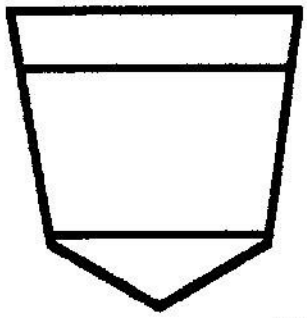
POCKET
COMPUTER
NEWSLETTER



Issue 37 - January

REVIEW OF THE SHARP PC-1350

The new Sharp PC-1350 is becoming available in the United States now. It is somewhat trimmer in size than the PC-1500 (about 3/8'ths of an inch less in length and width, and 1/4 inch less in thickness). It is equipped with a protective "half-shell" (in the same fashion as the PC-1261). When the unit is not in use, the shell protects the keyboard/display surface. When utilizing the PC, the shell is reversed and mounted on the bottom of the unit for safekeeping. Shell and all, the new PC measures approximately 7-1/4 inches in length, by 3 inches in width, and a bit of 3/4 of an inch in thickness. It is slim enough to fit in a typical shirt pocket, but its length would cause it to stick out somewhat if carried there.

The PC-1350 weighs in at about 9 ounces, quite a bit lighter than the 1500. Much of this decrease in weight is due to its lack of the 4 type AA batteries used in the PC-1500. These have been replaced by a pair of lithium cells (type CR-2032) that are said to be capable of powering the PC for approximately 250 hours of "normal" use.

The most striking feature of the new PC is its large, 4-line liquid-crystal (LCD) display. The display window is one by four inches in size, and occupies the upper-left portion of the keyboard surface when viewed from the normal operating position. The active pixel area within the display window measures 3/4 by 3-1/2 inches. It contains 150 columns and 32 rows of display elements. Individual dots (pixels) may be controlled to provide graphics. In the text mode, the pixels are used to form characters in a 5 by 7 matrix. Using such a matrix permits four lines of 24 characters-per-line to be displayed at one time. A contrast control on the right side of the PC gives a user the ability to adjust the display for comfortable viewing under various lighting conditions. A number of annunciators appear along the left border of the pixel matrix. These annunciators serve as mode and status indicators.

There are three rows of keys directly below the display. Two of these contain 11 keys, the third contains 10 with the ENTER key occupying the equivalent of two key positions. These keys represent the 26 letters of the alphabet, plus the SHIFT, DEF, SML, SPC, = and ENTER operations. The SML key permits selecting upper and lower case letters. The SHIFT key is used to provide alternate characters for some of the keys. These alternate characters consist primarily of punctuation marks (such as: ! " # $ % $?) and special mathematical operators (including < and >).

The right side of the keyboard holds a 4 by 5 matrix of keys (larger than the alphabetical keys) containing the numbers 0 - 9, the four primary mathematical operators (+, -, *, and /) and commonly used punctuation (period, comma, colon, semicolon and parenthesis). Above the 4 by 5 matrix are ten more keys used for controlling the display, selecting modes, etc., and a sliding on/off switch.

One marked improvement (over the PC-1500) in regards to the keyboard is that commonly used editing operations no longer require the use of the SHIFT key. For instance, you can insert and delete characters by simply pressing the INS and DEL editing keys. Furthermore, the punctuation marks that are frequently used in programming (colon, semicolon, comma and parenthesis) have their own keys and do not require the use of the SHIFT key. This considerably eases the task of keying in BASIC programs.

The PC-1350 contains about 3K of user memory (RAM). This does not seem like much for a "third-generation" PC. But, you can expand memory by installing a thin RAM-card in the back of the unit. A CE-201M card provides 8K of additional RAM. A model CE-202M card gives an additional 16K. However, these battery-backed expansion memories are not cheap. Buying a 16K card will almost double the basic price of a PC-1350 at the present time. There are, however, some nice features about these memory cards. First, they can maintain programs and/or data when removed from the unit. (Unlike the modules for the PC-1500 which were designed strictly for the storage of programs when removed from the PC.) The structure of these modules is such that you can switch programs and data bases by swapping in a new card (and without performing a lot of fancy procedures). You can elect to have the 3K of user memory within the PC-1350 tacked onto the end of the memory in the RAM card (and thus use it for variables storage) or to be ignored. The former option means you can use the memory card primarily for program storage (while using the 1350's RAM for data). The latter means you can install a complete self-contained software package (program and data) within a card and not be concerned about the boundary between the card's memory and that of the 1350's. (Software developers take note! This means you can develop complete packages without worrying about the types of complications that could appear when using expansion modules in the PC-1500!)

The PC-1350 contains 40K of ROM containing essentially the same type of BASIC interpreter as in the earlier Sharp 1250/1260 series. However, instead of features such as the "help" directory and "automatic calculations" found in the PC-1260/1261, the 1350's extra ROM is used to control the large display and provide capabilities for its extra input/output port.

What extra input/output port? Well, in addition to the port that is used to communicate with a CE-126P printer and cassette interface, the PC-1350 contains a second "serial communications" port! Yes, on the right hand end of the PC-1350 is a 15-pin connector that enables the 1350 to communicate with external devices. This port is designed to operate at logic levels of 0 and +5 volts. The port is programmable in BASIC and such parameters as: the baud rate (300 to 1200), number of bits-per-word, parity, number of stop bits, etc., can be set under program control. Stated another way, the PC-1350 has a great deal of the capability

of the earlier CE-158 communications interface built right into it! (This built-in communications capability is one reason the PC-1350's price is not quite as low as some people might have expected a third-generation PC to be.)

At the operational level, the PC-1350 is a lot more "confortable" PC to use than previous Sharp versions. It is apparent that Sharp has been doing its homework and listening to users to improve its products. Many of the nice touches are subtle but important. Just to give an idea of the kind of user-friendly type of operation implemented in this machine, consider how the PC-1350 handles a simple PRINT statement that contains a multi-line string of text. It utilizes automatic "word-wrap" so that when the PRINT statement is executed, the text is easy to read! Note that this is the BASIC interpreter itself that is doing the word-wrapping on the fly. The programmer does not have to be concerned about how a sentence will appear on the 4-line LCD. (Unless the programmer wants to over-ride word wrap. This is easy to do. Just use the CURSOR statement.)

Of course, having the big 4-line display on the PC adds to its general ease of operation. You can usually view most of the information on a program line (as the four lines show a total of 96 characters at a time). This, coupled with the single-function editing keys (character insert and delete), and the fact that punctuation commonly used in programming can be invoked without using a SHIFT key, makes the inputting of programs through the keyboard somewhat easier than on previous Sharp pocket computers.

