS 3-
    5,@(E):CALL
    STATUS 3-14,G:
    J=STATUS 3+7:
    CALL STATUS 3-
    35,J:G=G+@(E)
615:IF I$<>"V"
    LPRINT STR$ (F
    );" ";I$(0):
    NEXT F:RETURN
620:J=1
625:PRINT STR$ (F)
    ;" ";MID$ (I$(
    0),J,24)
630:IF ASC INKEY$
    =8LET J=J+1:
    GOTO 650
635:IF ASC INKEY$
    =12LET J=J-1:
    GOTO 650
640:IF ASC INKEY$
    <>13THEN 630
645:CLS :NEXT F:
    RETURN
650:IF J=0LET J=1
655:IF J>LEN I$(0)
    LET J=LEN I$(0
    )
660:GOTO 625
700:IF M=0GOSUB 96
    0:RETURN
705:I=0:INPUT "REC
    ORD NUMBER: ";
    I
710:IF I<1OR I>M
    THEN RETURN
715:IF I$="V"GOSUB
    605:RETURN
720:GOSUB 600:LF 5
    :RETURN
800:IF M=0GOSUB 96
    0:RETURN
805:CLS :H=0:INPUT
    "HOW MANY FIEL
    D SEARCH? ";H
810:IF H<1OR H>4OR
    H>KTHEN RETURN
815:RESTORE 885:
    FOR I=1TO H:
    READ I$:J$=
    STR$ I
820:CLS :@(I)=0:
    CURSOR :PRINT
    J$;I$:CURSOR 4
    :INPUT "FIELD
    NUMBER: ";@(I)
825:IF @(I)<1OR @(
    I))KTHEN 820
830:CLS :I$(0)="":
    CURSOR :PRINT
    "SEARCH";@(I);
    " FOR: ":
    CURSOR 14:
     INPUT "",I$(0)
835:IF LEN I$(0)>1
     6OR LEN I$(0)>
     @(@(I)+14)
     GOSUB 985:GOTO
     830
840:IF LEN I$(0)<1
     THEN 830
845:@$(I)=I$(0):
     NEXT I
850:FOR I=1TO M:
     FOR J=1TO H
855:CLS :PRINT "RE
     CORD";I
860:G=@(J):GOSUB 9
     90
865:I$="":FOR E=1
     TO LEN @$(J).I
     $=I$+CHR$ PEEK
     F:F=F+1:NEXT E
870:IF MID$ (I$,1,
     E-1)<>@$(J)
     NEXT I:RETURN
875:NEXT J
880:GOSUB 600:NEXT
     I:RETURN
885:DATA "st","nd"
     ,"rd","th"
900:IF M=0GOSUB 96
     0:RETURN
905:I=0.INPUT "REC
     ORD NUMBER. ";
     I
910:IF I<1OR I>M
     THEN RETURN
915:G=0:INPUT "FIE
     LD NUMBER: ";G
920:IF G<1OR G>K
     THEN 915
925:I$(0)="":INPUT
     "CHANGE TO: ";
     I$(0)
930:IF LEN I$(0)>@
     (G+14)GOSUB 98
     5:GOTO 925
935:GOSUB 990:G=G+
     14:GOSUB 955:
     RETURN
955:CALL STATUS 3-
     5,@(G):CALL
     STATUS 3-14,F:
     J=STATUS 3+7:
     CALL STATUS 3-
     63,J:RETURN
960:PRINT "FILE EM
     PTY!".BEEP 15:
     RETURN
985:CLS :PRINT "TO
     O LONG!":BEEP
     15:RETURN
990:F=STATUS 2+((I
     -1)*L)
992:IF G=1RETURN
994:FOR E=1TO G-1.
     F=F+@(E+14).
     NEXT E:RETURN
995:CLEAR :DIM I$(
     0)*41
996:POKE STATUS 3-
     35,&A5,&76,&D0
     ,&DF,&2A,&A5,&
     76,&D1,&18,&A5
     ,&76,&D2,&1A,&
     55,&41,&88
997:POKE STATUS 3-
     19,4,&B5,0,&41
     ,&9A,&84,&AE,&
     76,&D1,4,&AE,&
     76,&D2,&9A,4,&
     AE,&76,&D0,&9A
998:POKE STATUS 3-
     63,&A5,&76,&D0
     ,&DF,&2A,&A5,&
     76,&D1,&18,&A5
     ,&76,&D2,&1A,&
     45
999:POKE STATUS 3-
     49,&B7,0,&8B,4
     ,&51,&88,8,&9A
     ,&B5,32,&51,&8
     8,3,&9A
1000:POKE STATUS
      2,&84,&28,4,
      &2A,&BE,&D2,
      &EA,0,&A5,&7
      8,&A6,8,&A5
1005:POKE STATUS
      2+13,&78,&A7
      ,&A,&46,&46,
      &46,&FB,&A4,
      &AE,&76,&D0,
      &9A
1010:POKE STATUS
      2+25,&84,&AE
      ,&78,&67,4,&
      AE,&78,&68,&
      9A
1015:I=996.CALL
      STATUS 2,I.
      IF PEEK &76D
      0=&BSTOP
1020:X=STATUS 2-I
      -27
1025:A=STATUS 2-I
      -X
1030:CALL STATUS
      2+25,A
1035:POKE I-1,&3A
      ,&F1,&99,&D,
      3,&E4,&14,&F
      1,&AB,&53,&2
      0,&54,&4F
1040:POKE I+12,&4
      D,&42,&41,&4
      3,&4B,&20,&4
      6,&4D,&20,&3
      8,&2F,&38,&3
      3,&D,&FF
1045:POKE I-15,&1
      1
1050:CLEAR :DIM I
      $(0)*41:
      RETURN
STATUS 1        3566
```

only a single keystroke is needed. All other entries, such as data, must be terminated by pressing the ENTER key. For instance, to establish a new file just depress the N key when the menu is cycling. To enter a new file name, type in the desired name followed by the ENTER key.

At the Beginning

Load the program into the computer, place the PC into the RUN mode and type RUN. A continuously cycling menu will be displayed containing the following items:

N - NEW FILE
O - OUTPUT/TAPE
I - INPUT/TAPE
1 - FILE PARAMETERS
A - ADD RECORD
V - VIEW RECORD
P - PRINT RECORD
F - FIND RECORD(S)
C - CHANGE RECORD

An Illustration

To show how the program may be applied, a file for holding birthdays and anniversaries will be created. It will be named AUTOCALENDAR. The seven fields will be titled and sized as follows:

	Name	Length (Bytes)
Field 1:	TYPE	1
Field 2:	MONTH DUE	2
Field 3:	NAME	20
Field 4:	ADDRESS	20
Field 5:	CTY ST ZP	23
Field 6:	PHONE	12
Field 7:	NOTE	16

Setting Up A New File

A new file is created using the NEW FILE option. Depress the N key while the menu is cycling and NEW FILE NAME will be displayed. If a file already exists the programs asks SURE? Any response other than Y at this point will return the program to the main menu. Responding Y clears the memory of any previous file. The program will then prompt for a NEW FILE NAME. This procedure permits a user who has inadvertently selected the NEW FILE 'unction to exit without erasing the existing file.

If you desire to follow along with this example, enter the name AUTOCALENDAR for the new file name. The program will then ask FIELDS PER RECORD. Enter the number 7 for this illustration. Remember, the maximum permitted is 12, a value greater than that will not be accepted.)

The PC will then request the name of each field and its length in characters. Respond as outlined previously if you want to follow the example. After he file has been set up, the program will issue a beep, then display the maximum number of records hat may be stored (74 in this example). After this he program will go back to displaying the menu.

'rinting File Parameters

'his option causes the current filename, maximum umber of records that can be stored in the file, the umber of records used, the number of bytes in a ecord, and each field name and its length to be rinted. After outputting this information the rogram reverts to displaying the menu.

If you are following the example procedure, it is a good idea to select this option and verify that you ave set up the file correctly. Depress the number digit key while the menu is cycling to select this unction. If you find that the file is not set up as you lesire, then stop the program with the BREAK key. 'ou would then issue a RUN command to restart he program.

It is important to realize that once defined, you annot alter the file parameters without losing all he data currently in memory. Make sure you have hings set up properly before you start building up a ile of valuable data!

\dding A Record

low that a file has been created and its proper ormat verified, try adding a birthday to it. Depress he A key while the menu is cycling. The first field escription (TYPE:) will appear. Respond B to this ield for birthday. (Remember, field 1 in this pplication is used used to differentiate between nniversaries [A] and birthdays [B].)

The second field description, MONTH DUE, will hen be displayed. Respond with a two-digit nswer. Thus, 01 should be used for January. The alue 12 would be used for December. (If a single igit was used for January, then a single character earch of the MONTH DUE field for the digit 1 ould find both the 1 and the 12 entries. Using two haracters means that a properly specified search ill yield exactly the desired match.)

Continue entering the rest of the information or the record as prompted by the field names. (You might want to use the NOTE field for the actual birthdate or some other relevant information.)

When all the fields in the record have been inputted, the computer will beep. It will then show the record number assigned to that entry before going back to display the menu.

Viewing A Record

To display the contents of a record on the LCD, use this option. It is selected by pressing the V key when in the menu mode. Respond to the prompt for RECORD NUMBER by entering the desired value. (To view the example record just entered, respond with a 1, assuming it is the first entry in your file.)

The number that appears on the lefthand side of the screen while viewing is the field number. This is followed by the name of the field and then its contents. Press ENTER to display each consecutive field within a record.

The LCD can display 24 characters of the fields numbered 1 through 9 and 23 characters of the fields numbered 10 through 12. However, the forward and back arrow keys may be used to scroll the screen right and left when fields exceed these lengths.

When the last field in a record has been displayed, pressing the ENTER key will return the program to the cycling menu.

Printing A Record

Use the PRINT RECORD option, selected by pressing the P key, to list a record on the printer. (To list the example record, respond to the prompt for RECORD NUMBER with a 1.) When the record has been printed, control returns to the menu.

Changing A Record

Depress the C key when in the menu mode to bring up this option. It permits the altering of data within a specified field in a designated record. Respond appropriately to the prompts for RECORD NUMBER and FIELD NUMBER. When the prompt CHANGE TO is displayed, enter the desired data.

If the new information is different in length than the original field entry, the remainder of the field is filled with spaces. To delete the contents of a field, respond to the prompt CHANGE TO prompt by pressing just the ENTER key. This causes the entire field to be filled with spaces.

After a field has been changed, the program returns to the menu mode.

Finding Record(s)

Records in a file may be located using this option. From one to four fields may be searched at a time. After you have inputted several birthdays and anniversaries into the example file, try finding a particular birthday by month and name as follows:

Depress the F key while in the menu mode to bring up the prompt HOW MANY FIELD SEARCH? For purposes of illustration, assume a three-field search on the fields named TYPE, MONTH DUE and NAME.

Now respond to the 1st FIELD NUMBER prompt with a 1 to indicate that the first field (TYPE) is to be checked. Respond to SEARCH 1 FOR with a B for birthday in this example.

Respond to the 2nd FIELD NUMBER prompt with a 2 as the second field (MONTH DUE) is to be examined. Respond to the prompt SEARCH 2 FOR with the month of the birthday you are seeking.

Respond to 3rd FIELD NUMBER with a 3 to indicate that the third field (NAME) is to be inspected. Respond to SEARCH 3 FOR with the name of the person you are seeking.

After this entry the program will begin looking for the record that matches all the parameters requested. As it searches it will display the record number that it is examining. When it finds a record matching all the specifications, it will output it to the printer. When the search has been completed, the program returns to the menu mode.

It should be noted that a field search may be performed in any order. While field numbers 1, 2 and 3 were specified in the example, they could have been requested in the order 3, 1, and then 2.

Saving A File On Tape

Files may be saved on tape. Make sure the tape recorder is properly connected and set to the record mode. Check that the cassette has been advanced beyond the tape leader. Press the O key to bring up the TAPE READY? query. Any response other than Y will terminate the procedure. A response of Y will cause the file to be recorded on tape using the name of the current file. (For instance, AUTOCALENDAR, in the case of the current example.) When the file has been written, program operation reverts to the menu.

Since data files can *not* be verified, it is a good idea to make a second copy of important files on a second tape cassette. Just load in another tape and repeat the tape save process.

Restoring A File From Tape

Files that have been previously stored on tape may, of course, be read back for use by this program. Make sure that the tape unit is properly connected and in the playback mode. Press the I key to bring up the prompt SURE? An answer of Y at this point clears the memory of any previous file. The program will then prompt for the FILE NAME. This procedure provides an exit from inadvertent selection of the INPUT/TAPE option without erasing an existing file.

Inputting the name of a file (such as the example AUTOCALENDAR) causes the program to begin reading the tape. The file names it finds are displayed on the LCD.

A data file is actually saved in two parts by this program. Hence, when it is recovered, it will be processed as one file under the original user-assigned name (AUTOCALENDAR for example) and another file (AUTOCALENDAR*) that has the same name suffixed by an asterisk. The program will automatically process both files. When the procedure has been completed, the display will return to the cycling menu.

Deleting Records

In order to save memory, a delete option was not included in the program. However, if a record that has been created is no longer needed, then alter one of its fields tq signify that it has been deleted. For instance, place the letter D in field 1. When you need a new record, use the FIND RECORD(S) option to locate such a deleted record. Then use the CHANGE RECORD option to edit in your new data.

RABBIT CHASE

David Hergert, 4714 Chickering Ave., Cincinnati, OH 45232 submitted this clever and challenging game. It is designed to run on the Sharp PC-1500 or Radio Shack PC-2 equipped with at least 4K of RAM. It uses the printer to draw graphics at the beginning of the program. Additionally, if you successfully complete a game by concluding the fourth level, another set of graphics is drawn as a reward.

The game itself is played on the LCD. In the game, the player is pictured as a rabbit that is trying to eat all the vegetables in Farmer Brown's garden. Farmer Brown is trying to catch the rabbit with his tractor. The objective is for the rabbit to eat all of the vegetables on all four levels of play, before Farmer Brown catches it. Each level of play is harder than the previous one. (Level four is really tough!)

Pressing the number 4 key causes the rabbit to move to the right. The number 6 key changes its direction to the left. The rabbit has wrap-around capabilities, but the tractor does not. Don't let the tractor catch the rabbit!

If you become totally frustrated and unable to win the game at level four, you can see the final graphics by entering RUN 950. If you want to skip the opening printer graphics, use RUN 20. The normal start is to simply issue a RUN command.

FROM THE WATCH POCKET

Here it is, the end of our third year of publication. As is the custom in the year-end edition, I will take a little extra space for this column. This enables me to review progress in the past year as well as speculate on what may occur in the year ahead.

Not Much Happened

The past year was certainly not a stellar one in terms of PC progress. Hardly anything that might be considered revolutionary was introduced on the hardware side. Most of what might be called real progress boils down to just a few minor model changes.

Perhaps the most important advance came when Sharp was convinced that 2K of RAM was not enough for today's sophisticated PC users. The Sharp PC-1500A is an exact replica of the PC-1500 (right down to the ROM chips) with some more internal RAM. In the new PC-1500A there is 6K of user RAM in the address range &4000 through &57FF. There is another 1K of RAM, apparently intended for ML routines, tucked in the address range &7C00 through &7FFF. Just don't use the first byte of that area as it occassionally gets stuffed with a &0D when the BASIC input buffer overflows. (A software quirk?) Add in the area dedicated to storage of string and numeric variables and you have the 8K of memory Sharp is entitled to advertise. For all practical purposes, however, your BASIC programs must fit in 6K of memory, not the 8K you might assume.

Despite the hopes and prayers of many users, there was very little in the way of peripherals added by the PC manufacturers. Most notably absent was the introduction of any kind of practical external mass storage device. Of course, Sharp may view their 16K RAM module, backed up by batteries, as a viable alternative. Personally, I do not. At about $150 per clip, it is a long way from what I consider low cost mass storage. Of course, you can always spend a half-hour loading 16K of RAM from tape cassette, provided you don't catch a glitch.

Hurray For The Little Guys

Perhaps the most exciting news of the year is being generated by a small firm in Pennsylvania. *Robert Undi* of Usonics, *7901 Oak Hill Drive, Cheltenham, PA 19012*, reports that they have successfully mated a "stringy floppy" data drive to the PC-1500. Priced at about $400.00, this unit gives users the capability of storing and recovering data at a rate that allows 16K to be transferred in about 30 seconds. Supplies of the special drive are said to be somewhat limited. But, if you are looking for something about 50 times faster than audio cassette, you might look into this item.

Usonics also sells the new Brother EP-22 portable typewriter. This is the advanced version of the earlier EP-20. The beauty of this model is that it has a built-in RS-232C interface. Just connect it to your CE-158 RS-232C interface and you have a highly portable external printer. The EP-22 is priced at about $250.00. When not being used as a printer, it can serve as an electronic portable typewriter, complete with 2K of internal, editable, memory! A big cut above the earlier model, and at just a $50.00 premium, this seems like a good buy.

Put Your Program(s) On ROMs?

A growing trend among PC users in businesses is to develop packages of programs that can be used by many employees. Groups of these programs are then "burned" into ROMs for general distribution.

Several firms are now engaged in providing ROM programming services. Out in Indiana, a firm by the name Carleton Micro Systems is providing a complete modular package that can hold 8K or 16K of EPROM. The module is intended primarily for businesses, institutions or software vendors that need 25 or more packages of the same program(s). The firm takes software supplied on cassette or a battery-backed RAM module and converts it to their EPROM units. The modules disable the editing, program listing and store functions of the PC-1500 so that the installed software cannot be copied. In lots of 25, the 16K modules sell for approximately $150.00 each plus a software setup fee.

If you don't need 25 or more copies of a program and you can get by with just 8K of program storage, you might be interested in the module duplication service offered by Atlantic N.E. Marketing, *P.O. Box 921, Marblehead, MA 01945*. Duplication of your own 8K (battery-backed) module starts at about $100.00 for a single unit and declines as the quantity increases.

OEM Action Increasing

The pocket computer market currently seems to be flat to declining at the consumer level, but increasing in the OEM market. In other words, the PC is being repackaged by others to end up with a product that performs a specific function or serves a particular purpose.

Thus, there is now a company that sells a PC-1500 dressed up as a Bond Analyzer. For about $700.00 you can get a system, completely pre-programmed, that calculates just about everything a businessperson would want to know about bond prices, yields, etc. Another firm makes a specialty of providing PC-1500s to banks, all set up to can load an 8K BASIC program in about one minute. (It would take about 10 minutes to load the same amount into a PC-1500.) This means that the tape interface is a practical way to archive programs and data. You can keep your working copies on a bubble, knowing that should something ever happen, you won't have to take half-a-day to restore the information from audio tape cassettes. It may also be a good way for people to distribute programs. By the way, at least when using BASIC, the PC-5000 provides an option of writing programs to external devices, such as via the tape cassette interface, that cannot be listed or edited, only executed.

The optional printer ($395.00 extra) is amazing. In the first place, it is truly quite. It is the first printer I have ever seen on a portable computer that I would not be embarrased to operate in, say, a public library. However, it has its limitations and is not for everyone. It is relatively slow. It is designed to be sheet-fed. A long report means many stops to insert fresh sheets of paper. It requires special thermal paper or the use of special (expensive) ribbons when used with regular paper. If you need to obtain hardcopies of information when you are on the road, the printer is a must. However, if you can wait until you get back to your base of operations in order to get hardcopies, you might be better-advised to select another alternative purchase a separate external printer and drive i via the PC-5000's built-in RS-232 serial interface.

The PC-5000 does not use nickel-cadmiun batteries. It uses a special, sealed, lead-aci package that is surprisingly small. This will powe the unit for about 8 hours of operation if the optional printer is not used. Using the printe considerably reduces the battery life. A warninc light comes on when the battery needs recharging A recharger, supplied with the PC-5000, may be used to continuously power the unit, thereb keeping the battery continuously recharged unti needed for field-portability.

Looks Like A Winner

All-in-all, the PC-5000 looks like a lot of value, at this time, for its $1995.00 list price. While it will surely face stiff competition in the future, the fact that it is one of the first out of the gate will surely be an important factor in the race to gain popular acceptance. It contains a lot of memory, a powerful version of BASIC, the popular Microsoft operating system, and a menu-driven means of integrating a line of optional application packages. Right now it appears to be the lowest-priced unit with such capabilities. Put all that together and it sure looks like a potential winner!

LITTLE EDITOR

H. David Jackson, 126 Smithfield Drive, Endicott, NY 13760, provides this version of a text editor for PC users.

Introduction

LED is the abbreviation I have assigned to a program called the Little EDitor. This program may be used to input data sets and later edit, alter or delete that data as desired. Although it is small, it still contains many of the functions found in editors that run on large mainframe computers. It is somewhat slow because it is written in BASIC. But, what it lacks in speed it makes up for in function. The use of BASIC also allows the user to easily add to, delete or modify its operation.

LED is considered user friendly as commands are entered under function key control rather than having to type in a directive. Only data and certain parametric information, such as data set names, have to be typed in.

This program was developed on a Radio Shack PC-2 with an 8K memory module. A 4K module can be used, but the maximum size of the data set that could be handled will be considerably reduced. The program requires a printer/cassette interface for saving, restoring and printing the data sets.

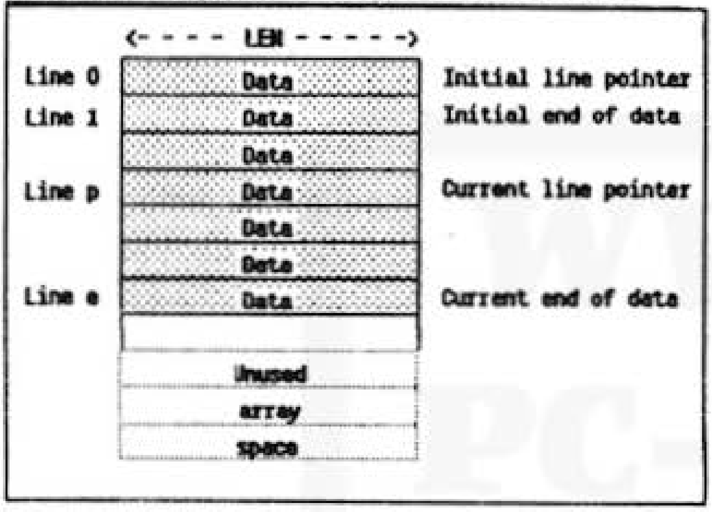
Data Set Size

The data set size is set by the user immediately upon entry to the program. A variable called NUM will be requested. This value is the maximum number of lines that will be entered. The largest value that may be used is 255. LEN will then be requested. This is the maximum length of the lines that will be used. The largest allowable value here is 80. These two variables are used to dimension an array that stores the data. Plan your data set size carefully as the array cannot be re-dimensioned without losing data already stored. Also, the entire array is saved on tape when requested, no matter how much or little data has actually been entered. This takes some time if maximum limits have been chosen and wastes time if the space is not needed. Of course, it is not necessary to fill all lines reserved. The array can contain unused space.

The size of the array (NUM times LEN) cannot exceed the amount of storage available. If it does, an error message will be displayed. Use the MEM command prior to running the program to see how much memory is available. With an 8K module there will be approximately 7500 bytes, 3400 bytes with a 4K module, provided that memory is not allocated for other purposes.

Regardless of the array size, internal pointers to the current line being entered or edited, as well as the last line in the data set, are maintained by the LED program. These are initially set to zero and one respectively. Using the end of data (EOD) allows the line pointer to be more easily set to selected locations without having to traverse large amounts of unused lines. Thus, the initialized data set is two lines long. Additional lines may be added as needed up to the extent of the array size (NUM) that was originally specified.

Diagram *Array and Data Space Allocations.*



Data Display and Modes

The amount of data that is seen at one time is limited to the 26 characters of the display screen. The length of a line, however, can be up to the LEN orginally specified. In order to accomodate this, two types of displays are used. The first is the input display. This is used when the program has been placed in the input mode. It is denoted by the presence of a question mark (?) at the left of the screen. In this case, the normal responses and editing associated with the use of the INPUT statement in BASIC apply. The display scrolls left one character at a time when the right end of the screen is reached.

The second type of display is associated with the edit mode. It is denoted by a flashing cursor. While in this mode, all of the function keys that will be described later are operational. Key usage in this mode is a little different. Various modes are used to tell the program what to do. The current mode(s) are indicated by the cursor. These modes are summarized as follows:

SHIFT MODE - This mode is similar to the regular shift key function used in the INPUT section. It, however, also modifies the commands associated with some of the cursor movement keys. This mode is entered by pressing the SHIFT key. If the current line is null (contains no data), then the program will enter the input mode (a ? appears on the left of the screen). This allows data to be entered faster than when in the edit mode. The program will remain in this mode for each line entered until a non-null line is found or the ENT key is pressed without entering any data. In either case, the program will revert back to the edit mode.

ALTERNATE MODE - This mode also changes the command that is associated with function and cursor movement keys as described later. This mode is set by pressing the Reserve mode change (up and down solid triangle) key.

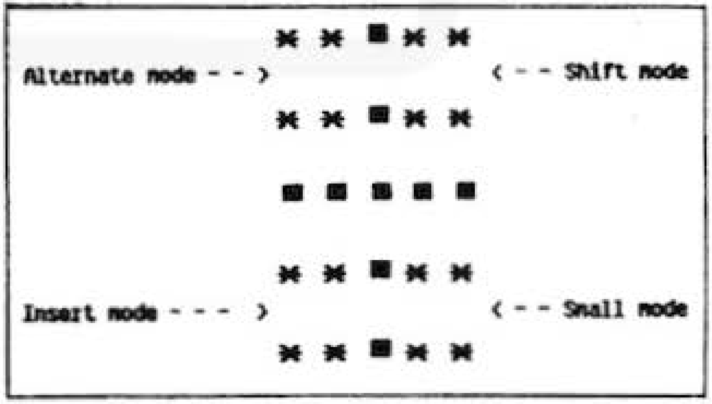
(The above two modes are automatically reset after another key is pressed. That is, they are only in effect for one key stroke.)

INSERT MODE - This mode indicates that characters are to be inserted at the cursor position instead of overlaying those that are already there. This mode is toggled by pressing the MODE key.

SMALL MODE - As the name implies, characters that are overlaid or inserted will be in lower case. This mode is toggled by pressing the SML key.

The cursor is used to indicate the current mode status. This is done by placing small blocks in each corner of the cursor. If just a flashing character (or lone underline when the position is blank) is observed, then none of the above modes are set. The accompanying diagram shows which corner blocks are active for the various modes. Note that more than one mode may be set at a time.

Diagram *Cursor Mode Indicators.*



In addition, when in the edit mode, the cursor and display positioning is slightly different than that used by the input mode. When the cursor leaves the right side of the display in the edit mode, it will be positioned to the middle of the display instead of just scrolling left one. This gives the user a more complete view of the line being edited and allows one to see data on both sides of the cursor. The same occurs, when moving to the left, unless the cursor is in the first position.

Normal Function Keys

The following functions or commands are executed by pressing the specified key when editing in the normal mode. A beep indicates that the depression of a function key (F1 - F6) has been recognized by the program.

F1 - LOAD: This command will load a data set that was previously stored on tape. It is necessary to position the tape to the proper spot. Then enter the name of the data set that is to be recovered. The program restores the array parameters that were saved along with the data. Pressing the ENT key by itself causes the command to be ignored.

F2 - ADD: This command will add a single null line immediately after the line on which the cursor currently resides. If you listen carefully, an audio click may be discerned, indicating that one line has been added.

F3 - CLEAR MARKS: This command will clear any marks that may have been placed on selected line(s) of data.

F4 - MARK DATA: The first press of this key will *mark* the line at the cursor position for later reference. This line will be highlighted by reverse video. A second press will mark a group of lines, all of which will be denoted by reverse video. The group comprises all lines between and including the marked lines. A marked line or group of lines may be moved, deleted or printed. Marks may also be used to limit changes to a selected line or group.

F5 - MOVE: Pressing this key causes the marked area to be moved after the line on which the cursor is currently located. The marked data is not erased. Thus, this directive may be used to duplicate information at various locations. (To erase data see alternate F2.)

F6 - FIND DATA: This key is used to indicate that a word or phrase is to be searched for in the data beyond the current cursor or in a marked area. If located, that line will become the current line in the display. If the search is unsuccessful, the line pointer remains the same.

Alternate Function Keys

The following functions will be executed upon pressing the specified key while editing in the alternate mode.

AF1 - SAVE: This command will save the data set and associated parameters. You will have to position the tape, make sure you have set the tape unit to the record mode, and enter a file name for the data set. Pressing the ENT key without other input will cause the command to be ignored.

AF2 - DELETE: This command causes the line that the cursor is on or a marked group of lines to be removed from the data set. A low tone is issued for each line that is deleted.

AF3 - DUPLICATE: This key causes the line that the cursor is on to be duplicated immediately after the line.

AF4 - Not used.

AF5 - PRINT: The entire data set or a marked group of lines is printed. Prior to outputting of the data, the program will prompt for the CSIZE desired by the operator.

AF6 - CHANGE DATA: A global change function. The system prompts for *from* data and *to* data. All cases of the former following the cursor to the end of the data set or within a marked group are modified to the latter. A null *from* entry causes the *to* data to be inserted at the start of each line. A null *to* entry will cause the *from* data to be removed.

Shifted Function Keys

If the shift mode is on, the function keys F1 - F6 will react as though in the input mode. That is, the characters !, ", #, $, % and & will be entered. The shift key is also used to enter @, >, <, ?, ;, : and comma as indicated on the keyboard.

The shift key is additionally used to activate the CLR key. This combination causes the data array to be re-initialized.

If the OFF key is pressed while in the shift mode, the program is terminated.

Cursor Control Keys

The cursor control keys perform in the following manner:

The right arrow causes the cursor to move one character position to the right.

Shift right arrow causes the character beneath the cursor to be deleted. The cursor remains at the same position.

The left arrow causes the cursor to move one character position to the left.

Shift left arrow causes the character to the left of the cursor to be eliminated. The cursor also moves left one position.

The up arrow causes the line pointer to move one line closer to the start of the data set (-1).

Alternate up arrow causes the line pointer to be moved five lines closer to the start of the data set (-5).

Shift up arrow causes the line pointer to be set to the start of the data set. (Note that this cannot be performed on a null line as the shift will cause the program to go into the input mode.)

The down arrow moves the line pointer one line closer to the end of the data set (+1).

Alternate down arrow causes the line pointer to move five lines closer to the end of data (+5).

Shift down arrow sets the line pointer to the end of data. (Again, this operation cannot be used while on a null line.)

Additional Key Capabilities

The following keys also assist during editing operations:

The RCL key adds five lines after the current line.

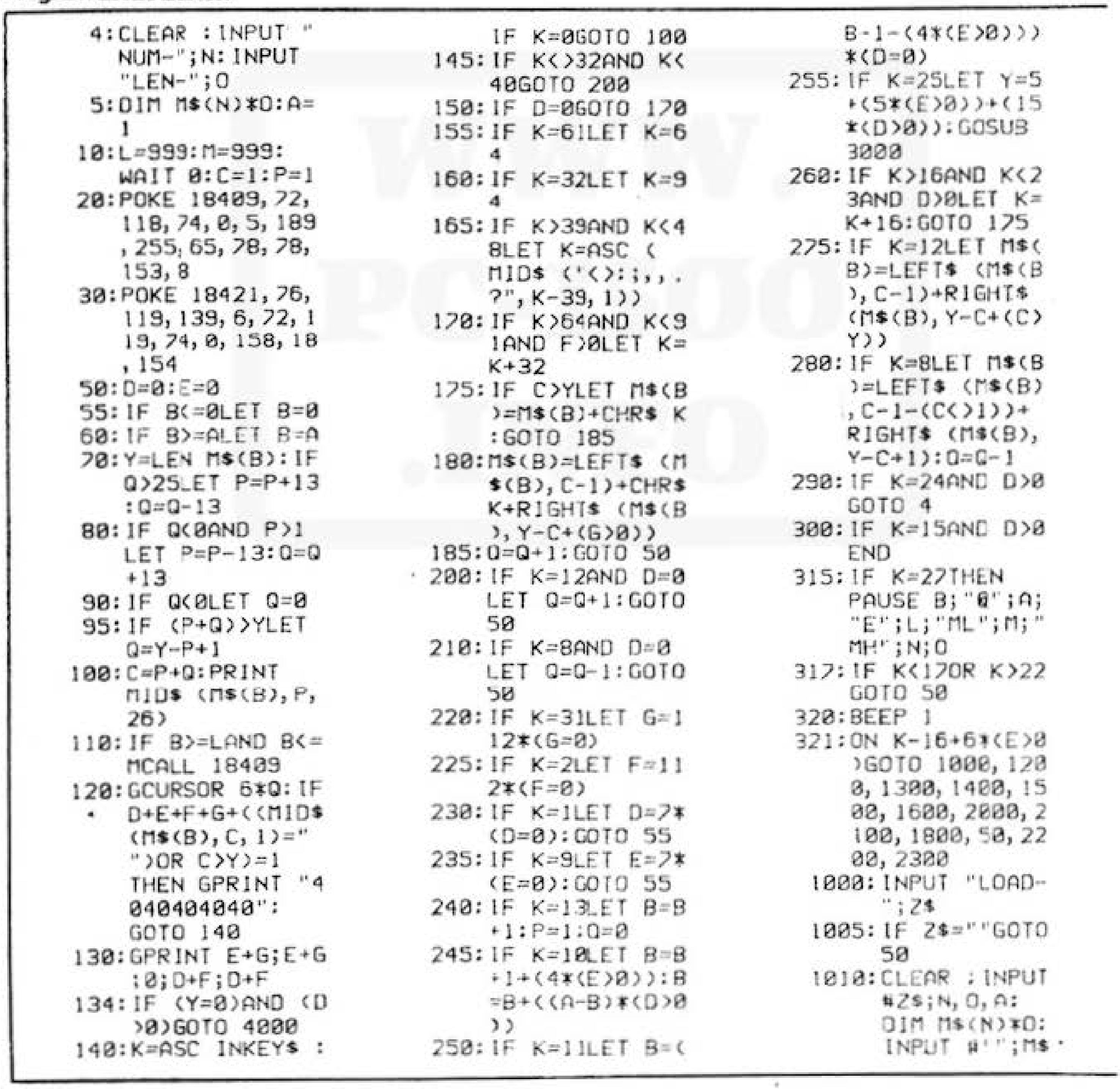
Alternate RCL adds 10 lines.

Shift RCL adds 20 lines.

A fast audio click announces the addition of each line.

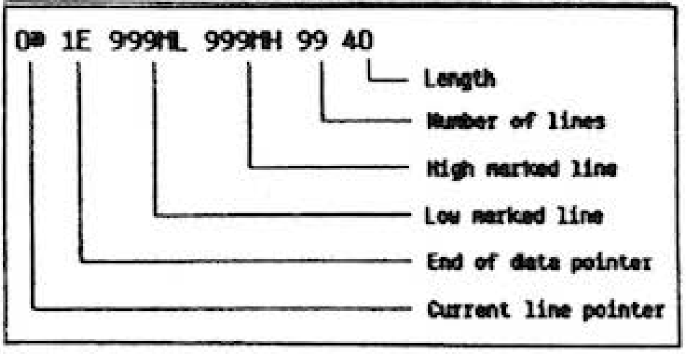
The DEF key displays the status of all the important pointers. This display uses the format illustrated in the accompanying diagram. (The diagram assumes that the data set initially had an array size whereby NUM=99 and LEN=40. The value 999 in the marked lines positions indicates that no lines are marked.)

Program *Little Editor.*