As for speed of operation, the PC-1350 benchmarks at approximately the same rate as the PC-1500. Preliminary research indicates, however, that the PC-1350 has a different CPU than that used in the PC-1500. In fact, it appears to utilize the same type of machine codes as that utilized in the Sharp PC-1250 & PC-1260 series. If this proves out to be true, it could be interesting for people who like to program at the machine language level.

All-in-all, the new Sharp PC-1350 seems to offer significant potential. The display is large enough to be of practical value, say, in a light word processing (memo-taking) environment. It is possible to envision, for example, this little PC being connected (via the serial port) to a full-sized keyboard or a combination keyboard and printer. That same serial port could also be used for uploading and downloading files to some sort of mass storage device or a desktop computer. Placing a 16K RAM card in the unit gives close to 20K of user memory. That is enough to handle quite a few types of practical applications. While the RAM cards are currently rather expensive, the price may drop as volume picks up. Even at their current prices, many professionals may find it beneficial to keep their most frequently used application programs on the cards, ready for virtually instantaneous use.

If Sharp is planning a fourth-generation PC, they might want to consider providing an operating system that would permit manipulating multiple files in memory at one time. The capabilities of the Hewlett-Packard handhelds, such as the HP-71B, in this regards, have much to recommend them. Pocket computers are especially valuable in environments that call for many types of relatively small tasks to be performed. Switching between those tasks is currently the biggest chore. The Sharp method of merging files (so that multiple programs can be executed but not edited) and its limited ability to handle files of data puts it at a distinct disadvantage in the hotly competitive PC market. (The HP-71B clearly has them beat in this regards.) However, at less than half the price of a HP-71B, the new Sharp PC-1350 can certainly be considered a bargain.

FOR PC-1250/51/60/61 USERS

PAYROLL PROGRAM FOR 1250/51 AND 1260/61 POCKET COMPUTERS

Here we are at the start of a new (alas, taxable) year. Seems like a good time to update our old Payroll program to take account of the new (higher) FICA tax rate and adjusted witholding rates. The original version of our Payroll program appeared way back in Issue 03 of *PCN* and a modified version, reorganized thanks to the efforts of *David Hotto*, was published in Issue 05.

This updated version has been slightly customized for the PC-1250/51 (as well as PC-3) and the PC-1260/61 models. A variation for lines 40 - 50 is provided as an alternate means of programming. (Why not try both methods? Keep the one you like best!)

If you run a small business, the program can ease the weekly chore of calculating payroll deductions. Alternately, if you are employed by someone else, you can get a close approximation of what your employer should be witholding. This program is based on widely published tax-witholding tables. However, there can be a variations of a percent or two between various witholding methods. (Consult your personal tax-advisor for tax advice. *PCN* provides this program without warranty of any kind and is not engaged in the practice of issuing tax advice.)

Program *Payroll Deductions.*

```
10:"$" USING "#####.##"
   :H=100
20:G=0: INPUT "ENTER GR
   OSS PAY? ";G
30:E=0: INPUT "NR. OF E
   XEMPTIONS? ";E
40:WAIT 0: PRINT "SINGL
   E(S)/MARRIED(M)? ":M
   $= INKEY$ : IF M$=""
   THEN 40
45:IF M$="S" LET M=0:
   GOTO 60
50:IF M$="M" LET M=1:
   GOTO 60
55:BEEP 1: GOTO 40
60:WAIT :X=0:T=G-20*E:
   GOSUB 600+200*M
70:F=.0705*G
80:X= INT (H*X+.5)/H:
   PRINT "FWT= ";X
90:F= INT (H*F+.5)/H:
   PRINT "FICA= ";F
```

```
100:Z=0: INPUT "OTHER WI
    THOLDINGS? ";Z
110:Z= INT (H*Z+.5)/H
120:G=G-X-F-Z: PRINT "NE
    T PAY= ";G
130:GOTO 20
600:IF T>=663 LET X=150.
    50+.37*(T-663):
    RETURN
610:IF T>=556 LET X=114.
    12+.34*(T-556):
    RETURN
620:IF T>=440 LET X=79.3
    2+.30*(T-440):
    RETURN
630:IF T>=292 LET X=42.3
    2+.25*(T-292):
    RETURN
640:IF T>=185 LET X=21.9
    9+.19*(T-185):
    RETURN
650:IF T>=84 LET X=6.84+
    .15*(T-84): RETURN
660:IF T>=27 LET X=.12*(
    T-27)
670:RETURN
800:IF T>=897 LET X=195.
    75+.37*(T-897):
    RETURN
810:IF T>=684 LET X=125.
    46+.33*(T-684):
    RETURN
820:IF T>=578 LET X=95.7
    8+.28*(T-578):
    RETURN
830:IF T>=472 LET X=69.2
    8+.25*(T-472):
    RETURN
840:IF T>=384 LET X=49.9
    2+.22*(T-384):
    RETURN
850:IF T>=192 LET X=17.2
    8+.17*(T-192):
    RETURN
860:IF T>=48 LET X=.12*(
    T-48)
870:RETURN
```

Alternate Method for Lines 40 - 50

```
40:WAIT 0:M=0: INPUT "S
   INGLE(S)/MARRIED(M)?
    ";A$
45:IF A$="S" LET M=0:
   GOTO 60
50:IF A$="M" LET M=1:
   GOTO 60
```

Program Memory Dump.

```
10:DIM A$(0)*16: INPUT
   "STARTING ADDRESS? "
   ;A
15:INPUT "ENDING ADDRES
   S? ";G: PRINT =
   PRINT
20:A$(0)="0123456789ABC
   DEF"
25:WAIT :W= INT (A/4096
   )
26:X= INT ((A-(W*4096))
   /256)
27:Y= INT ((A-(W*4096)-
   (X*256))/16)
28:Z= INT ((A-(W*4096)-
   (X*256)-(Y*16)))
29:PRINT MID$ (A$(0),W+
   1,1); MID$ (A$(0),X+
   1,1); MID$ (A$(0),Y+
   1,1); MID$ (A$(0),Z+
   1,1)
30:WAIT 0: FOR B=0 TO 7
   :C=( PEEK (A+B)) AND
   (&F0):D=1+ INT (C/16
   ):E=( PEEK (A+B))
   AND (&0F)
35:IF B=7 WAIT
40:PRINT MID$ (A$(0),D,
   1); MID$ (A$(0),E+1,
   1);" ";
45:NEXT B
48:F= INT (A/64):A=A+8:
   IF A>G THEN STOP
49:IF INT (A/64)<=F
   THEN 30
50:GOTO 25
```

MEMORY DUMP PROGRAM

Wow! User interest and excitement in the Sharp PC-1260 & PC-1261 models is really heating up. The discovery that the models contain the undocumented commands PEEK and POKE has lead to a new round of detective work on the CPU instruction set. Wouldn't you know it, urged on by such pocket computing pioneeers as Norlin Rober (of PC-1500 fame), others have been rapidly cracking the machine code used in the newer PCs. (It appears that the PC-1260 and PC-1261, as well as possibly the new PC-1350, use the same CPU as the 1250/51.) We will have detailed reports of this ongoing work in future issues of PCN.

Meanwhile, if you want to do some exploring on your own, we have adapted a small memory dump program (from Issue 26 of *PCW*) for use on 1260/1261 units. The adaptation is from a routine originally crafted by *Merlin Rober* for use on the PC-1500. A few "bells and whistles" have been added to make it a little easier to use. Note that line 15 contains a "PRINT=" statement. Set this statement to read PRINT=PRINT if you want the dump to the liquid-crystal display. Set it to read PRINT=LPRINT if you want the dump listed on a CE-125 or

CE-126P printer.

When you start this memory dump program, it will ask for starting and ending addresses. You can enter these in decimal format. Alternately, you can work in hexadecimal by preceding the values with the ampersand (&) sign.

If you are using the LCD for a dump, you need to press the ENTER key after each line of information has been displayed. Note that the memory is dumped in blocks of 64 hexadecimally-formatted bytes. Each group is preceded by the starting address (using hexadecimal notation) of that group.

When using the printer, the dump proceeds automatically until the block requested has been outputted. Groups of 64 bytes are separated by a line containing an address value.

You can use this memory dump program to start exploring your PC. On the PC-1261 it appears that some RAM is located starting at decimal address 16384 (&8000 hexadecimal). Formulas assigned by a user for use with the "automatic calculation mode" appear to be stored here.

This memory dump program should be readily adaptable to the PC-1250/51 models. Remove the PRINT= statement in line 15. Change the PRINT statements to LPRINT if you want to use a printer. (These alterations will be required on a 1250/51 since it does not support the PRINT= command.)

FOR HEWLETT-PACKARD HP-71B USERS

Program Payroll Deductions.

```
10 DELAY 8,.5 @ WIDTH 96 @ DESTROY ALL @ FIX 2
20 A$=KEY$ @ INPUT "ENTER GROSS PAY: ","0";G
30 INPUT "NR. OF EXEMPTIONS? ","0";E
40 INPUT "(S)INGLE/(M)ARRIED? ","S";M$
50 IF M$="S" OR M$="M" THEN 60 ELSE BEEP @ GOTO 40
60 X=0 @ T=G-20*E @ GOSUB M$
70 F=.0705*G
80 PRINT "FWT= ";X
90 PRINT "FICA= ";F
100 A$=KEY$ @ INPUT "OTHER WITHOLDING? ","0";Z
110 G=G-X-F-Z
120 PRINT "NET PAY= ";G
130 GOTO 20
600 'S': IF T>=663 THEN X=150.50+.37*(T-663) @ RETURN
610 IF T>=556 THEN X=114.12+.34*(T-556) @ RETURN
620 IF T>=440 THEN X=79.32+.30*(T-440) @ RETURN
630 IF T>=292 THEN X=42.32+.25*(T-292) @ RETURN
640 IF T>=185 THEN X=21.99+.19*(T-185) @ RETURN
650 IF T>=84 THEN X=6.84+.15*(T-84) @ RETURN
660 IF T>=27 THEN X=.12*(T-27)
670 RETURN
800 'M': IF T>=897 THEN X=195.75+.37*(T-897) @ RETURN
810 IF T>=684 THEN X=125.46+.33*(T-684) @ RETURN
820 IF T>=578 THEN X=95.78+.28*(T-578) @ RETURN
830 IF T>=472 THEN X=69.28+.25*(T-472) @ RETURN
840 IF T>=384 THEN X=49.92+.22*(T-384) @ RETURN
850 IF T>=192 THEN X=17.28+.17*(T-192) @ RETURN
860 IF T>=48 THEN X=.12*(T-48)
870 RETURN
```

PAYROLL DEDUCTIONS PROGRAM FOR HP-71B

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

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

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

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

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

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

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

FOR PC-1500 & PC-2 USERS

PC-1500 POTPOURRI

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

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

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

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

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

If you want to prevent outright duplication of your program, then you had better not plan on publishing it on magnetic tape. Anyone with a duplicating tape recorder unit can duplicate what is on a magnetic tape. If someone is serious about distributing your software (say, in a foreign country), all they need is one copy of your program recorded on tape and they are in business. Presto! They can be making copies in a matter of seconds. If someone is going to break

the copyright laws in this manner (assuming your work has been copyrighted), it is certainly not hard to do from a technical viewpoint. Remember, we are talking here about tape-to-tape copying. The program never has to be loaded into a computer! No amount of protection within the program or computer is going to be able to prevent this simple outright duplication of the information contained on the tape cassette! By placing their own labels on the copied cassettes, program pirates can easily end up selling your program (particularly in a foreign country) with little chance of being detected.

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

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

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

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

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

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

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

Code Breaking

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

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

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

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

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

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

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

handful of prospective jump or call locations, start examining the code in the vicinity of those addresses. See if you find the same code value immediately preceding each prospective jump or call address. Any luck? If so, you may have found the opcode for a subroutine "return" instruction!