```
   4:CLEAR :INPUT "
     NUM-";N:INPUT
     "LEN-";O
   5:DIM M$(N)*O:A=
     1
  10:L=999:M=999:
     WAIT 0:C=1:P=1
  20:POKE 18409,72,
     118,74,0,5,189
     ,255,65,78,78,
     153,8
  30:POKE 18421,76,
     119,139,6,72,1
     19,74,0,158,18
     ,154
  50:D=0:E=0
  55:IF B<=0LET B=0
  60:IF B>=ALET B=A
  70:Y=LEN M$(B):IF
     Q>25LET P=P+13
     :Q=Q-13
  80:IF Q<0AND P>1
     LET P=P-13:Q=Q
     +13
  90:IF Q<0LET Q=0
  95:IF (P+Q)>YLET
     Q=Y-P+1
 100:C=P+Q:PRINT
     MID$ (M$(B),P,
     26)
 110:IF B>=LAND B<=
     MCALL 18409
 120:GCURSOR 6*Q:IF
     D+E+F+G+((MID$
     (M$(B),C,1)="
     ")OR C>Y)=1
     THEN GPRINT "4
     040404040":
     GOTO 140
 130:GPRINT E+G;E+G
     ;0;D+F;D+F
 134:IF (Y=0)AND (D
     >0)GOTO 4000
 140:K=ASC INKEY$ :
     IF K=0GOTO 100
 145:IF K<>32AND K<
     40GOTO 200
 150:IF D=0GOTO 170
 155:IF K=61LET K=6
     4
 160:IF K=32LET K=9
     4
 165:IF K>39AND K<4
     8LET K=ASC (
     MID$ ("<>:;,,.
     ?",K-39,1))
 170:IF K>64AND K<9
     1AND F>0LET K=
     K+32
 175:IF C>YLET M$(B
     )=M$(B)+CHR$ K
     :GOTO 185
 180:M$(B)=LEFT$ (M
     $(B),C-1)+CHR$
     K+RIGHT$ (M$(B
     ),Y-C+(G>0))
 185:Q=Q+1:GOTO 50
 200:IF K=12AND D=0
     LET Q=Q+1:GOTO
     50
 210:IF K=8AND D=0
     LET Q=Q-1:GOTO
     50
 220:IF K=31LET G=1
     12*(G=0)
 225:IF K=2LET F=11
     2*(F=0)
 230:IF K=1LET D=7*
     (D=0):GOTO 55
 235:IF K=9LET E=7*
     (E=0):GOTO 55
 240:IF K=13LET B=B
     +1:P=1:Q=0
 245:IF K=10LET B=B
     +1+(4*(E>0)):B
     =B+((A-B)*(D>0
     ))
 250:IF K=11LET B=(
     B-1-(4*(E>0)))
     *(D=0)
 255:IF K=25LET Y=5
     +(5*(E>0))+(15
     *(D>0)):GOSUB
     3000
 260:IF K>16AND K<2
     3AND D>0LET K=
     K+16:GOTO 175
 275:IF K=12LET M$(
     B)=LEFT$ (M$(B
     ),C-1)+RIGHT$
     (M$(B),Y-C+(C>
     Y))
 280:IF K=8LET M$(B
     )=LEFT$ (M$(B)
     ,C-1-(C<>1))+
     RIGHT$ (M$(B),
     Y-C+1):Q=Q-1
 290:IF K=24AND D>0
     GOTO 4
 300:IF K=15AND D>0
     END
 315:IF K=27THEN
     PAUSE B;"0";A;
     "E";L;"ML";M;"
     MH";N;O
 317:IF K<17OR K>22
     GOTO 50
 320:BEEP 1
 321:ON K-16+6*(E>0
     )GOTO 1000,120
     0,1300,1400,15
     00,1600,2000,2
     100,1800,50,22
     00,2300
1000:INPUT "LOAD-
     ";Z$
1005:IF Z$=""GOTO
     50
1010:CLEAR :INPUT
     #Z$;N,O,A:
     DIM M$(N)*O:
     INPUT #"";M$
```

Diagram *Status Indicators for the LED Program.*



Error Messages

Normal system errors, such as exceeding storage capabilities, are announced by regular system error messages.

Errors associated with the editor itself cause five beeps to be emitted. This occurs if an attempt is made to add more lines than are defined for the array or move an unmarked data item. (Note that moving data may cause the array limits to be exceeded.)