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

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

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

FOR PC-1350 USERS

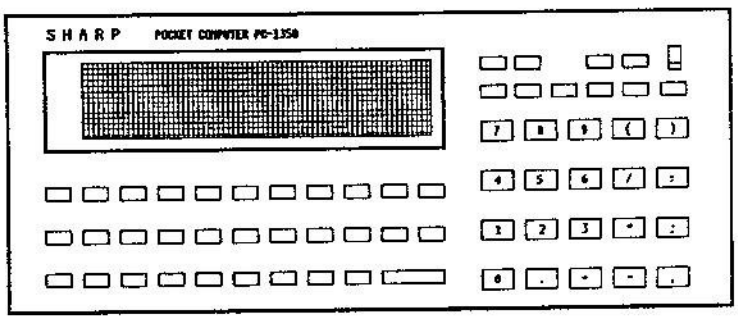
WELCOME ABOARD

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

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

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

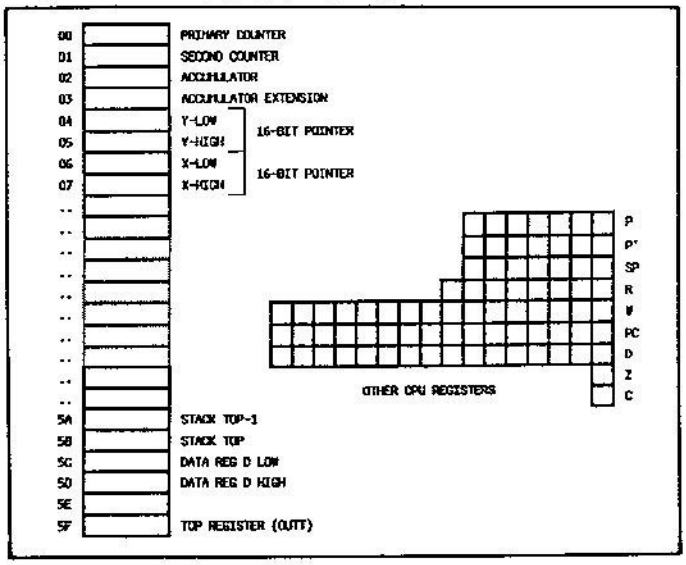
Now for the next hot piece of news. Preliminary investigation indicates that the PC-1350 utilizes the same CPU as that in the PC-1250 and PC-1260 series! And wouldn't you know it, a number of *PCW* fans have been hard at work busting the PC-1250 machine language code. In fact, next issue we will be publishing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



If you have been doing any exploring on your own, an accompanying diagram summarizes the internal registers associated with the unit's CPU. You can use the memory dump program described for the 1250 to begin mapping out the 1350. It appears that there is some RAM in the 1350 beginning at hexadecimal address 66000. This should be a fun machine to explore. Imagine what kind of machine language programming tools you could tuck into one of those 16K RAM cards?

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

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



FROM THE WATCH POCKET

The November, 1985, issue of McGraw-Hill's *BYTE* magazine has an interesting review (beginning on page 416) of a new pocket computer being manufactured in England. It is known as the Psion Organiser. The article was interesting for several reasons: the pure "hype" it contained, the author's apparent ignorance about the design and operation of other pocket computers that have been on the market for years, and the fact that the Psion Organiser uses a concept that seems readily applicable to other pocket computers, particularly models similar to the Sharp PC-1350.

The Psion Organiser has apparently been designed primarily as a data-processing device. Its claim to fame seems to be based on the concept of having its data base stored in a programmable read-only memory (PROM). Up to two (8K or 16K) PROM chips can be installed at a time. As the user creates a data base (such as a name and address file or telephone list), the PROM is blasted. This is a one-shot deal. Once a file is filled, you cannot re-program it unless you take the chip out and return it to a dealer. You can apparently, however, remove an entry from a file (probably by having the entry blasted to an all-zeroes condition) so that it is no longer processed by the program.

This is a nice concept that seems readily implementable on other PCs. Why couldn't Sharp make low-cost PROM cards for the PC-1350? PROMs are cheap. (Psion sells 16K versions for about $20.00.) The availability of such cards would allow users to create and maintain low cost data bases. Software vendors could also use such PROM cards to distribute application packages. Low cost ultra-violet erasers can be used to erase such PROMs so that they can be used again. This is a practical, feasible component.

But, whoever does it, let's hope they don't lay down the kind of hype that apparently surrounds the Psion machine. The BYTE article likens the tiny PROMs to serially accessed disk drives. That seems a pretty far-fetched comparison.

The BYTE article also contains other questionable implications. It alludes to the fact that the Psion machine contains a "proper" 8-bit microcomputer and states that pocket computers such as the Sharp PC-1500 are just "the modern version of the programmable calculator." It seems the author may be somewhat misinformed. The LH-5801 CPU inside the PC-1500 is certainly just as "proper" an 8-bit computer as the HD6301X touted for the Psion Organiser. The Psion unit is said to be "based on a very fast search algorithm with partial word matching" and to be capable of searching a 16K database in a mere five seconds. Wow! Incredible, what? Not really. Certainly machine language implementations on a PC-1500 or PC-1350 could perform just as well.

So what is the point of this discussion? Merely to point out that in England there is a company that thinks it can make a business out of putting a microcomputer into a pocket-sized package and selling its "database" capabilities. It need not have gone to all the trouble of designing such a hardware package. It could have simply added such capabilities to a PC-1500 or PC-1350. Seems the door for someone else doing this is still wide open! If Sharp can't see their way to doing it, perhaps some other enterprising firm will take a crack at it!

Speaking of Sharp. How about that new PC-1350? It seems a pretty nice machine, with some noteable improvements over the PC-1500. But, the left off a critical capability: there is no real-time clock! Shucks. There you are with a nice built-in serial communications port, a wonderful 4-line display, a powerful BASIC package, and no ability to perform real-time operations! No way to automatically turn the unit on and have it take measurements of its own accord. No means of recording the time of events or sounding an alarm to remind you of appointments. What a waste. Sharp blew a lot of the potential engineering and scientific market by leaving out this critical capability. They need to do some more market research about their potential users. Perhaps they should take a gander at Hewlett-Packard. Good ole HP wouldn't blow the opportunity to have their PCs hooked up as controllers and recorders by leaving out a real-time clock! The PC-1350 has plenty to offer, but it still ain't perfect.

If you are a businessperson who schedules a lot of appointments and does a lot of travel, you may be interested in a PC called the PAD. This is a special-purpose pocket computer. It will retain up to 80 appointments and memo notes in its 4K of memory. It *does* have a *real-time clock* so that it can alert you to an appointment by giving an alarm. There is also a section that can be used to store expenses in 8 categories. And, it can maintain an alphabeticized list of phone numbers. The PAD weighs just 5 ounces, measures 3-1/2 b6 5-1/4 by 3/8 inches, and runs for about 8 months on 3 silver oxide batteries. It is supplied with a vinyl case (that contains extra room for holding credit cards while traveling) and retails in the U.S. for $99.00. For additional information contact: **The Sharper Image**, *P.O. Box 26823, San Francisco, CA 94126-6823*, and ask for literature on their Electronic Memo Pad, product number ABX139.

1985 is going to be an exciting year for pocket computer afficionados. Look for some useful peripherals to come out in support of the PC-1350. A better mass storage device (than audio tape) is certainly realistic now. How about a battery powered floppy disk drive? That seems to be a possibility in the not-too-distant future. How about a miniature modem? A color printer? Say, how about some cables with connectors that mate to the PC-1350's serial port? That would help the "do-it-yourselfers" interface the new PC to "what-have-you" while the manufacturers decide what they are going to build.