```
      (*):GOTO 10
1200:GOSUB 1850:
      GOTO 50
1300:GOSUB 2500:
      GOTO 50
1400:M=B:IF L=999
      LET L=B
1410:IF M<LLET Z=
      M:M=L:L=Z
1420:GOTO 50
1500:IF L=999THEN
      BEEP 5:GOTO
      50
1505:Y=M-L+1:
      GOSUB 3000:
      IF A+Y>NTHEN
      RETURN
1510:BEEP 2,100,3
      00:IF L>BLET
      L=L+Y:M=M+Y
1520:FOR Z=1TO Y:
      M$(Z+B)=M$(Z
      +L-1):NEXT Z
      :GOTO 50
1600:X=B:Y=A:
      GOSUB 1900:T
      $="FIND-":
      GOSUB 1950:V
      =0:GOSUB 196
      0:IF TLET B=
      Z:GOTO 50
1610:PAUSE "NOT F
      OUND":GOTO 5
      0
1800:GOSUB 1850:M
      $(B+1)=M$(B)
      :GOTO 50
1850:Y=1:GOSUB 30
      00:RETURN
1900:IF L<999LET
      X=L:Y=M:L=99
      9:M=L
1910:RETURN
1950:W$="":PRINT
      T$;:INPUT ""
      ;W$
1955:U=LEN W$:
      RETURN
1960:T=0:FOR Z=X
      TO Y
1970:V=V+1:IF
      MID$ (M$(Z),
      V,U)=W$LET T
      =1:RETURN
1980:IF V<=(LEN M
      $(Z)-U+1)
      GOTO 1970
1990:V=0:NEXT Z:
      RETURN
2000:INPUT "SAVE-
      ";Z$
2005:IF Z$=""GOTO
      50
2010:PRINT #Z$;N,
      O,A:PRINT #"
      ";M$(*):GOTO
      50
2100:X=B:Y=B:
      GOSUB 1900:
      FOR Z=YTO A:
      M$(Z-Y+X)=M$
      (Z+1):NEXT Z
      :A=A-Y+X-1
2105:FOR Z=1TO Y-
      X+1:BEEP 1,2
      55,100:NEXT
      Z:GOTO 50
2110:FOR Z=YTO A-
      1:M$(Z-Y+X)=
      M$(Z+1):NEXT
      Z:GOTO 50
2200:X=0:Y=A:
      GOSUB 1900
2210:INPUT "SIZE-
      ";T:CSIZE T:
      FOR Z=XTO Y:
      LPRINT M$(Z)
      :NEXT Z:GOTO
      50
2300:X=B:Y=A:
      GOSUB 1900:T
      $="FROM-":
      GOSUB 1950
2310:T$="":PRINT
      "TO-";:INPUT
      "";T$
2320:IF U=0THEN
      FOR Z=XTO Y:
      M$(Z)=T$+M$(
      Z):NEXT Z:
      GOTO 50
2330:V=0:GOSUB 19
      60
2340:IF TLET X=Z:
      M$(Z)=LEFT$
      (M$(Z),V-1)+
      T$+RIGHT$ (M
      $(Z),LEN M$(
      Z)-V-U+1):V=
      V+LEN T$+1:
      GOTO 2330
2350:GOTO 50
2500:L=999:M=L:
      RETURN
3000:IF A+Y>NBEEP
      5:RETURN
3010:FOR Z=ATO B+
      1STEP -1:M$(
      Z+Y)=M$(Z):
      NEXT Z:FOR Z
      =B+1TO B+Y:M
      $(Z)="":BEEP
      1,1,1
3020:NEXT Z:A=A+Y
      :RETURN
4000:INPUT M$(B)
4010:D=D*(M$(B)<>
      ""):B=B+(D>0
      ):GOTO 55
```

Notes for Programmers

The following information is provided for those who may wish to add to, modify or remove functions.

Lines	Description
4 - 10	Initialize variables, array.
20 - 30	Set up inverse display routine.
50	Reset shift/alternate switches.
55 - 60	Adjust current line number if outside array bounds.
70 - 90	Determine length of current line, adjust display partition and cursor position if cursor is off the screen.
95	Adjust cursor position to end of line if higher than line length.
100	Determine cursor position, display current partition of current line.
110	If current line is within marked limited, display partition in reverse video.
120 - 130	Display editor cursor at screen cursor location.
134	If null line and shift mode go to input routine.
140	Get key data. Repeat display if no key pressed.
145	Determine if a function key was pressed.
150 - 165	If in shift mode, adjust key data to shifted value.
170	Adjust key data if small mode and alpha key.
175	If cursor is at end of line, add character to end of line.
180	Otherwise, overlay or insert character at cursor position depending on status of mode switch.
185	Increment screen cursor position and go display again.
200	Adjust cursor if unshifted right arrow.
210	Adjust cursor if unshifted left arrow.
220	Adjust insert switch if MODE pressed.
225	Adjust small switch if SML pressed.
230	Adjust shift switch if SHIFT pressed.
235	Adjust alternate switch if up/down triangle pressed.
240	Adjust line number, partition number and screen cursor if ENT is pressed.
245	Adjust line pointer if down arrow pressed.
250	Adjust line pointer if up arrow pressed.
255	Add multiple lines to data set if RCL is pressed
260	If shifted function key, adjust key data and go insert or overlay data in the current line.
275	Delete the character at the cursor if shifted right arrow key.
280	Delete character to left of cursor if shifted left arrow key.
290	Re-initialize if shift and CLR.
300	End if shift and OFF.
317 - 321	Go to appropriate function key routine.
1000 - 1010	Load data routine.
1200	Add a line routine.
1300	Clear marked data routine.
1400 - 1420	Mark a line routine.
1500 - 1520	Move data routine.
1600 - 1610	Find data routine.
1800	Duplicate a line routine.
1850	Add a single line subroutine.
1900 - 1910	Set limits subroutine.
1950 - 1955	Input parameters subroutine.
1960 - 1990	Find data subroutine.
2000 - 2010	Save data routine.
2100 - 2110	Delete a line or group routine.
2200 - 2210	Print data routine.
2300 - 2350	Change data routine.
2500	Reset marks subroutine.
3000 - 3020	Add lines subroutines.
4000 - 4010	Input data subroutine.

Variable	Description
A	End of data pointer.
B	Current line pointer.
C	Line cursor position.
D	Shift switch.
E	Alternate switch.
F	Small switch.
G	Insert switch.
K	Key data.
L	Low mark.
M	High mark.
N	NUM.
O	LEN.
P	Screen partition.
Q	Screen cursor position.
V	Line length.
M$	Data array.

INSIDE THE PC-1500

Norlin Rober, 407 North 1st Ave., Marshalltown, IA 50158 is back with lots of "inside" information for users of the Sharp PC-1500. (Virtually all of this information also applies to the Radio Shack PC-2.)

In this issue Norlin provides a recap of information relating to the use of RAM. In coming issues he will provide detailed information on the ROM. If you are a long time subscriber to *PCN* you are well acquainted with the type of valuable information Norlin is able to glean. If you are a new subscriber, we are sure you will marvel at the wealth of information served up by our long time PC *Super Sleuth!*

PC-1500 SYSTEM RAM

Addresses 7600-7C00 contain the Display Buffer, Input Buffer, fixed variables, and various other RAM required by the system.

Because of incomplete decoding, the memory located 7600-77FF may also be addressed as 7000-71FF, 7200-73FF, or 7400-75FF.

In the PC-1500, the memory located 7800-7BFF is also addressable as 7C00-7FFF. (In some versions of the PC-1500, this will be recall-only.)

In the PC-1500A, the area 7C00-7FFFF contains RAM, intended for storage of machine-language programs.

DISPLAY BUFFER

Each dot of the LCD is turned on by the presence of a '1' bit in a corresponding location in the Display Buffer. The upper four dots (U) in a column of the LCD are determined by one nybble, and the lower three dots (L) by another, as tabulated below.

Memory Address:	Columns controlled Lo nybble:	 Hi nybble:	Memory Address:	Columns controlled Lo nybble:	 Hi nybble:
7600	0 U	78 U	7700	39 U	117 U
7601	0 L	78 L	7701	39 L	117 L
7602	1 U	79 U	7702	40 U	118 U
7603	1 L	79 L	7703	40 L	118 L
7604	2 U	80 U	7704	41 U	119 U
7605	2 L	80 L	7705	41 L	119 L
. . .			. . .		
764C	38 U	116 U	774C	77 U	155 U
764D	38 L	116 L	774D	77 L	155 L

Display annunciators are set by bits as follows:

```
764E Misc:  01, BUSY; 02, SHIFT; 04, Katakana characters; 08, SMALL;
     10, III; 20, II; 40, I; 80, DEF
764F Modes:  01, DE; 02, G; 04, RAD; 10, RESERVE; 20, PRO; 40, RUN
```

774E Unused (By the PC-1500 or CE-150).
774F Unused

FIXED STRING VARIABLES

7650-5F	E$	76B0-BF	K$	7750-5F	P$	77B0-BF	V$
7660-6F	F$	76C0-CF	L$	7760-6F	Q$	77C0-CF	W$
7670-7F	G$	76D0-DF	M$	7770-7F	R$	77D0-DF	X$
7680-8F	H$	76E0-EF	N$	7780-8F	S$	77E0-EF	Y$
7690-9F	I$	76F0-FF	O$	7790-9F	T$	77F0-FF	Z$
76A0-AF	J$			77A0-AF	U$		

SYSTEM RAM

7800-784F comprise the stack used by the CPU. Storage into this stack begins at 784F, continuing in reverse direction through memory.

7850 Unused
7851 Unused
7852 Unused
7853 Unused
7854 Unused
7855 Unused
7856 Unused
7857 Unused
7858 Unused
7859 Unused
785A Unused
785B High byte, address of external character input routine
785C Low byte, external character input routine
785D If 00, Katakana displayed; 80, displayed and printed; FF, neither
785E High byte, location of table of Katakana character codes
785F Unused
7860 High byte, START OF ROM in Module; FF if no Module present.
7861 High byte, START OF BASIC program in ROM Module; FF if no Module.
7862 Low byte, START OF BASIC program in Module; FF if no Module.
7863 High byte, START OF RAM
7864 High byte, TOP OF RAM
7865 High byte, START OF BASIC program in RAM
7866 Low byte, START OF BASIC program in RAM
7867 High byte, END OF BASIC program (stop byte address)
7868 Low byte, END OF BASIC program
7869 High byte, START OF EDIT
786A Low byte, START OF EDIT
786B Flags: 01, BEEP OFF; 80, RMT ON
786C Unused
786D Unused
786E Unused
786F Unused
7870 Unused
7871 WAIT setting: 01, WAIT; 02, WAIT nnnn; 03, WAIT 0
7872 High byte, WAIT time counter
7873 Low byte, WAIT time counter
7874 Flags: 01, Cursor enabled; 80, display currently saved 7B10-7BAB
7875 CURSOR POINTER (current display column number)
7876 Character position number in display, with INPUT statement
7877 Complement of number of display positions left for INPUT prompt
7878 Specification for BEEP frequency
7879 Cassette Parameter: 1, MERGE; 2, CHAIN; 4, Data file; 8, M.L.; 10, RMT 1; 40, CLOAD?; 80, load operation
787A Unused

787B Position of blink character in display, plus 8
787C Flags: 01, blink cursor enabled: 80, a character is now blinked
787D Code of character blinked
787E High byte, location (in Display Buffer) of Blink Cursor
787F Low byte, location of Blink Cursor
7880 Display Parameter; determines display at READY, depending on bits set. 20, display contents of R0; otherwise, contents of Input Buffer. 10, display as BASIC line; 04, include colon after line number. 40, cursor activated in display. 80, ERROR message displayed. 08, RESERVE keyphrase displayed. 01, RESERVE template displayed, with previous display saved in 7B10-7BAB.
7881 Parsing Parameter; specifies type of instruction that will conform to rules of syntax for evaluating BASIC expression.
7882 Subpointer, used with GOSUB and FOR stacks; also, 01 indicates a USING statements that is not part of a PRINT or PAUSE statement.
7883 High byte, LET Pointer; also, 80 indicates no data pass with CALL.
7884 Low byte, LET Pointer; also used to store code of RESERVE key.
7885 With LET Pointer: length if string, 88 if numeric variable
7886 High byte, INPUT Pointer (starting address of variable)
7887 Low byte,INPUT Pointer
7888 With INPUT Pointer: length if string, 88 if numeric variable
7889 Flag: 01, variable subscript(s) being parsed
788A Program Halt Parameter: 10, waiting for input; 20, halt for PRINT; 40, INPUT statement pending; 80, Break.
788B Low byte, Input Buffer Pointer
788C Number of function input arguments (used with parsing routine).
788D TRACE: 00, OFF; other, ON. (Contents of 79D1 stored here by TRON)
788E TRACE Parameter: 00, starting execution of new line; 01, starting execution of a program; 02, expression in Input Buffer is to be evaluated; 04, expression has been evaluated (immediate mode)
788F Low byte, Output Buffer Pointer
7890 Low byte, FOR stack pointer
7891 Low byte, GOSUB stack pointer
7892 Low byte, BASIC DATA STACK Pointer, used by parsing routine
7893 Low byte, BASIC PENDING OP STACK Pointer, used by parsing routine
7894 Low byte, String Buffer Pointer
7895 USING editing character: 10, comma separation; 20, forced sign; 40, asterisk fill; 80, scientific. (01 used to check syntax in interpretation of USING statement)
7896 USING number of characters,including sign, before decimal point
7897 USING number of characters in string; 00, unspecified
7898 USING number of characters including and following decimal point
7899 High byte, START OF VARIABLES in Main memory
789A Low byte, START OF VARIABLES in Main memory
789B Error Code
789C High byte, CURRENT line number; 00 if no program in progress
789D Low byte, CURRENT line number; 00 if no program in progress
789E High byte, beginning address of CURRENT program
789F Low byte, beginning address of CURRENT program
78A0 High byte, PREVIOUS address
78A1 Low byte, PREVIOUS address
78A2 High byte, PREVIOUS line number
78A3 Low byte, PREVIOUS line number
78A4 High byte, beginning of program containing PREVIOUS line
78A5 Low byte, beginning of program containing PREVIOUS line
78A6 High byte, SEARCH address
78A7 Low byte, SEARCH address

78A8 High byte, SEARCH line number
78A9 Low byte, SEARCH line number
78AA High byte, beginning of program containing SEARCH line
78AB Low byte, beginning of program containing SEARCH line
78AC High byte, BREAK address
78AD Low byte, BREAK address
78AE High byte, BREAK line number
78AF Low byte, BREAK line number
78B0 High byte, beginning of program containing BREAK line
78B1 Low'byte, beginning of program containing BREAK line
78B2 High byte, ERROR address
78B3 Low byte, ERROR address
78B4 High byte, ERROR line number
78B5 Low byte, ERROR line number
78B6 High byte, beginning of program containing ERROR line
78B7 Low byte, beginning of program containing ERROR line
78B8 High byte, ON ERROR address
78B9 Low byte, ON ERROR address
78BA High byte, ON ERROR line number
78BB Low byte, ON ERROR line number
78BC High byte, beginning of program containing ON ERROR line
78BD Low byte, beginning of program containing ON ERROR line
78BE High byte, DATA pointer; 80 indicates no DATA line yet located.
78BF Low byte, DATA pointer

FIXED VARIABLES

78C0-CF	A$	7920-27	E	7858-5F	L	7998-9F	T
78D0-DF	B$	7928-2F	F	7960-67	M	79A0-A7	U
78E0-EF	C$	7930-37	G	7968-6F	N	79A8-AF	V
78F0-FF	D$	7938-3F	H	7970-77	O	79B0-B7	W
7900-07	A	7940-47	I	7978-7F	P	79B8-BF	X
7908-0F	B	7948-4F	J	7980-87	Q	79C0-C7	Y
7910-17	C	7950-57	K	7988-8F	R	79C8-CF	Z
7918-1F	D			7990-97	S		

SYSTEM RAM <FOR OPTIONS>

79D0 ROM Bank: 00, ROM 1; 01, ROM 2
79D1 OPN device code: 60, LCD; 5C, CMT; 58, MGP; C4, LPRT; C0, COM
79D2 Unused
79D3 If 55, bypass setting of Modulation clock frequency, serial output
79D4 If 55, bypass keyboard scan; obtain input from external device
79D5 Unused
79D6 Unused
79D7 Unused
79D8 Unused
79D9 Used (by option)
79DA If 55, interrupt routine address in option is obtained from 79DB-D
79DB High byte, interrupt routine address in option
79DC Low byte, interrupt routine address in option
79DD Unused
79DE Unused
79DF Unused

79E0 High byte of X-coordinate, in signed binary (GRAPH mode); also used for high byte of address of first line to be LLISTed

79E1 Low byte of X-coordinate, in signed binary (GRAPH mode); also used for low byte of address of first line to be LLISTed

79E2 High byte of Y-coordinate, in signed binary (GRAPH mode); also, high byte of address of last line to be LLISTed

79E3 Low byte of Y-coordinate, in signed binary (GRAPH mode); also, low byte of address of last line to be LLISTed

79E4 High byte, paper reverse feed count (counts from 0001 to 01FF)

79E5 Low byte, paper reverse feed count

79E6 Location of pen (00 to D8)

79E7 Low byte, extension of X-coordinate beyond available space

79E8 High byte, extension of X-coordinate beyond available space

79E9 Pen parameter; specifies whether pen is to be raised or lowered

79EA LINE TYPE (0 to 9), GRAPH mode

79EB Dotted-line counter

79EC Current pen position: 00, up; 01, down

79ED X-motor hold counter

79EE Motor Phase; stored into Port C

79EF Y-motor hold counter

79F0 Print Mode: 00, TEXT; FF, GRAPH

79F1 Printer disable: 0F, pen change mode; FF, low btry or head blocked

79F2 ROTATE setting (0 to 3)

79F3 COLOR setting (0 to 3)

79F4 CSIZE setting (1 to 9)

79F5 LPRINT parameter: 00, single-item; 04, comma LPRINT; 08, semicolon LPRINT; 01, last item reached. Also used with LLIST for maximum permissable line length.

79F6 With LINE, used as direction parameter; with LLIST, to determine line feed; with COLOR, to save pen location.

79F7 Type of data LPRINTed: 00, numeric; FF, character string

79F8 Temporary storage of pen location during paper feed

79F9 Flag: flag indicating powerup or interrupt in progress

79FA Unused

79FB Unused

79FC Unused

79FD Unused

79FE Unused

79FF LOCK Mode: 00, LOCK; FF, UNLOCK

BASIC REGISTERS

Seven registers, used in BASIC computation, are located as follows:

7A00-07	R0	(Floating-point accumulator)
7A08-0F	R1	(Scratch register)
7A10-17	R2	(Second operand)
7A18-1F	R3	(Scratch register)
7A20-27	R4	(Scratch register)
7A28-2F	R5	(Scratch register)
7A30-37	R6	(Temporary storage register)

Each of these registers may contain a two-byte signed-binary number; a floating-point decimal number; or a pointer to a character string. The eight bytes in a register are utilized as follows:

BYTE	SIGNED BINARY	FLOATING-POINT	STRING POINTER
0	Unused	Exponent (binary)	Unused
1	Unused	Sign: 80 specifies -	Unused
2	Unused	1st & 2nd digits	Unused
3	Unused	3rd & 4th digits	Unused
4	B2 (to identify)	5th & 6th digits	D0 (to identify)
5	1st byte	7th & 8th digits	High byte, address
6	2nd byte	9th & 10th digits	Low byte, address
7	Unused	11th & 12th digits	Number of characters

Mantissas of calculated floating-point numbers are rounded to ten digits prior to display, printing, or storage. A string pointer formed by execution of CHR$ uses C1 instead of D0 as byte 4.

The area 7A00-7A37 is also used for other purposes, as follows:

The powerup routine uses addresses 7A10-14 (7A30-34 in earlier ROM versions) to temporarily save the addresses to be stored into the memory allocation pointers (RAM addresses 7860-64). Other pertinent information is stored by the powerup routine as follows:

7A20 If 00, powerdown was by 'OFF' key; 01, timed powerdown; 02, other type powerdown; 04, memory allocation pointers have been changed.
7A21 Set to other than zero by option with low battery power, indicating need for display of 'CHECK' message.
7A22 Set by option to indicate need for 'CHECK' message

A powerdown resulting from 7 minutes of non-use is identified by storage of A0, A1, ..., AF into addresses 7A10-7AFF; the stack pointer is saved in 7A30-31. (The powerdown initiated by use of the OFF key stores 50, 51, ..., 5F into 7A10-7AFF.)

The following are used in calculation of transcendental functions:

7A18 Flag: 00, SIN, COS, or ASN; 01, LOG, LN, or EXP; 20, ACS; 40, TAN or ATN; 80, result must be subtracted from 90 (inverse trig fctns)
7A20 Flag: 00, SIN; 01, COS

The PRINT and PAUSE routines use the area 7A10-7A34 to form a string of characters that represent numeric data formatted for display. The same area is used to form an ERROR or BREAK message.

The area beginning 7A08 is used as a scratch area in the interpretation of USING statements, and in execution of certain printer routines. In some of the printing routines, the contents of CPU register Y are saved in 7A26-7A27.

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

and a calculation mode to provide people with a highly portable tool capable of serious number crunching. Repetitive calculations may be performed by writing programs in BASIC or buying application packages that solve specific types of problems. A special CALC mode makes it easy to perform single calculations.

The 64 kilobytes of ROM is accompanied by 17.5 kilobytes of RAM in the basic unit. The unit is equipped with four slots that can accept additional RAM or ROM modules. Up to 16K of extra RAM or 256K of extra ROM can be added to the unit using these slots.

Space is also provided in the machine to install an optional magnetic card reader and an HP Interface Loop (HP-IL). The card reader reads (and writes) 10-inch magnetic strips that can hold up to 1.3 kilobytes of information. The HP-IL modules permits the HP-71B to be connected to a variety of peripheral devices such as a magnetic tape drive, printers and test equipment.

The pocketable unit is powered by four "AAA" alkaline batteries or by an optional alternating current adaptor.

The 71B has a block QWERTY-style keyboard with typing aids for facilitating use of BASIC. A separate 10-digit pad is used for numeric inputs. The entire keyboard is redefinable. Mylar overlays are available so that users can customize the computer for specific purposes. The display is an 8 by 132-element, single-line dot matrix LCD with a large font. It is supported by status annunciators. Twenty-two characters within a 96-character line may be displayed at a time. However, since each element in the display is individually controllable, users may create their own set(s) of graphics or customized characters.

The version of HP extended BASIC used in the machine provides over 240 instructions. In addition to compliance with IEEE floating-point math standards, it includes enhancements such as trigonometric functions and statistics operations. The language also provides: complete control of display pixels, ability to maintain multiple programs in memory, dynamically declared variables, and the ability to define multi-line user functions. Furthermore, additional keywords can be added to extend the language.

A built-in timer allows events to be controlled in real time. Time and date information is retained even when the HP-71B is turned off.

The handheld computer also has a calculation mode that enables it to function as an advanced calculator. Even though this mode is not programmable, it does provide error-checking and results can be tracked as a calculation is performed. Variables assigned values in the BASIC mode can be used in the CALC mode (as well as the reverse).

HP plans to support the new handheld with a series of application programs including math, curve fitting, AC circuit analysis, surveying, finance and text-editing. The firm also plans to provide a combination Assembly/FORTH System for developing special application programs. There will also be a user's library made available for the new model. Programs submitted by users will be made available through the library.

Hewlett-Packard Company has also indicated that the HP-71B will be a completely open machine. That is, it is encouraging third-party vendors to develop software, hardware and interfaces for the computer. It is making extensive documentation and a choice of development languages available as well as offering support of several media on which software may be delivered.

For instance, in addition to the usual owner's documentation, HP is making available three volumes of internal design specifications. These manuals contain detailed information on the unit's internal hardware and software. Three types of software development languages will be available: HP-71B Assembly, FORTH and BASIC. Software vendors will be able to distribute their wares on ROMs, cassette tapes and magnetic strip cards.

The HP-71B is available from HP authorized dealers. It is said to be priced at less than $550.00. For the location of the nearest dealer phone, toll free, 1-800-FOR-HPPC.

SCHEDULE PLOTTER PROGRAM

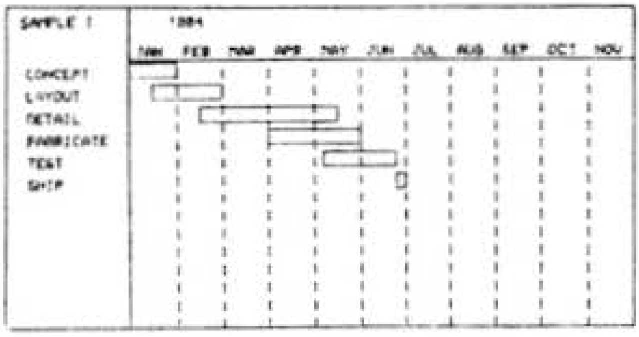
Richard H. Chrystie, 14824 E. Walbrook, Hacienda Heights, CA 91745, submitted this program. You can use it to plot schedules over a period of weeks or months.

To use it, just load the program and execute a RUN command or press the DEF key and and then the S key. Respond to the queries and watch your schedule be plotted right before your eyes!

You can customize the program to your own liking. For instance, if, when using it in the monthly mode, you want it to start with a month other than January, change the start date in line 620. You can then change the column month labels that begin at line 780.

Program *Schedule Plotter.*

```
 10:"S"REM   SCHED
    PLOT
 11:TEXT :COLOR 0:
    CLEAR
 15:DIM A$(12),B(1
    2),C(12)
 20:INPUT "PROGRAM
     TITLE?",B$
100:GOSUB 700
110:GLCURSOR (0,-4
    00)
120:IF Z$="M"THEN
    600
400:N=1
410:INPUT "TASK TI
    TLE?",A$(N)
420:INPUT "START (
    DAYS FROM NOW)
    ?",B(N):B(N)=B
    (N)*6
430:INPUT "TASK DU
    RATION (DAYS)?
    ",C(N):C(N)=C(
    N)*6
435:IF (B(N)+C(N))
    >330THEN LET C
    (N)=330-B(N)
440:L=183-(15*N)
449:COLOR 2:K=-80
450:GLCURSOR (L,-1
    3):LPRINT A$(N
    )
455:LINE (L,K-B(N)
    )-(L+10,K-B(N)
    -C(N)),,,B
460:GLCURSOR (0,-3
    80)
470:IF N=12THEN 49
    8
480:INPUT "WANT AN
    OTHER TASK?",D
    $
481:IF D$="N"THEN
    498
485:N=N+1
487:GOTO 410
498:GLCURSOR (0,-5
    00)
499:INPUT "WANT AN
    OTHER PLOT?";Z
    $
500:IF Z$="Y"THEN
    502
501:GLCURSOR (0,-5
    00):END
502:GOTO 11
600:N=1
610:INPUT "TASK TI
    TLE?",A$(N)
620:INPUT "START (
    MOS AFTER 1 JA
    N)?",B(N):B(N)
    =B(N)*30
630:INPUT "TASK DU
    RATION (MOS)?"
    ,C(N):C(N)=C(N
    )*30
635:IF (B(N)+C(N))
    >330THEN LET C
    (N)=330-B(N)
640:L=183-(15*N)
649:COLOR 2:K=-80
650:GLCURSOR (L,-1
    3):LPRINT A$(N
    )
655:LINE (L,K-B(N)
    )-(L+10,K-B(N)
    -C(N)),,,B
660:GLCURSOR (0,-3
    80)
670:IF N=12THEN 69
    8
680:INPUT "WANT AN
    OTHER TASK?",D
    $
681:IF D$="N"THEN
    698
685:N=N+1
687:GOTO 610
698:GLCURSOR (0,-5
    00)
699:GOTO 499
700:GRAPH
710:COLOR 0
715:LINE (0,0)-(21
    5,-410),,,B
720:LINE (0,-80)-(
    215,-80)
730:LINE (180,-80)
    -(180,-410)
740:FOR X=110TO 37
    0STEP 30
745:LINE (0,-X)-(1
    80,-X),5
750:NEXT X
755:CSIZE 1:ROTATE
    1:COLOR 1
760:GLCURSOR (203,
    -10)
775:LPRINT B$
776:INPUT "WEEKS O
    R MONTHS (W/M)
    ?",Z$
777:IF Z$="W"THEN
    800
780:GLCURSOR (103,
    -85):LPRINT "J
    AN"
781:GLCURSOR (183,
    -115):LPRINT "
    FEB"
782:GLCURSOR (183,
    -145):LPRINT "
    MAR"
783:GLCURSOR (183,
    -175):LPRINT "
    APR"
784:GLCURSOR (183,
    -205):LPRINT "
    MAY"
785:GLCURSOR (183,
    -235):LPRINT "
    JUN"
786:GLCURSOR (183,
    -265):LPRINT "
    JUL"
787:GLCURSOR (183,
    -295):LPRINT "
    AUG"
788:GLCURSOR (183,
    -325):LPRINT "
    SEP"
789:GLCURSOR (183,
    -355):LPRINT "
    OCT"
790:GLCURSOR (183,
    -385):LPRINT "
    NOV"
791:GLCURSOR (203,
    -105):LPRINT "
    1984"
792:RETURN
800:GLCURSOR (183,
    -90):LPRINT "1
    "
801:GLCURSOR (183,
    -120):LPRINT "
    2"
802:GLCURSOR (183,
    -150):LPRINT "
    3"
803:GLCURSOR (183,
    -180):LPRINT "
    4"
804:GLCURSOR (183,
    -210):LPRINT "
    5"
805:GLCURSOR (183,
    -240):LPRINT "
    6"
806:GLCURSOR (183,
    -270):LPRINT "
    7"
807:GLCURSOR (183,
    -300):LPRINT "
    8"
808:GLCURSOR (183,
    -330):LPRINT "
    9"
809:GLCURSOR (183,
    -360):LPRINT "
    10"
810:GLCURSOR (183,
    -390):LPRINT "
    11"
811:GLCURSOR (203,
    -105):LPRINT "
      WEEKS"
812:RETURN
STATUS 1         1722
```



APPOINTMENT CALENDAR

This program can be of use to almost everybody. It is particularly valuable because it gives you the capability of searching for a key word within a field to help you locate a specific appointment.

For example, suppose you had been invited to a golf game on Friday, September 23rd. But, you do not remember the exact time. Just run the program and select item 2 (Find Appointment) from the menu. Respond to the prompt for a date. If you make a mistake while entering it, error-trapping will advise you and prompt you again. Once you have entered the date, the PC will list all your committments on that date. The listing will be in the order in which you entered them. (In keeping with a goal of having enough memory available for a practical calendar, a rather lengthy sorting routine was omitted.) When the appointments for the given date have been shown, the program returns to its main memory.

Here is another example. Suppose you are running out of a favorite product that is sold only by Amway at Amway parties. You cannot remember when the next party is scheduled. Just run the program and select item 3 (Find Date) from the menu. In a few seconds the computer will begin displaying all of your Amway parties scheduled for the entire year! You could even spell Amway wrong and, unless you really mutilate it, the program will still find it for you!

If you like, you can also review appointments by the month (choose item 4 from the main menu). Other capabilities included in the program are: deleting the appointment file, saving data, loading data and adding appointments.

Operating Tips

There are a few things you need to remember about the PC-2 and PC-1500 Pocket Computers when

Program *Appointment Calendar.*