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

__ Enroll me as a 1985 Subscriber (Issue numbers 37-44). $24.00 in U.S. (U.S. $30.00 to Canada/Mexico. Elsewhere U.S. $40.00 payable in U.S. funds against a U.S. bank.)
__ Enroll me as a 1984-85 Subscriber (Issue numbers 31-44). $42.00 in U.S. (U.S. $51.00 to Canada/Mexico. Elsewhere U.S. $70.00 payable in U.S. funds against a U.S. bank.)
__ Enroll me as a 1983-85 Subscriber (Issue numbers 21-44). $78.00 in U.S. (U.S. $93.00 to Canada/Mexico. Elsewhere U.S. $120.00 payable in U.S. funds against a U.S. bank.)
__ Enroll me as a 1982-85 Subscriber (Issue numbers 11-44). $102.00 in U.S. ($125.00 to Canada/Mexico. Elsewhere U.S. $160.00 payable in U.S. funds against a U.S. bank.)
__ Check here if paying by MasterCard or VISA. Please give credit card information below.

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

Name: _____

Addr: _____

City: _____ State: ____ Zip: _____

MC/VISA #: _____

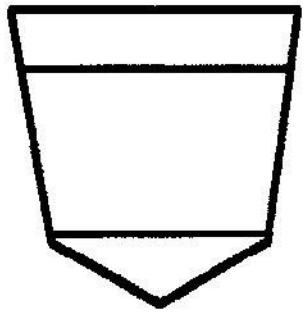
Signature: _____ Exp. Date: _____

mail this order form to:

POCKET COMPUTER NEWSLETTER

P.O. Box 252, Seymour, CT 06483

POCKET COMPUTER NEWSLETTER



Issue 38 - February

REVIEW OF TRAMSOFT TOOLKIT

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

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

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

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

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

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

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

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

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

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

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

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

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

LINK -- combines program modules.

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

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

PROGRAM -- activates a designated module.

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

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

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

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

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

FOR PC-1250/51/60/61 USERS

MACHINE CODE EXPOSED

Rick Wenger, 6221 - 18th Avenue, Kenosha, WI 53140, has provided the information contained in this article. Rick states that the work he has done would never have gotten off the ground without the pioneering efforts and encouragement of *Merlin Raber.*

He further advises that the information provided herein supersedes that published in Issue 26 of ***PCM.*** Note that the character code table and memory map for the PC-1250, contained in that earlier article, continue to be relevant with two corrections: the address range 0000 - 1FFF holds the so-called "hidden" ROM, and 4464 - 450A contains bit maps used for character generation.

When using the following information, you should bear in mind that the PC-1250 appears to have a rather frail operating system from the point of view of machine language experimenters! Machine code should not be placed in REM statements or often-used string variables. It seems that even trying to print some "undefined" character codes can cause the PC to crash. Rick says that attempts to create a protected area for machine language code have not proved practical as M.L. programs cannot be called from the area above E000. He recommends that code be poked above any BASIC program and below dimensioned variables. (Addresses C400 or C500 seem good starting points.)

If a machine language routine is called from BASIC, the following CPU register should either be left undisturbed or restored to their initial values: SP, C1, 16, 58, 59, 5A and 5B. Other CPU registers appear to be fair game. (CPU register nomenclature is described later in this article.)

The "hidden" ROM has a number of interesting utilities. (Some of which, do not appear to be used in the 1250). For instance, the BCD values used in the calculation of various transcendental functions are stored beginning in the vicinity of address 08E8. The startup routine is at 0000.

Description of CPU Registers

Refer to the accompanying diagram for a pictorial view of the CPU registers referred to in this discussion.

There are ninety-six 8-bit registers (presumably these are located on the CPU chip) which Rick has numbered from 00 to 5F (using hexadecimal notation). These 96 registers are *not* part of the addressable 64K of memory. He refers to these 96 registers in the following presentation as the Central Processing Array (CPA). Some of these 96 registers serve special purposes as indicated in the accompanying table.

Besides the CPA registers, there are some other important CPU registers. These are shown in another table.

Note that there appears to be just two flags accessible to the machine language programmer. The Z (zero) flag is set or reset when certain types of operation result in a condition of zero or non-zero. The C (carry) flag is set or reset when operations such as an addition or subtraction requires or does not require a carry or borrow.

Rick feels that the existence of iterative BCD add and subtract instructions means that detection of half-carry or overflow status was not needed by the ROM programmers.

A list of the CPU instructions that Rick has discovered, along with brief explanations of the various types, is shown in an accompanying table. Instructions that do not contain a description are self-explanatory or analogous to others that are described in the listing.

This list should get the typical M.L. enthusiast off to a good start. (Remember, the same CPU chip is used in the Sharp PC-1250, PC-1251, PC-1260, PC-1261 and PC-1350 units as well as the Radio Shack PC-3.) Perhaps in a future issue of ***PCM*** we can reorganize this material by machine code order to assist those working on disassembly projects.

We think readers will agree it is a monumental piece of work. Congratulations, Rick!

Table ***The CPA Registers in the PC-1250 CPU.***

CPA Register	Name	Description
00	C0	Primary counter. Used for iterative load instructions, etc.
01	C1	Another counter. Set to 01 throughout PC-1250 ROM to facilitate loading 2-byte words.
02	A	An 8-bit accumulator.
03	B	Combines with A to form a 16-bit register pair. B is also used to pass parameters to subroutines through the convenience of the EXA B instruction.
04	YL	Y-low, part of a 16-bit register pair capable of pointing to addressable memory.
05	YH	Y-high.
06	XL	X-low, part of a 16-bit register pair capable of pointing to addressable memory.
07	XH	X-high.
.	.	
.	.	
5A	(5A)	Stack storage. The stack works down from (5B). High byte is higher on the stack than low byte.
5B	(5B)	Start of subroutine stack.
5C	(5C)	Associated with low byte of data register D.
5D	(5D)	Associated with high byte of data register D.
.	.	
5F	T	Top CPA register. Associated with OUT T instruction. Contents of this register used to control beeper, display, and "BUSY" annunciator.