```
  5:DIM M$(60)*25,
    N$(60)*25,J$(2
    )*2,K$(2)*5
 10:"A":CLS :WAIT
    99:CURSOR 4:
    PRINT "APPOINT
    MENT CALENDAR"
 12:CLS :BEEP 1,10
    0
 20:CLS :CURSOR 8:
    PRINT "***MENU
    ***"
 22:CLS
 30:PAUSE "1.ENTER
     DATES/APPOINT
    MENTS"
 40:PAUSE "2.FIND
    APPOINTMENT"
 50:PAUSE "3.FIND
    DATE"
 60:PAUSE "4.REVIE
    W APPTS BY MON
    TH
 70:PAUSE "5.I/O O
    PTIONS      "
 75:PAUSE "6 DELET
    E APPOINTMENT"
 80:PAUSE "7.SEE M
    ENU AGAIN"
 90:INPUT M
100:ON MGOSUB 1000
    ,2000,3000,400
    0,5000,6000,70
    00
110:GOTO 12
1000:BEEP 1,150:
     PRINT "IS TH
     IS AN ORIGIN
     AL ENTRY?"
1002:INPUT "
           (Y/N)?
     ";J$
1003:IF J$="N"
     THEN GOTO 80
     00
1004:FOR L=1TO 60
1005:INPUT "ENTER
      DATE (MO/DA
     ): ";M$(L)
1015:IF M$(L)="ZZ
     "THEN GOTO 1
     050
1016:IF LEN (M$(L
     ))<>5THEN
     GOTO 9000
1020:INPUT "ENTER
      APPOINTMENT
     :";N$(L)
1030:NEXT L
1040:GOTO 20
1050:RETURN
2000:BEEP 1,150
2010:PAUSE "FIND
     APPOINTMENT"
2015:INPUT "ENTER
      DATE (MO/DA
     ): ";K$
2020:IF LEN (K$)<
     >5THEN BEEP
     5,150,25:
     PRINT "FORMA
     T IS 4 DIGIT
     S---REDO":
     GOTO 2015
2021:X=1:Y=0:GOTO
     2025
2024:X=X+1
2025:FOR X=XTO L-
     1
2030:IF K$=M$(X)
     THEN 2070
2040:NEXT X
2045:IF Y>=1THEN
     2110
2050:PRINT "NO AP
     POINTMENT ON
     : ";K$
2055:K$=""
2060:RETURN
2070:PRINT "YOUR
     APPOINTMENT
     ON: ";M$(X);
     "IS:"
2075:PRINT "IS: "
     ;N$(X)
2077:Y=Y+1
2080:FOR C=1TO 75
     :NEXT C
2093:GOTO 2024
2110:PRINT "NO MO
     RE APPTS ON
     THIS DATE"
2120:K$=""
2130:RETURN
3000:BEEP 1,150
3005:CLS :CURSOR
     8:PRINT "FIN
     D DATE"
3010:INPUT "ENTER
      APPOINTMENT
      ";U$
3012:X=0:Y=0
3014:X=X+1
3020:FOR X=XTO L-
     1
3025:IF U$=N$(X)
     THEN 3050
3030:IF LEFT$ (N$
     (X),4)=LEFT$
     (U$,4)THEN 3
     050
3035:IF RIGHT$ (N
     $(X),4)=
     RIGHT$ (U$,4
     )THEN 3050
3040:NEXT X
3042:IF Y>=1THEN
     3090
3045:GOTO 3080
3050:PRINT M$(X);
     " ";N$(X)
3052:Y=Y+1
3055:FOR R=1TO 10
     0:NEXT R
3057:GOTO 3014
3060:U$=""
3070:RETURN
3080:PRINT "THAT
     APPT IS NOT
     ON FILE"
3090:U$=""
3100:RETURN
4000:CLS :BEEP 1,
     150
```

using this program.

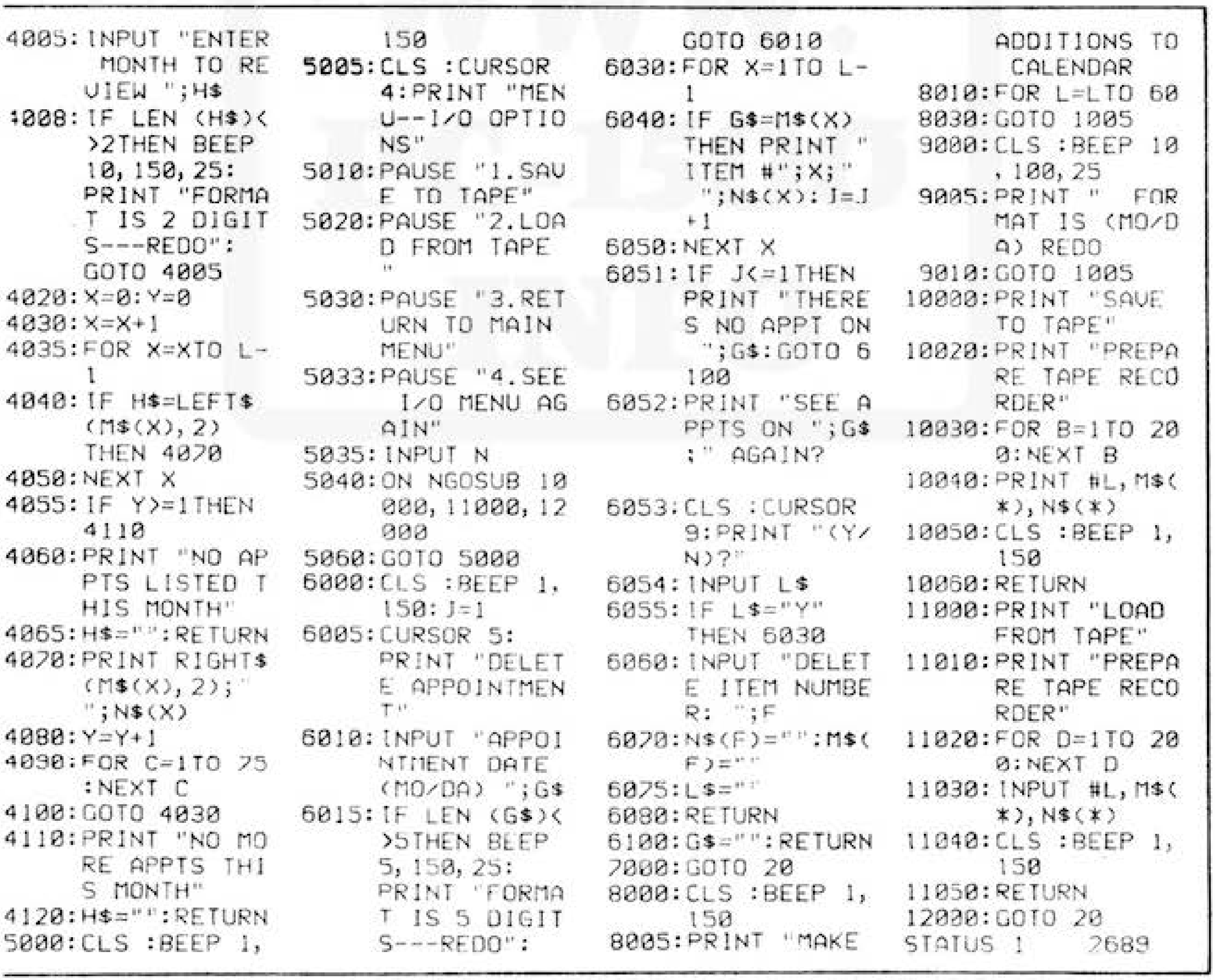
First, remember that when you give a RUN command, all variables stored in user memory will be lost. Thus, you should only use the RUN command when you use the program for the first time or after loading the *program* from tape. It is necessary to do so under these circumstances in order to properly dimension the string arrays used by the program.

How do you start the program without typing RUN? Just press the DEF key followed by the letter A (DEF/A).

In order to start building up an appointments list you will need to choose item 1 from the main menu. When you do this you will queried as to whether this is an original entry. If it is an original entry, (the first time you have entered information into an appointment file) respond with a Y for yes. A yes response causes the data file to be initialized to the empty condition. If you are adding data to a file that already has appointments stored, then be sure to answer N for no!

You can customize the display timings to your own liking. There are three basic methods of controlling the length of time a display stays on the screen of a PC-2 or PC-1500. The PAUSE and PRINT commands are used in this program. PAUSE gives a display time that is set at about one second. Using the PRINT command coupled with WAIT (followed by a number), you can keep the display on for whatever period is desired. A WAIT value of 99 is used in this program. It is specified in line 10. If you would like the displays to last a little longer, increase the value in line 10. Remember that altering this value will change all PRINT statements used in the program but has no effect on the PAUSE statements.

This program has been designed to fit in a 4K



```
4005:INPUT "ENTER
      MONTH TO RE
     VIEW ";H$
4008:IF LEN (H$)<
     >2THEN BEEP
     10,150,25:
     PRINT "FORMA
     T IS 2 DIGIT
     S---REDO":
     GOTO 4005
4020:X=0:Y=0
4030:X=X+1
4035:FOR X=XTO L-
     1
4040:IF H$=LEFT$
     (M$(X),2)
     THEN 4070
4050:NEXT X
4055:IF Y>=1THEN
     4110
4060:PRINT "NO AP
     PTS LISTED T
     HIS MONTH"
4065:H$="":RETURN
4070:PRINT RIGHT$
     (M$(X),2);"
     ";N$(X)
4080:Y=Y+1
4090:FOR C=1TO 75
     :NEXT C
4100:GOTO 4030
4110:PRINT "NO MO
     RE APPTS THI
     S MONTH"
4120:H$="":RETURN
5000:CLS :BEEP 1,
     150
5005:CLS :CURSOR
     4:PRINT "MEN
     U--I/O OPTIO
     NS"
5010:PAUSE "1.SAV
     E TO TAPE"
5020:PAUSE "2.LOA
     D FROM TAPE
     "
5030:PAUSE "3.RET
     URN TO MAIN
     MENU"
5033:PAUSE "4.SEE
      I/O MENU AG
     AIN"
5035:INPUT N
5040:ON NGOSUB 10
     000,11000,12
     000
5060:GOTO 5000
6000:CLS :BEEP 1,
     150:J=1
6005:CURSOR 5:
     PRINT "DELET
     E APPOINTMEN
     T"
6010:INPUT "APPOI
     NTMENT DATE
     (MO/DA) ";G$
6015:IF LEN (G$)<
     >5THEN BEEP
     5,150,25:
     PRINT "FORMA
     T IS 5 DIGIT
     S---REDO":
     GOTO 6010
6030:FOR X=1TO L-
     1
6040:IF G$=M$(X)
     THEN PRINT "
     ITEM #";X;"
      ";N$(X):J=J
     +1
6050:NEXT X
6051:IF J<=1THEN
     PRINT "THERE
     S NO APPT ON
      ";G$:GOTO 6
     100
6052:PRINT "SEE A
     PPTS ON ";G$
     ;" AGAIN?

6053:CLS :CURSOR
     9:PRINT "(Y/
     N)?"
6054:INPUT L$
6055:IF L$="Y"
     THEN 6030
6060:INPUT "DELET
     E ITEM NUMBE
     R: ";F
6070:N$(F)="":M$(
     F)=""
6075:L$=""
6080:RETURN
6100:G$="":RETURN
7000:GOTO 20
8000:CLS :BEEP 1,
     150
8005:PRINT "MAKE
     ADDITIONS TO
      CALENDAR
8010:FOR L=LTO 60
8030:GOTO 1005
9000:CLS :BEEP 10
     ,100,25
9005:PRINT "  FOR
     MAT IS (MO/D
     A) REDO
9010:GOTO 1005
10000:PRINT "SAVE
      TO TAPE"
10020:PRINT "PREPA
      RE TAPE RECO
      RDER"
10030:FOR B=1TO 20
      0:NEXT B
10040:PRINT #L,M$(
      *),N$(*)
10050:CLS :BEEP 1,
      150
10060:RETURN
11000:PRINT "LOAD
      FROM TAPE"
11010:PRINT "PREPA
      RE TAPE RECO
      RDER"
11020:FOR D=1TO 20
      0:NEXT D
11030:INPUT #L,M$(
      *),N$(*)
11040:CLS :BEEP 1,
      150
11050:RETURN
12000:GOTO 20
STATUS 1    2689
```

machine. If you have an 8K memory module installed (which nets you 10K), you can change the number of potential records by the changing the values in the loops at lines 1004 and 8010, to something in the order of 100. If you do this you must also change the numbers within the parenthesis of the dimension declarations (in line 5) to 100 instead of 60. If you assign too many records in the dimension statements, you will simply get an OUT OF MEMORY error when you try to use the program. Reduce the values used if this occurs.

Finally, if you have never saved a data file before using your PC, then you may be in for a bit of a surprise. Did you realize that even if you only have two appointment entries in your appointment file, a tape data save command will save all of the records that have been dimensioned, even though most of them are empty? Yes, the PC will save every bit of the reserved array space, whether or not it contains information. The result is that saving the appointment file on tape will take a fixed amount of time, regardless of how many entries you have made. Have patience, please.

This program was submitted by: *Steve Eichman, 5809 Northland Road, Manteca, CA 95336.*

FORMATTED MORTGAGE PROGRAM

Here is a program that will calculate monthly payments and amortization information on a mortgage or loan and print out a neatly formatted schedule.

The program asks for the mortgage/loan amount, annual interest percentage and amount of monthly payments. If the monthly payment is specified, the amortization will be calculated. Optionally, if the monthly payment is not given (simply press ENTER at the prompt), the program will ask the user for amortization and calculate the appropriate monthly payments. Next, the starting date and number of payments to be printed in the schedule must be provided. Finally, the program gives the option of specifying a number of extra payments. If only regular payments will be made, press the ENTER key at this prompt. When executed the program lists a summary of the input data, followed by a payment schedule that shows the payment number, date, interest and principal portions of the payment, and the account balance as of that date.

NOTICE: This program was written to calculate mortgage schedules according to Canadian laws. However, with a few simple changes it can be adapted to other types of loans.

In Canada, the law states that interest on mortgages may not be charged in advance. This means that a lender is not entitled to interest more often than the mortgage's compounding frequency which is a maximum of semi-annually. Thus, if interest payments are in fact being made more often than the law permits (such as monthly), then interest is being charged in advance, unless an appropriate discount is made. This may be done by reinvesting the interest portion of the monthly payment at the mortgage rate until the end of the current compounding period. This effectively reduces the actual interest rate. The effective

Program *Formatted Mortgage.*

```
  5:PAUSE "... MOR
    TGAGE SCHEDULE
     ...":LOCK
 10:REM     1983 -
     Peter V. HART
    MANN
 20:CLEAR :CSIZE 2
    :B=10:U=1:DIM
    PN(B),EP(B)
 30:A$="Jan":B$="F
    eb":C$="Mar":D
    $="Apr":E$="Ma
    y":F$="Jun"
 40:G$="Jul":H$="A
    ug":I$="Sep":J
    $="Oct":K$="No
    v":L$="Dec"
 50:INPUT "AMOUNT
    = $ ";A
 60:INPUT "INTERES
    T (%) = ";I
 70:C=2
 80:INPUT "MONTHLY
     PAYMENTS = $"
    ;P
 90:IF P=0INPUT "A
    MORTIZED OVER
    (yrs) = ";Z:N=
    Z*12
100:INPUT "STARTIN
    G DATE- DAY: "
    ;D,"MONTH: ";M
    ,"YEAR: ";Y
110:Y=Y-100*INT (Y
    /100)
120:INPUT "NO. OF
    PAYMENTS = ";U
130:INPUT "EXTRA P
    AYMENTS ? ";Z$
140:IF Z$<>"Y"GOTO
     170
150:FOR J=0TO B:
    INPUT "PAYMENT
     # = ";PN(J)
155:IF PN(J)=0GOTO
     170
160:INPUT "PAYMENT
     AMOUNT = $";E
    P(J):NEXT J
170:R=(1+(I/100)/C
    )^(C/12)-1
180:IF P>0GOTO 200
190:X=(R+1)^N:P=A*
    R*X/(X-1):GOTO
    210
200:N=LOG (P/(P-A*
    R))/LOG (1+R):
    Z=N/12
210:COLOR 2:LPRINT
    "--------------
    -----"
220:COLOR 3:LPRINT
    "MORTGAGE SCHE
    DULE "
230:COLOR 2:LPRINT
    "--------------
    -----"
240:COLOR 0:LF 1:
    USING "#######
    "
250:LPRINT "AMOUNT
      = $";A
260:LPRINT "INTERE
    ST=";USING "##
    ###.##";I;"%"
270:LPRINT "YEARS
      =";USING "##
    ####.##";Z
```

monthly interest rate is given by the equation:

$$r = ((1 + i/c)^{(c/12)}) - 1$$

In this formula, r = the effective monthly interest rate, i = the annual interest rate and c = the number of compounding periods per year.

Program Options

The maximum number of extra payments that can be handled by the program is a function of the amount of memory available. The value of variable B in line 20 of the listing sets the allowable limit. In an unexpanded PC-1500 the number of such payments is limited to 11. (B is set to 10 and is then used to DIMension an array having 11 elements.) If you have additional memory in your PC you can increase this value accordingly.

```
280:COLOR 1:LF 1:
    LPRINT "EFF.MO
    NTHLY RATE:"
290:LPRINT USING "
    ##.###########
    ";R;"%"
300:COLOR 3:LF 1:
    LPRINT "MONTHL
    Y PAYMENTS:"
310:LPRINT "     $"
    ;USING "#####.
    ##";P
320:COLOR 2:LPRINT
    "-------------
    -----"
330:LF 2:CSIZE 1:
    COLOR 0: IF U=0
    GOTO 510
340:LPRINT "  #
    DATE        INT.
      PRINC. BALAN
    CE"
350:LPRINT "------
    --------------
    --------------
    --"
360:LPRINT TAB 3;
    USING "###";D;
    @$(M);Y;TAB 29
    ;USING "######
    #";A
370:K=M+1:IF K>12
    LET Y=Y+1:IF K
    >12LET K=1
380:S=0:T=0
390:FOR J=KTO 12:Q
    =P
400:FOR F=1TO B:IF
    PN(F)<>UGOTO 4
    20
410:Q=EP(F):GOTO 4
    30
420:NEXT F
430:G=A*R:H=Q-G:A=
    A-H:IF A>0.01
    GOTO 450
440:H=H+A:Q=H+G:A=
    0
450:LPRINT USING "
    ###";U;D;@$(J)
    ;Y;USING "####
    ##.##";G;USING
    "#####.##";H;
    USING "#######
    ";A
460:S=S+G:T=T+H:U=
    U+1:IF A=0GOTO
    480
470:NEXT J
480:LPRINT "------
    --------------
    --------------
    --":X=Y+1900
490:LPRINT " TOTAL
    ";USING "#####
    ";X;USING "###
    ####.##";S;
    USING "#####.#
    #";T:LF 1
500:K=1:Y=Y+1:IF U
    <UAND A>0.01
    GOTO 380
510:UNLOCK :END
STATUS 1         1649
```

To use the program to calculate other types of mortgages or simple loans, change the title in line 5 to suit your purposes. Also change line 70 to:

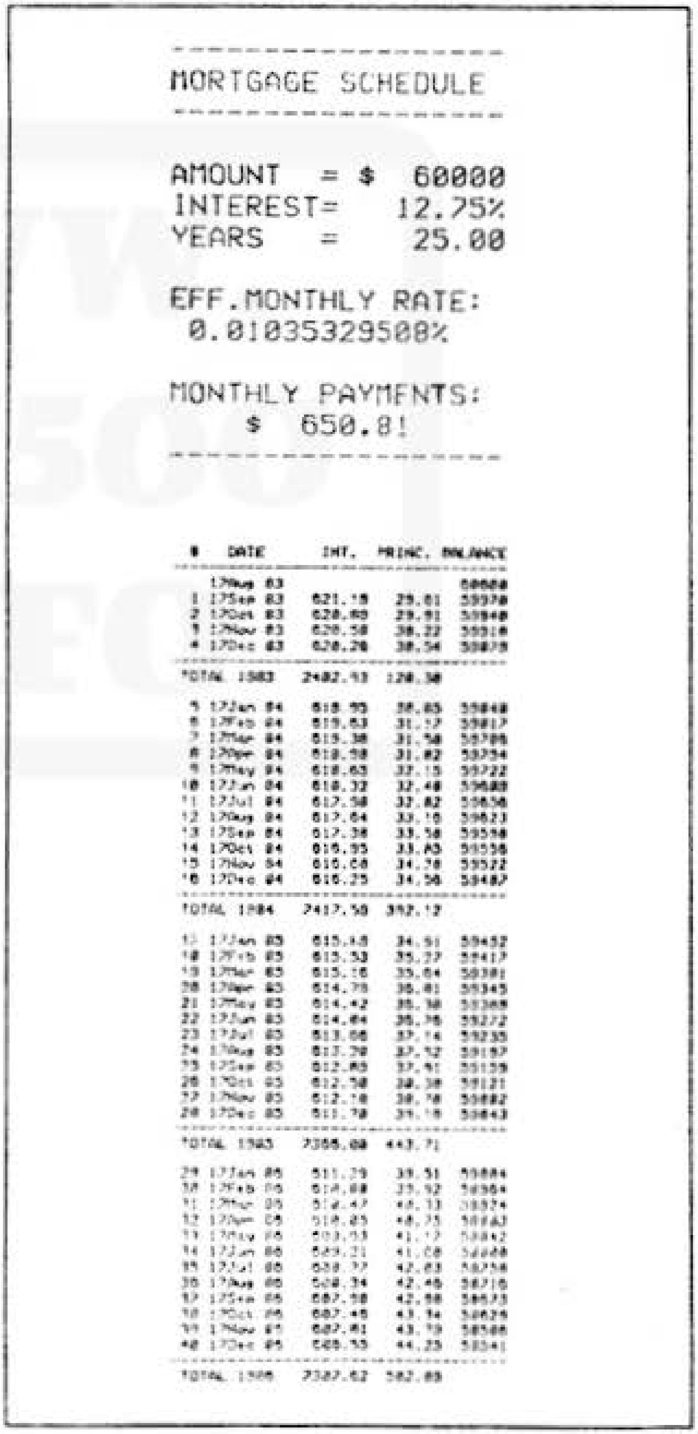
70 INPUT COMPOUNDED(* A YEAR)";C

and alter line 270 to read:

270 R = I

This program was submitted by: *Peter V. Hartmann, P.O. Box 7003, Station A, Toronto, Ontario, Canada M5W 1X7.*

Sample *Listing from Formatted Mortage Program.*