Diagram *PC-1250 CPU Registers.*

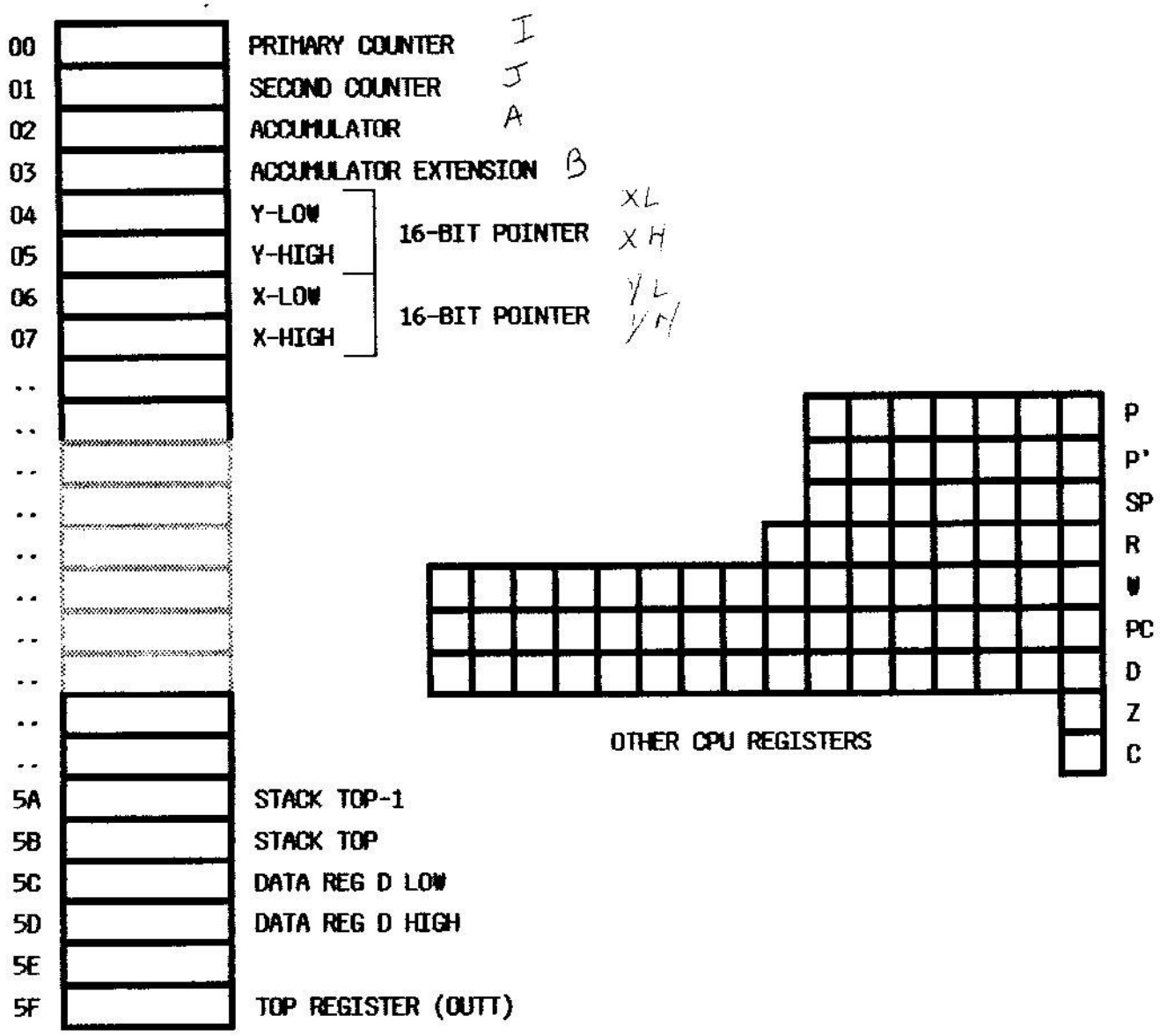


Table *Other PC-1250 CPU Registers.*

Register Size (Bits)	Name	Description
7	P	Primary pointer to the CPA.
7	P'	Secondary pointer to the CPA.
7	SP	Stack pointer to return address stack in CPA.
16	W	Points to addressable memory.
16	PC	Program counter. Indicates next instruction to be executed in memory.
16	D	Data register that obtains input from keyboard.
8	R	Working register. Retains last "operator" value.
1	Z	Zero flag.
1	C	Carry flag.

Table *PC-1250 Machine Language Instruction Set.*

LOAD INSTRUCTIONS

LDA #nn 02
Load accumulator (location 02 in the C.P.A.) with immediate data nn.

LDA P 20
Load accumulator with contents of P. P is not changed.

LDA P' 21

LDA SP 22

LDA (P) 59
Load accumulator with the contents of the C.P.A. register pointed to by P. P is not changed.

LDA (W) 57

LDA (PC) 56

LDB #nn 03

LDC0 #nn 00

LDC1 #nn 01

LDP #nn 12

LDP A 30

LDP' #nn 13

LDP' A 31

LDSP A 32

LDW #nnnn 10
Load 16-bit register W with next two immediate data bytes. High byte precedes low byte.

LDWL #nn 11
Low byte of W is loaded with immediate data nn. High byte of W is not changed.

LD (P)(W) 55
C.P.A. register pointed to by P is loaded with the contents of the memory location pointed to by W.

LD (P)(PC) 54

LD (W) A 52

LD (W)(P) 53

ADD INSTRUCTIONS

ADDA #nn 74
Immediate data nn is added to the accumulator. Carry and Zero flags are affected by the result.

ADD (P) A 44

ADD (P+) AB 14
A 16-bit addition operation. P>>P+1. Carry and zero flags affected by the full 16-bit result.

ADC (P) A C4
Add with initial value of carry included in the calculation.

ADD (P) #nn 70

ADEC0 (P) A 0C
The BCD number in the location pointed to by P' is added to the BCD number pointed to by P. P and P' are then decremented. The process is repeated, (utilizing the Carry flag) the number of times specified by the contents of C.P.A. register C0. (P' is decremented an extra time for some reason.)

ADEC0 (P)(P') 0E

ITERATIVE LOADS

LDDA (Y) 25
Decrement 16-bit register Y, then load the accumulator with the contents of the location pointed to by Y. New value of Y placed into W.

LDD (X) A 27

LDIA (Y) 24

LDI (X) A 26
Increment 16-bit register X. Then load the location pointed to by X with the contents of the accumulator. New value of X placed into W.

LD0 (P)(P') 08
Block C.P.A. transfer. Load the C.P.A. register pointed to by P with the contents of the C.P.A. register pointed to by P'. Repeat this process C0 times. Contents of C0 is not altered. The number of transfers done is equal to the contents of C0 plus 1.

LD0 (P)(W) 18
Block transfer from memory to C.P.A. registers. Identical to LD0 (P)(P') except W is not incremented after the last load (for some reason?).

LD0 (P) A 1E
The C.P.A. location pointed to by P is loaded with the contents of the accumulator. P is then incremented. The process is repeated C0 times.

LD0 (P) HR 35
Block transfer from hidden ROM to C.P.A.! Starting address of hidden ROM must be in 16-bit C.P.A. register AB (with A, the accumulator, containing the low byte of the address value). C.P.A. registers pointed to by P through (P+C0) are loaded. P goes to P+C0+1. (Hidden ROM in the PC-1250 has the address range 0 - 1FFF.

LD0 (W) A 1F

LD1 (P)(P') DA
Similar to LD0 (P)(P') except C1 is used as the counter.

LD1 (P)(W) 1A

EXCHANGE INSTRUCTIONS

EXA B DA

EXA (P) DB
Exchange contents of A and B. [(P)>>R, A>>(P), R>>A]

ITERATIVE EXCHANGES

EX0 (P)(P') 09
Similar to LD0(P)(P') except contents are exchanged each go-around.

EX0 (P)(W) 19

EX1 (P()P') 0B

EX1 (P)(W) 1B

SUBTRACT INSTRUCTIONS

SUBA #nn 75
Immediate data nn is subtracted from the accumulator. Carry and Zero flags are affected by the results.

SUB (P) A 45

SUB (P+) AB 15

SBC (P) A C5
Subtraction with initial value of the Carry flag included in the calculation.

SUB (P) #nn 71

SDEC0 (P) A 0D

SDEC0 (P)(P') 0F
BCD number at location pointed to by P' is subtracted from BCD value pointed to by P. P and P' decremented. Process repeated number of times indicated by contents of C0. The Carry and Zero flags are affected by the result of the total answer, not just the highest or lowest byte.

SHORT FORM LOADS

SLP 00 80
Short-form Load P with xx instruction. Single-byte instruction. No immediate data byte required.

SLP 01 81

. . . ..

. . . ..

. . . ..

SLP 3F BF

JUMP INSTRUCTIONS

JMP nnnn 79
Jump unconditionally to nnnn.

JCC nnnn 7D
Jump if Carry flag is cleared to address nnnn.

JCS nnnn 7F
Jump if Carry flag is set to address nnnn.

JZC nnnn 7C
Jump if Zero flag is cleared to address nnnn.

JZS nnnn 7E
Jump if Zero flag is set to address nnnn.

FORWARD BRANCHES

Note: When a forward or reverse branch is performed, the low byte of the new address for the PC is computed and stored in (SP-1).

FWD nn 2C
Jump forward unconditionally to current address + nn.

FCC nn 2A
Jump forward if Carry cleared to current address + nn.

FCS nn 3A
Jump forward if Carry Set to current address + nn.

FZC nn 28
Jump forward if Zero cleared to current address + nn.

FZS nn 38
Jump forward if Zero set to current address + nn.

REVERSE BRANCHES

REV nn 2D
Jump reverse unconditionally to current address - nn.

RCC nn 2B

RCS nn 3B

RZC nn 29

RZS nn 39

RDR nn 2F
Reverse branch with decrement and repeat. (SP) is decremented by one and a reverse branch is done if Carry is not set (cleared). If Carry is set, SP is incremented and execution continues sequentially. (One would normally push the number of repetitions desired onto the stack before using an RDR loop. Thus, it is unwise to call a subroutine from within an RDR loop.)

ROTATE INSTRUCTIONS

RDA 58
Rotate digits in accumulator. (Exchange high and low nibbles.)

RLA 5A
Rotate bits in accumulator to the left through the Carry flag.

RRA D2
Rotate bits in accumulator to the right through the Carry flag.

RLD (P) 1D
Rotates nibbles in a portion of the C.P.A. (registers P-C0 through P) to the left. Right-most nibble becomes zero. Left-most nibble is lost. P-C0-1>>P.

Rotates nibbles in a portion of the C.P.A. to the right. P+C0+1>>P.

RRD (P) 1C

INCREMENT INSTRUCTIONS

INCA 42
INCB C2
INCC0 40
INCC1 60
INCP 50
P is incremented by one. Carry and Zero flags are not affected.

INCX 06
The 16-bit X register pair is incremented. The Carry and Zero flags are not affected. New X>>W.

INCY 04
INC (08) 48
The C.P.A. location pointed to by the contents of C.P.A. register 08 is incremented by one. The Carry and Zero flags are affected.

INC (09) 68
INC (0A) 4A
INC (0B) 6A

DECREMENT INSTRUCTIONS

DEA 43
DEB 63
DEC0 41
DEC1 C1
DEP 51
DEX 07
DEY 05
DE (08) 49
DE (09) C9
DE (0A) 4B
The C.P.A. location pointed to by the contents of C.P.A. register 0A is decremented by one.

DE (0B) CB

BOOLEAN OPS & COMPARES

ANDA #nn 64
Contents of accumulator are ANDed with immediate data nn.

ORA #nn 65
BITA #nn 66
CPA #nn 67
AND (P) A 46
OR (P) A 47
BIT (P) A C6
CP (P) A C7
Flags C and Z Set/Reset based on (P)-A, although subtraction operation is virtual and does not change register contents.

AND (P) #nn 60
OR (P) #nn 61
Contents of C.P.A. location pointed to by P are ORed with immediate data nn.

BIT (P) #nn 62
CP (P) #nn 63
AND (W) #nn D4
OR (W) #nn D5
BIT (W) #nn D6
The Zero flag is set or reset based on the result obtained when the memory location pointed to by W is ANDed with immediate data byte nn. The Carry flag and other registers are not affected by the operation. (SP-1) used as scratchpad for Boolean ops involving W register.

ZBITA #nn 76
The Zero flag is set or reset based on results of A ANDed with nn, except that if Z is reset at the start, it remains reset. The Carry flag is not affected. R>>C0.

ZCPA #nn 77
ZBIT (P) #nn 72
ZCP (P) #nn 73
The Zero and Carry flags are set or reset based on results of (P) - nn, except that if Z is cleared at the start, it remains cleared. R>>C0.

SUBROUTINE INSTRUCTIONS

JSR nnnn 78
Jump to subroutine at address nnnn.

RTS 37
Return from subroutine.

SHORT FORM CALLS

SS 00 nn E0
Short-form jump to subroutine within hidden ROM. A two-byte call instruction.

SS 01 nn E1
. . . . ..
. . . . ..
. . . . ..
SS 1F nn FF

MISCELLANEOUS

SET C D0
Z flag is set

CLR C D1
Z flag is set

REFR #nn 4E
Suspends CPU operation for a length of time proportional to immediate data byte nn. Reroutes current to lines opened by OUTT.

IN nn 6B
Sets or resets Zero flag based on the status (active or non-active) of the RSV/PRO/RUN/OFF slide switch or BRK key. Data byte nn determines the signal that is to be tested.