```
-----------------
MORTGAGE SCHEDULE
-----------------

AMOUNT   = $  60000
INTEREST=    12.75%
YEARS    =    25.00

EFF.MONTHLY RATE:
 0.01035329508%

MONTHLY PAYMENTS:
    $  650.8!
-----------------

 #   DATE      INT.   PRINC. BALANCE
------------------------------------
    17Aug 83                   60000
  1 17Sep 83  621.19   29.61   59970
  2 17Oct 83  620.89   29.91   59940
  3 17Nov 83  620.58   30.22   59910
  4 17Dec 83  620.26   30.54   59879
------------------------------------
 TOTAL 1983  2482.93  120.30

  5 17Jan 84  619.95   30.85   59848
  6 17Feb 84  619.63   31.17   59817
  7 17Mar 84  619.30   31.50   59786
  8 17Apr 84  618.98   31.82   59754
  9 17May 84  618.65   32.15   59722
 10 17Jun 84  618.32   32.48   59689
 11 17Jul 84  617.98   32.82   59656
 12 17Aug 84  617.64   33.16   59623
 13 17Sep 84  617.30   33.50   59590
 14 17Oct 84  616.95   33.85   59556
 15 17Nov 84  616.60   34.20   59522
 16 17Dec 84  616.25   34.56   59487
------------------------------------
 TOTAL 1984  7417.50  392.13

 17 17Jan 85  615.89   34.91   59452
 18 17Feb 85  615.53   35.27   59417
 19 17Mar 85  615.16   35.64   59381
 20 17Apr 85  614.79   36.01   59345
 21 17May 85  614.42   36.38   59309
 22 17Jun 85  614.04   36.76   59272
 23 17Jul 85  613.66   37.14   59235
 24 17Aug 85  613.28   37.52   59197
 25 17Sep 85  612.89   37.91   59159
 26 17Oct 85  612.50   38.30   59121
 27 17Nov 85  612.10   38.70   59082
 28 17Dec 85  611.70   39.10   59043
------------------------------------
 TOTAL 1985  7365.88  443.71

 29 17Jan 86  611.29   39.51   59004
 30 17Feb 86  610.88   39.92   58964
 31 17Mar 86  610.47   40.33   58924
 32 17Apr 86  610.05   40.75   58883
 33 17May 86  609.63   41.17   58842
 34 17Jun 86  609.21   41.60   58800
 35 17Jul 86  608.77   42.03   58758
 36 17Aug 86  608.34   42.46   58716
 37 17Sep 86  607.90   42.90   58673
 38 17Oct 86  607.45   43.34   58629
 39 17Nov 86  607.01   43.79   58586
 40 17Dec 86  606.55   44.25   58541
------------------------------------
 TOTAL 1986  7307.62  502.09
```

FLIGHT COMPUTER USES PC-1500

A new flight computer dubbed the PETREL has been announced. The unit is based on a Sharp PC-1500 Pocket Computer. A custom made memory expansion module about the size of a matchbook expands the memory of the PC-1500 by 32K of ROM and 6K of RAM. This additional memory allows the unit to hold a data base containing the location of all 1000 VOR stations in the United States. This data base coupled with proprietory operating programs enables the unit to compute the latitude and longitude of any Victor airways intersection after entry of just the VOR identifiers and radials. The system can also compute as many RNAV waypoints as desired for great circle routes up to 5000 miles in length. Up to 104 intersections or waypoints may be stored by the user and recalled by identifier alone. A user can also add up to 127 new entries or changes to the VOR data base. Furthermore, a variety of flight planning, preflight, navigation and inflight functions are included in the computing package.

For instance, the preflight capabilities allow the pilot to enter the empty weight and moments for fuel, oil, seats and baggage just once. This information is retained by the system for future use. Before each flight the operator adds the actual fuel and weight at each station. The PETREL system then computes the total weight and center of gravity figures as well as maximum gross weight and C.G. limits. In fact, the package will store the data needed for two separate aircraft so that one craft can be flown without losing the data used by the other. Additionally, the unit handles all typical conversions used by pilots: miles-nautical to miles-kilometers-feet, farenheit-celsius, liters-gallons-imperial gallons, and kilograms-pounds. It also handles conversions of true airspeed from calibrated airspeed or Mach numbers. The system provides 10 auxiliary memory scratch pads so that converted numbers can be entered directly into operating programs.

While navigating the PETREL can compute the total distance, flight time and fuel requirements without map reference. Three separate flight plans can be stored and run independently. Winds and climb requirements are computed automatically. The distance and bearing between any two airway points or geographic points can be determined in seconds without map reference. ETAs and fuel requirements can then be computed. Using its self-contained VOR data base, LORAN-C and other users can compute and display in a matter of seconds, the latitude and longitude of any VOR intersection. Only the VOR identifier and radial is entered to obtain this information. For a DME fix, just add the distance. Up to 104 intersections or other fixes can be stored and later recalled using just the identifier.

While inflight the system aids pilots through the maintenance of two 15-item general purpose check lists. A holding pattern routine displays the proper pattern entry procedure and provides a countdown timer for the outbound leg. If you input the final approach course, distance and wind, the PETREL will compute the time to missed approach and display a countdown/countup timer with an alarm. The programs are designed to keep inflight key punching to a minimum. For instance, the Distance-Time-Fuel program allows the operator to enter just the distance, leaving the speed and fuel consumption as they had been previously defined, and quickly obtain a current read out of the flight time, fuel used and ETA.

The PETREL Flight Computer is the result of several years of programming and engineering development work. It represents one of the most powerful applications of a pocket computer ever announced. It uses a combination of machine language and BASIC routines plus a comprehensive VOR data base stored on ROM. The PETREL currently is priced at $695.00. For more information contact: *Somerset Engineering Company, P.O. Box 14, Concord, MA 01742.* The phone number is (617) 369-1090.

GRAPHICS PACKAGE FOR THE PC-2

Radio Shack has announced the availability of a program called called the PC-2 Graphics Pak. It is said that the software package makes it easy to create charts using the Radio Shack TRS-80 Pocket Computer Model PC-2 combined with the PC-2 Printer/Cassette Interface.

The package of programs can be used to draw six different types of charts: horizontal and vertical bar charts, bar-segment charts, point, line and pie charts.

Data may be entered from the keyboard of the PC. It is turned into chart format using combinations of labeling and shading. A data file used in a chart can be stored on cassette tape and used to create another chart.

The package retails for $19.95 in the U.S. at Radio Shack Computer Centers and other Radio Shack outlets.

INSIDE THE PC-1500

This is a continuation of the series by *Norlin Rober, 407 North 1st Avenue, Marshalltown, IA 50158.* We pick up in this issue from where we left off in *PCN* Issue 31, with a recap of information relating to the use of RAM in the PC-1500. (This information also applies to the Radio Shack PC-2.)

Then Norlin goes into the subject of ROM routines used in the PC-1500 (which are again generally applicable to the PC-2). There are many routines within the BASIC ROM that may be of use to machine language programmers. Norlin has done an amazing job of uncovering these routines and providing a synopsis of their capabilities. Space does not permit the publication of all the ROM information he has dug up in this issue, so look for more in future editions of *PCN.*

THE BASIC STACK

The BASIC stack, located 7A38-7AFF, includes a FOR stack, a GOSUB stack, and an arithmetic stack.

The FOR loop stack begins at 7A38. Up to 16 loops may be pending at a given time (if the BASIC stack is not required for other purposes). Each FOR register contains 12 bytes, used as follows:

0-1 Memory address of control variable
2-3 Test value (in signed binary)
4-5 Step size (in signed binary)
6-7 Address of next statement following FOR; plus 80 if not the first statement in a line.
8-9 Number of line containing the statement following FOR
A-B Address of beginning of program in progress

GOSUB return addresses are stacked beginning at 7AFF, proceeding in reverse direction through memory. Up to 33 return addresses may be pending (if the BASIC stack is not required for other purposes). Each GOSUB register contains 6 bytes, as follows:

0-1 Address of statement following GOSUB; plus 80 if not the first statement in a line.
2-3 Number of line containing the statement following GOSUB
4-5 Address of beginning of program in progress

The space between the FOR and GOSUB stacks comprises the arithmetic stack, which is used by the parsing routine in evaluating expressions. Pending numeric data (or string pointers) will be stacked (in 8-byte blocks) starting at 7A3C, or at the location four bytes after the last address used for storing a FOR register. Codes representing pending operations are stacked in reverse direction through memory, starting at 7AFF, or at the location just preceding the stacked GOSUB registers.

Each pending operation code contains two bytes, of which the first indicates the priority of the operation. Those representing functions consist of the token codes. (The tokens for SQR and PI represent the corresponding symbols.) The other pending operation codes follow:

84 5E	↑	70 51	OR
83 21	sign change	70 50	AND
82 2F	/	60 2C	comma separating arguments
82 2A	*		
81 2D	-	5A xx	variable name

81 2B	+
80 06	>=
80 05	<=
80 04	=
80 02	>
80 01	<
80 00	<>

59 xx	variable name
...	
41 xx	variable name
40 80	@(
20 28	left-parenthesis

SYSTEM RAM

7B00-7B07 contain the current value of the random number.

7B08 Unused
7B09 Counter for time that a key is held down
7B0A High byte, timed powerdown counter (counts from FE1D1D to FFFFFF)
7B0B Second byte, timed powerdown counter
7B0C Low byte, timed powerdown counter
7B0D Cursor Blink counter (counts from 80 to FF)
7B0E Cursor Control Parameter: 80 indicates down arrow key required to continue program execution. Other bits control cursor repeat.
7B0F Key matrix code for depressed key

STRING BUFFER

The 80-byte String Buffer, located 7B10-7B5F, contains the codes for character strings formed by the parsing of string expressions.

The String Buffer and Output Buffer together form a temporary storage area into which the previous contents of the Display Buffer are saved, during display of a program line at an execution halt, or during the display of a BREAK message or RESERVE template.

Certain printing routines use portions of the String Buffer as a scratch area. Address 7B1F is used as a flag, containing 00 for LLIST and FF for auto print of the Input Buffer contents in immediate mode.

OUTPUT BUFFER

The 80-byte Output Buffer is located 7B60-7BAF.

The codes for the characters to be displayed in response to a PRINT, PAUSE, or immediate calculation, are stored, in the required format, in addresses 7B60-7B79. (Codes for numeric output are first placed into the 7A10-34 area, then transferred here to construct the display; codes for string output are transferred from the string buffer.)

With LPRINT, the characters representing numeric data to be printed are stored, in the required format, in the area 7B80-7B9E. Another portion of the Output Buffer (ending 7B7F) serves as a scratch area used to form printed characters. Other locations are used as follows:

7BA9 A line feed (following printing) is specified by 04
7BAA USING editing character: 10, comma separation; 20, forced sign; 40, asterisk fill; 80, scientific
7BAB USING number of characters preceding decimal point
7BAC USING number of characters in string; 00 if unspecified
7BAD USING number of characters including and following decimal point
7BAE High byte, saved value of register Y
7BAF Low byte, saved value of register Y

Graphic printing routines use the area 7B92-7BAD to store coordinates of points specified by LINE or RLINE statements, as well as of the present location. The following addresses are also used:

7B83 Flag: 00, Line; FF, Box
7B84 Count of number of line segments to be drawn
7B85 Flag: 00, LINE; nonzero, RLINE

Prior to recording (or locating) a cassete file, a header is formed in the Output Buffer, located as follows:

7B60-67 Lead-in; consists of the bytes 10, 11, ..., 19
7B68 File type: 00, ML; 01, BASIC; 02, RESERVE; 04, data
7B69-78 File name; null bytes if unspecified
7B79-81 Unused
7B82-83 Beginning memory address of file; not used for data files
7B84-85 Number of bytes in file, less 1; zero for data file
7B86-87 Starting address for automatic execution, ML files only; if not specified, FFFF.

During a load operation, header information read from the recorded file is stored into the Output buffer as follows:

7A91-A0 File name
7AA1-A9 Unused
7AAA-AB Beginning memory address (used when CLOAD M statement does not specify an address)
7AAC-AD Number of bytes in file, less 1; zero for data file
7AAE-AF Starting address for automatic execution of ML Program

INPUT BUFFER

The 80-byte Input Buffer, located 7BB0-7BFF, holds the codes for characters as they are entered from the keyboard. It is also used for storing codes that form input prompts, displayed program lines, expressions to be evaluated in immediate mode, and displayed RESERVE key phrases.

USABLE ROM SUBROUTINES

The list which follows contains information related to a number of subroutines in ROM that are usable in user machine-language programs.

A subroutine that is addressable as a Vector Call will be listed as 'Call nn'; the address of the subroutine will follow, in parentheses.

CLEARING AND MOVING

BLOCKS OF MEMORY

Call BA (F763) clears the contents of memory from addresses X to X+UL. Register Y is not affected.

Subroutine D3C5 clears memory from addresses X to X+U; Register Y is not affected.

Subroutine D3C7 fills memory, from addresses X to X+U, with the byte contained in Register A. Register Y is not affected.

CLEARING OF BUFFERS

Call F2 (EE71) clears the Display Buffer (7600-764D and 7700-774D). Register Y is not affected.

Subroutine EF81 clears the Output Buffer (7B60-7BAF). Register Y is not affected.

Subroutine D02B fills the Input Buffer with 0D codes. Both Register Y and the Input Buffer Pointer (788B) are set to point to 7BB0, the beginning address of the Input Buffer.

DATA TRANSFERS TO BUFFERS

Call 94 (EC5C) copies the contents of addresses X to X+UL-1 into the Output Buffer, starting at the position specified by the Output Buffer Pointer (788F). If insufficient room is available, flag C will be set.

Subroutine EDC1 copies the contents of the Display Buffer into the area containing the String and Output Buffers. Subroutine EDD8 copies these codes back into the Display Buffer.

Subroutine FBCB, with C clear, copies the contents of the String Buffer into the Output Buffer.

Subroutine FBCB, with C set, copies the contents of the Output Buffer into the String Buffer.

FLOATING-POINT ARITHMETIC

The floating-point routines use the BASIC registers located at the following addresses:

R0:	7A00-7A07
R2:	7A10-7A17
R6:	7A30-7A37

Register R0 act as a floating-point accumulator, with R2 containing the second operand. Register R6 is used for temporary storage; except as noted, it is unaffected by the floating-point arithmetic routines.

The 8 bytes in a floating-point register are used as follows:

Byte 0	Exponent (in signed binary)
Byte 1	Sign: 00 = +, 80 = -
Byte 3	1st and 2nd mantissa digits (binary-coded decimal)
Byte 4	3rd and 4th mantissa digits
Byte 5	5th and 6th mantissa digits
Byte 6	7th and 8th mantissa digits
Byte 7	9th and 10th mantissa digits
Byte 8	11th and 12th mantissa digits

PRELOADING OF XH AND YH

In the case of certain indicated routines, XH and YH must contain 7A prior to execution of the routine.

Call 54 (F7B0) will load XH and YH with 7A.

Unless otherwise noted, both XH and YH will contain 7A following execution of any of the floating-point arithmetic routines listed.

DATA TRANSFER BETWEEN R0 AND FIXED VARIABLES

Variable into R0: subroutine DC20. Precede by loading X with initial address of variable. (Following execution, XH and YH will not necessarily contain 7A.)
R0 into variable: subroutine DC0C. Precede by loading X with initial address of variable. Register Y will be unchanged by this routine.

The initial addresses of fixed numeric variables are as follows:

A	7900	E	7920	I	7940	N	7968	R	7988	V	79A8
B	7908	F	7928	J	7948	O	7970	S	7990	W	79B0
C	7910	G	7930	K	7950	P	7978	T	7998	X	79B8
D	7918	H	7938	L	7958	Q	7980	U	79A0	Y	79C0
				M	7960					Z	79C8

CLEARING OF REGISTERS

Clear R0: Call EC (F757). Register Y is unchanged by this routine.
Clear R2: Subroutine F753. (XH and YH must contain 7A.)

DATA TRANSFER BETWEEN REGISTERS

R0 into R2: Call E6 (F70D).
R0 into R6: Call 80 (F707).

R2 into R0: Call 56 (F73D). (XH and YH must contain 7A.)
R2 into R6: Subroutine F701. (XH and YH must contain 7A.)

R6 into R0: Subroutine F737.
R6 into R2: Call 68 (F715). (XH and YH must contain 7A.)

Exchange R0, R2: Call 66 (F7B9). (XH and YH must contain 7A.)
Exchange R0, R6: Call 64 (F7B5). (XH and YH must contain 7A.)

DATA TRANSFER BETWEEN R0 AND BASIC STACK

The BASIC stack, located 7A3C-7AFF, permits up to 24 PUSHes of R0.

PUSH R0 onto BASIC stack: Subroutine DBF5.
POP R0 from BASIC stack: Call 30 (DC16)

ARITHMETIC OPERATIONS

All calculated results are normalized, with underflow to zero; each contains a 12-digit mantissa. Calculation of trigonometric functions and their inverses is based on the current trigonometric mode setting.

If an overflow results or an illegal operation is attempted, each of these routines will terminate with flag C set, and an error code in UH. Included are ERROR 37 (overflow), ERROR 38 (division by zero), and ERROR 39 (illegal function input argument).

R0+R2 into R0: Call F0 (EFBA)
R0-R2 into R0: Subroutine EFB6
R0*R2 into R0: Call 7E (F01A)
R0/R2 into R0: Call 58 (F084)
R0↑R2 into R0: Subroutine F89C (Register R6 used)

R0*R0 into R0: Subroutine F019
1/R0 into R0: Call 6E (F080). (XH and YH must contain 7A.)
SQR(R0) into R0: Subroutine F0E9

ABS(R0) into R0: Subroutine F597 (X, Y, and R2 will be unchanged.)
INT(R0)into R0: Subroutine F5BE
SGN(R0) into R0: Subroutine F59D (Y and R2 will be unchanged.)

Round R0 to ten digits: Subroutine F932 (Y and R2 will be unchanged.)

3.14159265359 (PI) into R0: Subroutine F5B5. (PI also stored into R2)

LN(R0) into R0: Subroutine F161
LOG(R0) into R0: Subroutine F165
EXP(R0) into R0: Subroutine F1CB
10↑R0 into R0: Subroutine F1D4

SIN(R0) into R0: Subroutine F3A2 (Register R6 used)
COS(R0) into R0: Subroutine F391 (Register R6 used)
TAN(R0) into R0: Subroutine F39E
ASN(R0) into R0: Subroutine F49A (Register R6 used)
ACS(R0) into R0: Subroutine F492 (Register R6 used)
ATN(R0) into R0: Subroutine F496 (Precede by clearing flag C)

DEG(R0) into R0: Subroutine F531 (Register R6 used)
DMS(R0) into R0: Subroutine F564 (Register R6 used)

COMPARISONS

Subroutine D0D2 performs a comparison of R0 and R2. The comparison performed will depend on the contents of A, as follows:

A=00:	R0<>R2	A=04:	R0=R2
A=01:	R0<R2	A=05:	R0<=R2
A=02:	R0>R2	A=06:	R0>=R2

If the test condition is met, R0 will contain 1, with flag Z clear; otherwise, R0 will contain zero, with flag Z set. Following execution, YH will not necessarily contain 7A.

CONSTANTS (FROM ROM TABLE) INTO R2

In each routine, XH and YH must contain 7A prior to execution.

1 into R2: Call 6A (F88F)
3.14159265359 (PI) into R2: Subroutine F875

57.2957795131 (180/PI) into R2: Subroutine F866
0.434294481903 (1/LN 10) into R2: Subroutine F87B
0.6 into R2: Call 62 (F88B)
0.9 into R2: Subroutine F87F
90 into R2: Subroutine F883
180 into R2: Subroutine F887

SEPARATION OF INTEGER AND FRACTION

Call 60 (F6B4) will separate the integral and fractional parts of R0, placing the integral part into R0 and the fractional part into R2. Neither result will be normalized. The given value of R0 is taken to be positive. XH and YH must contain 7A prior to execution.

REMOVAL OF SIGN

Call 6C (F6FB) loads the sign byte of R0 into A, and makes R0 positive. XH and YH must contain 7A prior to execution.

NORMALIZATION OF R0

Either of the subroutines Call E8 (F661) or Call 52 (F663) will normalize the contents of R0. The former applies a positive sign to the result; the latter takes the contents of A as the sign byte. In both routines, underflow results in zero; overflow sets flag C, with the code for ERROR 37 in Register UH.

Register XH must contain 7A prior to execution. Register Y is not changed by either routine.

BINARY-TO-DECIMAL CONVERSION

A two-byte binary number contained in Register U is converted to its decimal equivalent by execution of Call 10 (DD2D). The instruction calling this subroutine must be followed by a data byte, which specifies the effect produced. Two options exist: the converted value may be stored into R0; or, character codes for its display may be stored into memory, beginning at the address contained in Register Y.

Data byte	Interpretation of U	Disposition of result
00	unsigned binary	stored into R0
40	unsigned binary	characters stored, no sign
60	unsigned binary	characters stored, with sign
80	signed binary	stored into R0
C0	signed binary	characters stored, no sign
E0	signed binary	characters stored, with sign

Register YH will not necessarily contain 7A following execution.

DECIMAL-TO-BINARY CONVERSION

The binary equivalent of the contents of R0 (ignoring fractions) is calculated and stored into Register U by Call D0 (D0F9). The result is also stored into R0 (in binary format). A will equal UL.

Two data bytes must follow the instruction calling this subroutine.

The first of these specifies the range of values permitted; the second is added to the return address when the contents of R0 are not within that range, in which case UH will contain the code for ERROR 19.

Tabulated below are the permitted ranges (in decimal) for given values of the first passed byte. (When negative values are permitted, results are in signed binary.)

00:	0 to 65535	0A:	0 to 155
02:	0 to 65279	0C:	0 to 80
04:	-32768 to 32767	0E:	0 to 26
06:	0 to 32767	10:	0 to 25
08:	0 to 255		

If any of the above passed bytes is increased by 1, zero will be eliminated as a permitted value.

BINARY ROUTINES

STORAGE AND RECALL OF TWO-BYTE DATA

Call F6 (DDB5), followed by data bytes nn nn, stores UH into address nnnn and UL into nnnn+1. Following execution, A will contain UL; X will contain nnnn+1; Y and U are unchanged.

Call F4 (DBBC), followed by data bytes nn nn, loads UH with the contents of nnnn and UL with the contents of nnnn+1. A will contain the same byte as UH; X and Y are unchanged.

Call CA (C001), followed by data byte nn, stores XH into address 78nn and XL into 78nn+1. Following execution, A will contain XL; U will contain 78nn+1; X and Y are unchanged.

Call CC (DDC8), followed by data byte nn, loads XH with the contents of 78nn and XL with the contents of 78nn+1. A will contain the same byte as XH; Y and U are unchanged.

OPERATIONS IN BINARY

Subroutine DAA8 will replace U by its complement. (This is equivalent to replacing U by &10000-U.) X and Y are unchanged.

Subroutine DFE2 replaces U by U-X. If the result is negative, flag C is cleared, with UH containing 16. X and Y are unchanged.

Call 50 (DA71) replaces Y by U*Y; the result is also placed into X. If the result exceeds FFFF, flag C is set. Contents of U are lost.

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

algorithm for producing the reduced matrix which does not guarantee the correct ordering of the rows. The error trap is therefore incorrect. Also, their matrix reduction is halted when n-1 of the n variables are discovered; this leads to values for X(n) which do not necessarily satisfy the n'th equation! These serious flaws render this program almost useless. If the program signals "NOT SOLVABLE", this may not be the case. If it does yield a solution, this solution should be checked in all the equations.

10. Non-Linear Systems of Equations. Two non-linear systems in two unknowns are solved by Newton's approximation method. The user enters the functions, their partial derivatives, and a degree of precision. Enter the functions as the instructions on page 85 of the manual show, not as in the examples.

11. Integration. To avoid a common problem of numerical integration, investigate the asymptotic nature of the function being integrated. For example, y = 1/(x-1)^2 integrated from 0 to 2 will give an error n=5, but a numerical value when n=10, while the true value is infinite.

12. Cubic Spline Interpolation. The interpolation point x must be intermediate between x(1) and x(n). Also, the entry of the value y" at x(1) and x(n) are crucial. They should not simply be taken to zero, as in the examples.

It should be noted that the 2047 bytes of RAM are used extensively by a number of the programs. Functions are entered as program lines with confusing labels. Why couldn't a simple parsing routine have been used for function input, as Radio Shack did in its Plotter program? The manual lacks documentation as to variable usage and uses conflicting notation. The Mathematics Library often misses its mark as a useful tool.

ADDRESSING PIXELS ON THE PC-1500

This program was submitted by: *Patrick L. Pollet, University of Petroleum and Minerals, Department of Chemistry, UPM Box 298, Dhahran International Airport, P.O. Box 144, Saudi Arabia.*

one of the nice features of the PC-1500 is the possibility of producing graphics using the GPRINT instruction. Almost unlimited applications could be thought of, such as graphics games, sophisticated messages or, more seriously, redefining the keyboard by adding new characters such as French accentuated vowels or Greek or Russian alphabets.

Other than "try-and-hit" techniques, the only way to get the GPRINT hexadecimal codes for a desired pattern is to draw it on a 7 by 156 grid and then translate into hexadecimal using the "bit pattern" of every column. That process is quite long and prone to errors.

The GPRINTAID ROUTINE provided here gives you the possibility of addressing every individual dot of the screen with a flashing cursor that can be moved in any of 4 directions by pressing the following numeric keys:

"4"	cursor left by one row
"2"	cursor down by one line
"6"	cursor right by one row
"8"	cursor up by one line

Program: *GPRINTAID*

```
3A90 58 7A 5A 00    3ADC 2C 93 29 B7    3B2C B5 40 AE 79    3B78 B7 3A 81 02
3A94 6A 9B B5 00    3AE0 0D 8B 6F B7    3B30 00 BD FF AE    3B7C B3 06 28 24
3A98 51 88 03 E9    3AE4 36 89 0F A5    3B34 79 01 91 8F    3B80 B9 0F BB 30
3A9C 78 75 00 B5    3AE8 78 75 F9 B3    3B38 9E 40 B7 35    3B84 B7 3A 81 02
3AA0 01 AE 79 00    3AEC 01 B7 9C 93    3B3C 89 07 15 AB    3B88 B3 06 2A 48
3AA4 BD FF AE 79    3AF0 3B AE 78 75    3B40 79 00 1E 9E    3B8C 7B 4A 7F B5
3AA8 01 48 74 4A    3AF4 9E 4D B7 34    3B44 3A B7 01 89    3B90 00 43 A4 0E
3AAC 00 5A 00 6A    3AF8 89 0D A5 78    3B48 07 15 A9 79    3B94 BE A7 81 B5
3AB0 9B 55 CD 88    3AFC 75 FB B1 01    3B4C 01 1E 9E 45    3B98 00 43 24 0E
3AB4 88 05 CD 8C    3B00 91 4C AE 78    3B50 9E 9C 5A 9B    3B9C BE A7 81 14
3AB8 A5 78 75 1A    3B04 75 9E 5E B7    3B54 15 89 03 56    3BA0 A7 79 00 8B
3ABC 15 AB 79 00    3B08 32 89 16 A5    3B58 9E 06 54 14    3BA4 12 A5 79 01
3AC0 FD 88 CD 88    3B0C 79 00 D9 B7    3B5C AE 79 00 BE    3BA8 B3 02 AE 79
3AC4 6A 20 BE AB    3B10 80 81 02 B5    3B60 B0 EB BE A9    3BAC 01 A7 79 F5
3AC8 2A 88 05 15    3B14 01 AE 79 00    3B64 D5 BF AB EF    3BB0 99 41 BE A9
3ACC A9 79 01 FD    3B18 BD FF AE 79    3B68 BE B7 3F 5A    3BB4 F1 9E 4A BE
3AD0 0A CD 88 6A    3B1C 01 91 76 9E    3B6C 00 E9 79 01    3BB8 B1 16 BE A9
3AD4 20 BE AB 2A    3B20 3A B7 38 89    3B70 00 55 2A F1    3BBC E4 BE AB E6
3AD8 88 05 BE E4    3B24 15 A5 79 00    3B74 89 0F BB 30    3BC0 BA A7 69 38
                    3B28 F9 D5 81 02
```

Once the desired dot has been reached it can be turned on by pressing the number "5" key or turned off by pressing the "shift" key (for correction). In these latter two cases, the cursor will then move automatically to the next dot below or to the top of the display if the screen's limits have been crossed. This feature allows the drawing of solid black area or the erasing of an entire screen area simply by keeping the keys down. Once the desired graphic pattern has been composed, pressing the "ENTER" key will cause the printer to produce the corresponding GPRINT hexadecimal codes for archiving or other uses. (The use of any other keys except "break" is ignored.)

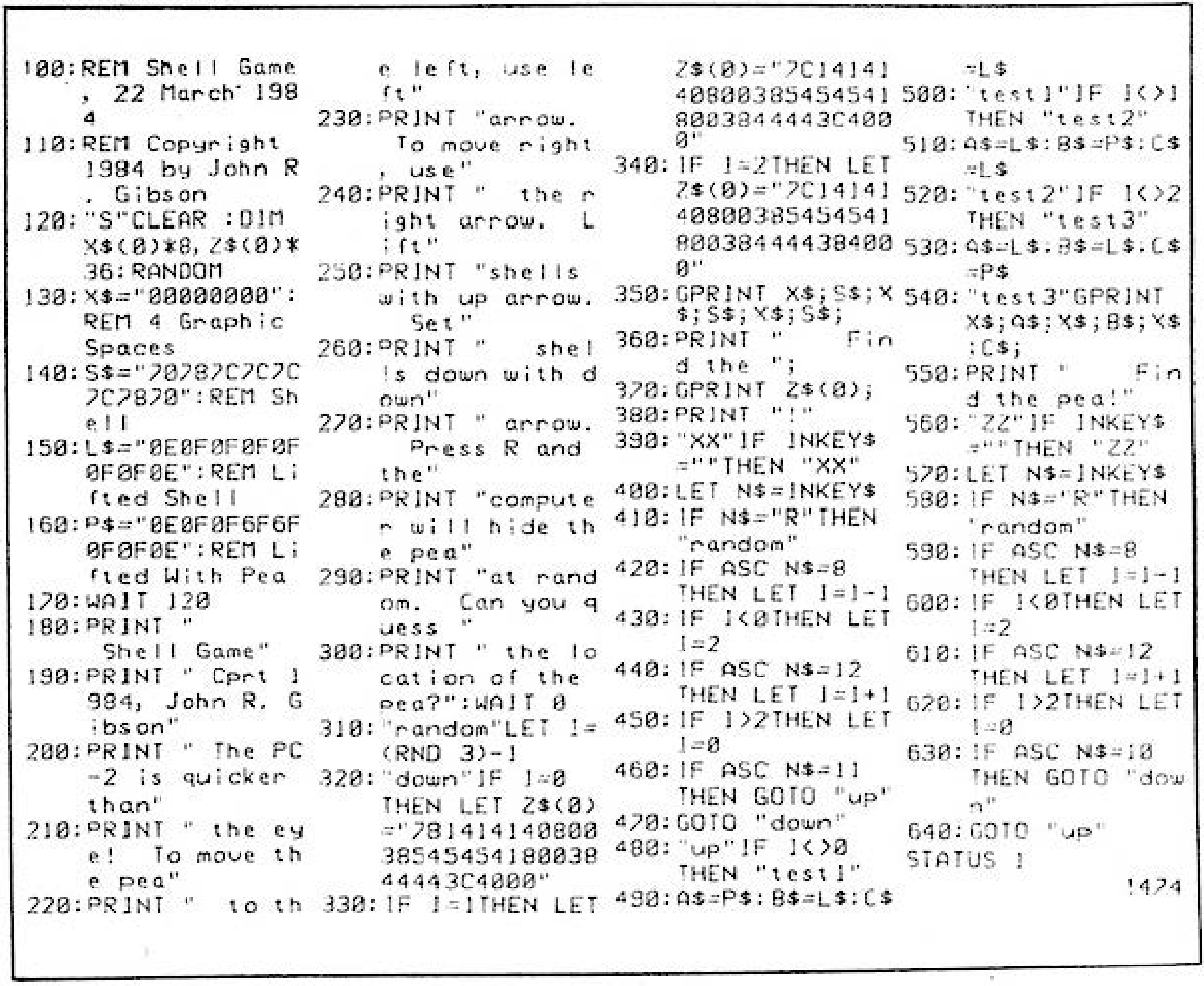
The routine is written entirely in machine code. This code is fully relocatable. After loading the 306 bytes in a convenient block of memory it is suggested that you protect it using the command NEW+START ADDRESS+307. You can then access the routine by using CALL ADDRESS or, even better, by using the following BASIC line:

1 CALL ADDRESS: WAIT: GPRINT: END

This line allows the pattern to stay on the screen after the displaying of the codes until you press the "ENTER" key again.

By the way, you can change the sensitivity of the keys (and the flashing rate) by modifying the value in register UL at the addresses &3AC5 and &3AD4 in the accompanying listing. (The value 20 is used at these points in the listing.)

Program *Shell Game.*



```
100:REM Shell Game
    , 22 March 198
    4
110:REM Copyright
    1984 by John R
    . Gibson
120:"S"CLEAR :DIM
    X$(0)*8, Z$(0)*
    36: RANDOM
130:X$="00000000":
    REM 4 Graphic
    Spaces
140:S$="70787C7C7C
    7C7870":REM Sh
    ell
150:L$="0E0F0F0F0F
    0F0F0E":REM Li
    fted Shell
160:P$="0E0F0F6F6F
    0F0F0E":REM Li
    fted With Pea
170:WAIT 120
180:PRINT "
     Shell Game"
190:PRINT " Cprt 1
    984, John R. G
    ibson"
200:PRINT " The PC
    -2 is quicker
    than"
210:PRINT " the ey
    e!  To move th
    e pea"
220:PRINT "  to th
    e left, use le
    ft"
230:PRINT "arrow.
     To move right
    , use"
240:PRINT "  the r
    ight arrow.  L
    ift"
250:PRINT "shells
    with up arrow.
      Set"
260:PRINT "   shel
    ls down with d
    own"
270:PRINT " arrow.
      Press R and
    the"
280:PRINT "compute
    r will hide th
    e pea"
290:PRINT "at rand
    om.  Can you g
    uess "
300:PRINT " the lo
    cation of the
    pea?":WAIT 0
310:"random"LET I=
    (RND 3)-1
320:"down"IF I=0
    THEN LET Z$(0)
    ="7814141408000
    3854545418003B
    44443C4000"
330:IF I=1THEN LET
    Z$(0)="7C14141
    4080038545454l
    80038444443C400
    0"
340:IF I=2THEN LET
    Z$(0)="7C14141
    408003854545 41
    8003844443840 0
    0"
350:GPRINT X$;S$;X
    $;S$;X$;S$;
360:PRINT "    Fin
    d the ";
370:GPRINT Z$(0);
380:PRINT "!"
390:"XX"IF INKEY$
    =""THEN "XX"
400:LET N$=INKEY$
410:IF N$="R"THEN
    "random"
420:IF ASC N$=8
    THEN LET I=I-1
430:IF I<0THEN LET
    I=2
440:IF ASC N$=12
    THEN LET I=I+1
450:IF I>2THEN LET
    I=0
460:IF ASC N$=11
    THEN GOTO "up"
470:GOTO "down"
480:"up"IF I<>0
    THEN "test1"
490:A$=P$:B$=L$:C$
    =L$
500:"test1"IF I<>1
    THEN "test2"
510:A$=L$:B$=P$:C$
    =L$
520:"test2"IF I<>2
    THEN "test3"
530:A$=L$:B$=L$:C$
    =P$
540:"test3"GPRINT
    X$;A$;X$;B$;X$
    ;C$;
550:PRINT "    Fin
    d the pea!"
560:"ZZ"IF INKEY$
    =""THEN "ZZ"
570:LET N$=INKEY$
580:IF N$="R"THEN
    "random"
590:IF ASC N$=8
    THEN LET I=I-1
600:IF I<0THEN LET
    I=2
610:IF ASC N$=12
    THEN LET I=I+1
620:IF I>2THEN LET
    I=0
630:IF ASC N$=10
    THEN GOTO "dow
    n"
640:GOTO "up"
STATUS 1
                1474
```

SHELL GAME

This program was submitted by *John R. Gibson, 924 Chapman Drive #9, Colorado Springs, CO 80916.* He explains it in the following manner.

Here is a simulation of the old shell game. The arrows on the PC-2 are used to move the pea and shells. Can you keep up? If you are in on the secret you will always know the location of the pea!

The Effect

You can always find the hidden pea whether it is hidden by a spectator or at random by the computer.

The Presentation

There are two ways to present the effect. First, the pea can be hidden manually so that the magician must "guess" its location. (The other players can have you leave the room or turn away while they shuffle the pea around.) Alternately, press R to have the computer randomly select a hiding place. Either way, you as the magician will not be fooled.

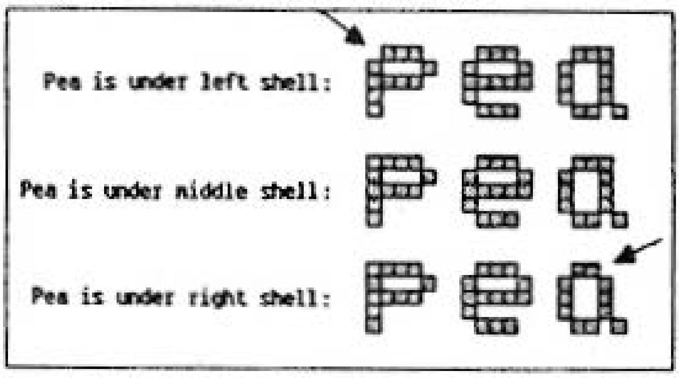
Press DEF/S to execute the program after it has been loaded. A short set of instructions is followed by a display of three shells. To lift the shells and view the pea, push the up arrow. To move the pea right use the right arrow. Use the left arrow to move the pea in that direction. After you have positioned the pea where you want it, use the down arrow to cover the pea.

Note: The program is running continuously in order to read these control keys. Thus, to stop you *must* use the break key.

The Secret

In this trick, the computer is your accomplice. It signals the location of the pea using the word "pea" in the display. If the upper left corner of the letter "p" is missing, then the pea is on the left. If the pea is in the center, no signal is given. When the pea is on the right, the upper right corner of the a is gone.

Diagram *Secret of the Shell Game!*



BINARY MAGIC

This program was submitted by *John R. Gibson, 924 Chapman Drive #9, Colorado Springs, CO 80916.* John has the following to say about his program.

This program can be enjoyed on three levels. To the computer newcomer, this program can be presented as a magic trick. It is impressive both visually and intellectually. On the second level the student will find it a practical demonstration of the binary numbering system. Play around with it for awhile and you will have a much better understanding of how this counting system works. Even advanced computerists will be amused by the program as it is unsurpassed as an example of PC graphics.

The Effect

A spectator selects a graphics object from those displayed (16 total). The person then answers four questions (truthfully) with a yes or no. The psychic computer then reveals the object the spectator had in mind.

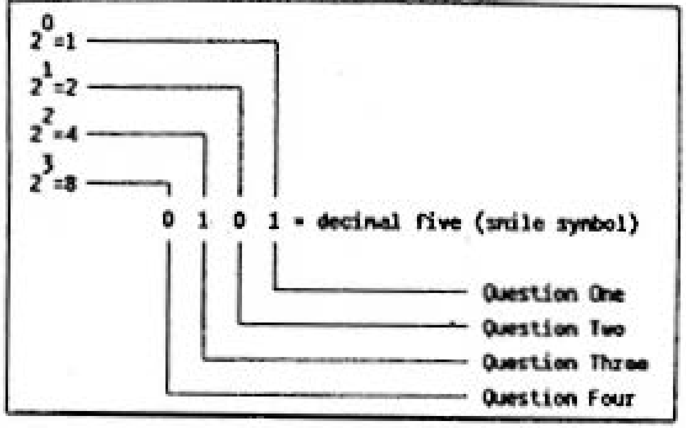
The Presentation

Load the program and hit DEF/B to execute. Some instructions will be displayed, followed by the graphics symbols. Choose one. Press the ENTER button. The computer will ask whether or not your symbol is among the ones displayed. If it is, press Y and the ENTER key. If not, then press N and the ENTER key or alternately just press ENTER. This process is repeated three more times after which the computer reveals the object that was selected by the spectator! To do the trick again, press ENTER.

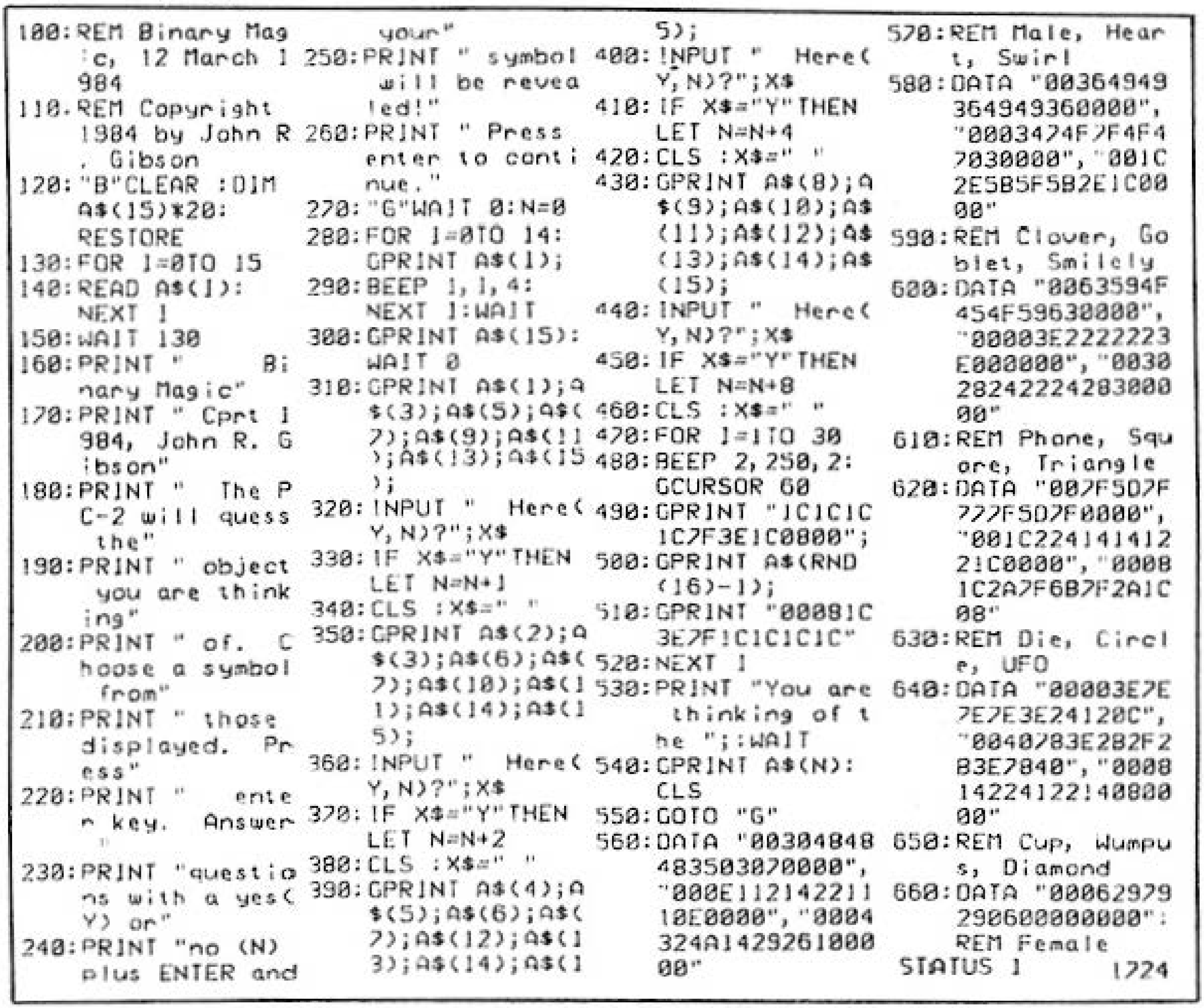
The Secret

The trick is self-working. Each object is represented by a binary number from 0000 (male symbol) to 1111 (female symbol) where 0 = no and 1 = yes. See the accompanying diagram.

Diagram *The Secret of Binary Magic!*



Program *Binary Magic.*



```
100:REM Binary Mag
    ic, 12 March 1
    984
110.REM Copyright
    1984 by John R
    . Gibson
120:"B"CLEAR :DIM
    A$(15)*20:
    RESTORE
130:FOR I=0TO 15
140:READ A$(I):
    NEXT I
150:WAIT 130
160:PRINT "     Bi
    nary Magic"
170:PRINT " Cprt 1
    984, John R. G
    ibson"
180:PRINT "  The P
    C-2 will quess
     the"
190:PRINT " object
     you are think
    ing"
200:PRINT " of.  C
    hoose a symbol
     from"
210:PRINT " those
    displayed.  Pr
    ess"
220:PRINT "    ente
    r key.  Answer
     "
230:PRINT "questio
    ns with a yes(
    Y) or"
240:PRINT "no (N)
    plus ENTER and
    your"
250:PRINT " symbol
     will be revea
    led!"
260:PRINT " Press
    enter to conti
    nue."
270:"G"WAIT 0:N=0
280:FOR I=0TO 14:
    GPRINT A$(I);
290:BEEP 1,1,4:
    NEXT I:WAIT
300:GPRINT A$(15):
    WAIT 0
310:GPRINT A$(1);A
    $(3);A$(5);A$(
    7);A$(9);A$(11
    );A$(13);A$(15
    );
320:INPUT "  Here(
    Y,N)?";X$
330:IF X$="Y"THEN
    LET N=N+1
340:CLS :X$=" "
350:GPRINT A$(2);A
    $(3);A$(6);A$(
    7);A$(10);A$(1
    1);A$(14);A$(1
    5);
360:INPUT "  Here(
    Y,N)?";X$
370:IF X$="Y"THEN
    LET N=N+2
380:CLS :X$=" "
390:GPRINT A$(4);A
    $(5);A$(6);A$(
    7);A$(12);A$(1
    3);A$(14);A$(1
    5);
400:INPUT "  Here(
    Y,N)?";X$
410:IF X$="Y"THEN
    LET N=N+4
420:CLS :X$=" "
430:GPRINT A$(8);A
    $(9);A$(10);A$
    (11);A$(12);A$
    (13);A$(14);A$
    (15);
440:INPUT "  Here(
    Y,N)?";X$
450:IF X$="Y"THEN
    LET N=N+8
460:CLS :X$=" "
470:FOR I=1TO 30
480:BEEP 2,250,2:
    GCURSOR 60
490:GPRINT "1C1C1C
    1C7F3E1C0800";
500:GPRINT A$(RND
    (16)-1);
510:GPRINT "00081C
    3E7F1C1C1C1C"
520:NEXT I
530:PRINT "You are
     thinking of t
    he ";:WAIT
540:GPRINT A$(N):
    CLS
550:GOTO "G"
560:DATA "00304848
    483503070000",
    "000E11214221 1
    10E0000","0004
    324A1429261000
    00"
570:REM Male, Hear
    t, Swirl
580:DATA "00364949
    364949360000",
    "0003474F7F4F4
    7030000","001C
    2E5B5F5B2E1C00
    00"
590:REM Clover, Go
    blet, Smilely
600:DATA "0063594F
    454F59630000",
    "00003E2222223
    E000000","0030
    28242224283000
    00"
610:REM Phone, Squ
    are, Triangle
620:DATA "007F5D7F
    777F5D7F0000",
    "001C22414141 2
    21C0000","0008
    1C2A7F6B7F2A1C
    08"
630:REM Die, Circl
    e, UFO
640:DATA "00003E7E
    7E7E3E24120C",
    "0040783E282F2
    83E7840","0008
    14224122140800
    00"
650:REM Cup, Wumpu
    s, Diamond
660:DATA "00062979
    29060000000":
    REM Female
 STATUS 1        1724
```

INSIDE THE PC-1500

This is the third part of the current series on the Sharp PC-1500. The information is provided through the efforts of *Norlin Rober, 407 North 1st Avenue, Marshalltown, IA 50158.* We pick up in this issue from where we stopped in *PCN* Issue 32. That is with a continuation of synopsis of a number of the routines contained in the ROM of the PC-1500. (Of course, virtually all of this information is applicable to the Radio Shack PC-2). These routines are capable of performing many useful functions when called as machine language subroutines, etc. Norlin has spent a great deal of time gleaning this information and providing an easy to read summary of their uses. The next page picks up where the previous article left off -- in the midst of presenting information on binary routines.

BINARY FORMAT IN R0

The following format is used for data in R0:

7A04	B2, identifying R0 as containing binary
7A05	First byte of data
7A06	Second byte of data

Data in this format may not be used as input for floating-point arithmetic routines. It may, however, constitute the contents of a variable; BASIC routines will correctly interpret it.

Subroutine D9E7 stores the contents of U into R0, in binary format. Following execution, A will contain UL. Y and U are unchanged.

Subroutine D9E3 stores the byte pointed to by X into R0, in binary format. Following execution, UL will contain that same byte; UH will contain 00. Y is unchanged.

CHARACTER STRING ROUTINES

STRING POINTER

A string pointer in R0 has the following format:

7A04	D0, identifying R0 as a string pointer
7A05	High byte, beginning address of string
7A06	Low byte, beginning address of string
7A07	Number of characters in string

Call 24 (DEAF) stores a string pointer into R0, taking X as the beginning address, and A as the length of the string. Y is unchanged.

Call DC (DEBC) loads X with the address, and UL and A with the length of the string pointed to by R0. Register Y is unchanged.

Subroutine D048 will copy the string beginning at X, containing UL characters, into memory beginning at address Y. After execution, Y will contain the address following that of the last character stored.

EQUIVALENTS OF BASIC FUNCTIONS

(1) String input, numeric output. Prior to execution, R0 must contain a pointer to the string; the numeric result will be placed into R0.

(a) ASC: Subroutine D9DD. Precede by loading YL with 60. Result will be in binary format.
(b) LEN: Subroutine D9DD. Precede by loading YL with 64. Result will be in binary format.
(c) VAL: Subroutine D9D7. Result will be in decimal.

(2) Numeric input, string output. Prior to execution, R0 must contain numeric data (either binary or decimal format), and 7894 must contain 10. Following execution, R0 will contain a pointer to the resulting string, and UH will contain either 00 or an error code.

(a) CHR$: Subroutine D9B1. (C1 is placed into 7A04, instead of D0)
(b) STR$: Subroutine D9CF.

STRING COMPARISONS

Subroutine D0F9 will compare the two strings indicated by string pointers in R0 and R2 (here designated as R0$ and R2$). The comparison performed will depend on the contents of A, as follows:

A=00:	R0$<>R2$	A=04:	R0$=R2$
A=01:	R0$<R2$	A=05:	R0$<=R2$
A=02:	R0$>R2$	A=06:	R0$>=R2$

If the test condition is met, R0 will contain 1 with flag Z clear; otherwise, R1 will contain 0, with flag Z set. R2 will be unchanged. (NOTE: Comparisons using A=5 or 6 will not work with ROM version A01.)

VARIABLES USED BY BASIC

LOCATING A NAMED VARIABLE

A variable specified by its name may be located by Call 0E (D461). Preceding execution, the variable sought must be specified, according to the following rules:

(1) UH must contain the ASCII code for the first character (A to Z) of the name of the variable. (@( ) and @$( ) may not be used here.)
(2) For one-character names, UL must contain zero. For two-character names, UL must contain the ASCII code for the second character--except that when the second character is 0 to 9, UL should contain 10 to 19.
(3) If a string variable is specified, UL must be increased by 20; if a dimensioned variable, by 80.
(4) When a single subscript is used, 01 must be stored into 788C, and R0 must contain the value of the subscript (in decimal).
(5) When a double subscript is used, 02 must be stored into 788C. The first subscript (in decimal) must be pushed into the BASIC data stack; the second subscript (in decimal) must be contained in R0.

Two data bytes must follow the instruction calling this subroutine. The first of these two passed bytes should be 5A. Following execution of Call 0E, U will contain the beginning address of the variable. Also, R0 will contain a variable pointer, having the following format:

7A04	D0
7A05	High byte of variable address
7A06	Low byte of variable address
7A07	Length of string variable; 88 if numeric

If a subscript is too large, or a subscripted variable was not previously dimensioned, the return address is incremented by the second passed byte, with UH containing an error code. If an non-subscripted previously-undefined variable was sought, 7A05-7A06 will contain the codes for the variable name, with 80 added to the first code.

LOADING R0 WITH VARIABLE CONTENTS

Call 0E (D461) may be used to load R0 with the contents of a variable specified by name. Preceding execution, the variable sought must be specified according to the same rules used above.

The first of two passed bytes must be 52. Following execution, R0 will contain numeric data or a pointer to a string, as determined by the variable specified.

If a subscript is too large, or a subscripted variable was not previously dimensioned, the return address is incremented by the second passed byte, with UH containing an error code. If an non-subscripted previously-undefined variable was sought, R0 will contain zero.

CLEARING VARIABLES

Subroutine D080 clears all BASIC variables. No change in Y.

Subroutine D091 clears Main-memory variables only, by resetting the Start of Variables pointer (7899-789A). No change in X, Y, or U.

TIME and RANDOM NUMBER

TIME (FORMAT USED BY BASIC)

Subroutine DE82 copies TIME into R0, in the same format used by BASIC. Registers XH and YH will contain 7A following execution.

Subroutine DE1D stores R0 into TIME, with R0 given in the same format used by BASIC. One data byte is passed to this subroutine; it is added to the return address if R0 specifies TIME incorrectly.

TIME (ALTERNATE FORMAT)

Data may be transferred between R0 and the timer with the following format used in R0:

7A00	Not used
7A01	Not used
7A02	1st nybble=month number (hex); 2nd nybble=week day (0 to 6)
7A03	Date of month (decimal)
7A04	Hour (0 to 23, decimal)
7A05	Minute (0 to 59, decimal)
7A06	Second (0 to 59, decimal)
7A07	Not used

Subroutine E59A copies R0 into TIME; X and Y are unchanged.

Subroutine E5B4 copies TIME into R0; X and Y are unchanged.

RANDOM NUMBER

Subroutine F5DD places RND (R0), interpreted as in BASIC, into R0. The random number is replaced by its next value. Both R2 and R6 are used; XH and YH will contain 7A following execution.

Call 5C (F61B) replaces the random number (located 7B00-7B07, in decimal format) by its next value. Precede by loading XH and YH with 7A. Following execution, XH and YH will contain 7A. The contents of both R0 and R2 will be changed.

READING CODES FROM BASIC

EVALUATION OF EXPRESSION IN BASIC

Call DF (D6DF) evaluates an expression specified by BASIC code, beginning at the address contained in Y. The result is placed into R0. The BASIC expression is terminated by any of the codes 00 (null), 0D (ENTER), 2C (comma), 3A (colon), or 3B (semicolon).

The instruction calling this subroutine must be followed by a data byte, which will be added to the return address (with UH containing an appropriate error code) if an error occurs.

Numeric results obtained from AND, OR, NOT, ASC, LEN, POINT, PEEK, or PEEK# are in signed binary; other numeric results are in decimal. String results are expressed as string pointers in R0, with the string itself located in the String Buffer.

CONVERSION OF HEX CHARACTERS TO BYTE

Subroutine ED95 loads A with a byte determined by a pair of codes for hex digits, located starting at address X. Codes for characters other than 0-9 or A-E will clear flag C, signaling an error. After execution, X will point to the next address; Y and U are unchanged.

SEARCH FOR BASIC PROGRAM LINE

Subroutine D2EA searches for the address in memory of the BASIC line whose number (in hexadecimal) is contained in U. The search will begin at the address indicated by the START OF BASIC (or START OF BASIC in ROM) pointer. It will end if a stop byte is encountered.

The same subroutine will search for the line with a specified label. In this case, UH must contain FF, and R0 must contain a string pointer pointing to the string forming the label sought. If necessary, a search for a label will bypass stop bytes, continuing until END OF BASIC is reached.

In either case, if the line sought is located, the address of the first statement in that line is placed into 78A6-A7. (The address at which the line's number begins is less than that address by 3.)

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