OUT T DF
OUT T SUMMARY
Bit 0 - Display on if 1, Busy on if 0
Bit 3 - Power down if 1
Bit 4 - Tighten Beeper if 1
Slacken Beeper if 0
Bit 5 - Beeper on if 1
Bit 6 - Distorts Beeper if 1
Bits 1,2,7 - ????

INC P IF C0 6F
PUSH A 34
POP A 5B
PUSH XX #nnnn 7A
Pushes the address value nnnn onto the stack. First nn byte is higher in the stack. Do not know purpose of the XX byte in this 4-byte instruction.

CLR (SP) D8
CLA 23
INA D 4C
NOP 33

FOR HEWLETT-PACKARD HP-71B USERS

LEARN MORSE CODE

Here is an interesting program that you can use to learn (or refresh) Morse code skills. It is capable of generating these signals over a range of speeds suitable for training purposes. The audible tone it produces can be selected to suit an individual user.

The code is sent as cipher groups. There are five characters per group. This method is highly effective as a training technique because the student cannot guess or deduce what character will be sent next, as might be the case if ordinary text was utilized.

As the audio Morse code sound is produced by the program, the corresponding character is shown on the display. Thus, a novice can use the program to "learn" the code just be listening and watching the LCD screen. Once some proficiency has been developed, the student can start writing down the characters without looking at the display. The program stores the cipher code it sends in a buffer. At the end of a block of 200 characters, the user can opt to have the cipher groups shown on the display for confirmation purposes.

Once the program has been loaded into memory (the listing shown has a byte count of 1343), it is started with a simple RUN command. Tone and speed values should be given as an integer in the range 1 to 5. (Tone and speed increase as the associated value increases.) Once the speed value has been inputted, operation is automatic until a group of 200 characters has been sounded. To view the characters that have been sent at the conclusion of a block, respond with a Y to the query WANT LISTING?

Close examination of lines 200 - 280 in the listing should yield an understanding of how characters are generated by the program. You can add or delete characters by placing appropriate data in the A$ array. (Don't forget to change the DIM statement in line 10 and the RND statement in line 70, if additions or deletions are made.)

If you have a program for the HP-71B that you would like to share with *POW* readers, or tips for HP-71B users, drop a note to the editor. The -71B is a powerful hand-held that deserves to receive support and recognition!

Program *Morse Code on the HP-71B.*

```
10 DELAY 0 @ DESTROY ALL @ OPTION BASE 0 @ DIM A$(39)[8],B(5),C(5),C$(199)[1] @ GOSUB 200
20 INPUT "TONE? ";T @ T=ABS(INT(T)) @ IF T<1 OR T>5 THEN BEEP @ GOTO 20
30 T=T*200 @ T=T+400
40 INPUT "SPEED? ";S @ S=ABS(INT(S)) @ IF S<1 OR S>5 THEN BEEP @ GOTO 40
50 S=S*.02 @ S=.12-S @ P=S
60 DISP "" @ L=0 @ RANDOMIZE
70 R=INT(RND*40)
80 C$(L)=A$(R)[1,1] @ DISP TAB(11);C$(L)
90 D=VAL(A$(R)[2,2])
100 FOR J=3 TO D+2 @ C(J-3)=VAL(A$(R)[J,J]) @ NEXT J
110 FOR J=0 TO D-1
120 IF C(J)=1 THEN 140
130 BEEP T,S @ GOTO 150
140 BEEP T,S*3
150 NEXT J @ WAIT S @ L=L+1 @ IF L/5=INT(L/5) THEN WAIT 5*S
160 IF L#200 THEN 70
170 INPUT "WANT LISTING? ";W$ @ IF UPRC$(W$[1,1])="Y" THEN GOTO 300
180 GOTO 10
200 A$(0)="A201" @ A$(1)="B41000" @ A$(2)="C41010" @ A$(3)="D3100" @ A$(4)="E10"
210 A$(5)="F40010" @ A$(6)="G3110" @ A$(7)="H40000" @ A$(8)="I200" @ A$(9)="J40111"
220 A$(10)="K3101" @ A$(11)="L40100" @ A$(12)="M211" @ A$(13)="N210" @ A$(14)="03111"
230 A$(15)="P40110" @ A$(16)="Q41101" @ A$(17)="R3010" @ A$(18)="S3000" @ A$(19)="T11"
240 A$(20)="U3001" @ A$(21)="V40001" @ A$(22)="W3011" @ A$(23)="X41001"
250 A$(24)="Y41011" @ A$(25)="Z41100" @ A$(26)="1501111" @ A$(27)="2500111"
260 A$(28)="3500011" @ A$(29)="4500001" @ A$(30)="5500000" @ A$(31)="6510000"
270 A$(32)="7511000" @ A$(33)="8511100" @ A$(34)="9511110" @ A$(35)="0511111"
280 A$(36)=".6010101" @ A$(37)=",6110011" @ A$(38)="?6001100" @ A$(39)="/510010"
290 RETURN
300 DELAY 8 @ STD
310 FOR I=0 TO L-1
320 W$=STR$(I+1) @ IF I<9 THEN W$="00"&W$ @ GOTO 340
330 IF I<99 THEN W$="0"&W$
340 IF I=0 THEN 360
350 IF I/10#INT(I/10) THEN 370
360 DISP W$;":  ";
370 DISP C$(I); @ IF (I+1)/5=INT((I+1)/5) THEN DISP "  ";
380 IF (I+1)/10=INT((I+1/10) THEN DISP ""
390 NEXT I
400 GOTO 10
```

FOR PC-1500 & PC-2 USERS

PASSWORD PROTECTION

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

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

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

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

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

Program Registers

The program uses various memory locations throughout memory

Program *Password Protection*.

```
7C01 B5 40 AE 26   7C71 FD 88 B5 36   7CE1 50 58 78 5A   7D51 52 44 3A BE
7C05 4E B5 43 AE   7C75 AE 78 75 BE   7CE5 60 6A 0A F5   7D55 ED 3B B5 4E
7C09 76 4F EB 79   7C79 E2 43 83 5A   7CE9 88 03 F2 BA   7D59 AE 78 75 BE
7C0D F1 FF FD BE   7C7D FD C8 B5 2A   7CED 70 01 F2 FD   7D5D E2 43 C3 42
7C11 E3 79 D5 00   7C81 BE ED 4D FD   7CF1 58 B5 0B FD   7D61 FD C8 BE ED
7C15 FD 58 B5 0A   7C85 8A FD 0A 41   7CF5 CA FD 6A 4A   7D65 4D FD 8A FD
7C19 FD CA FD 6A   7C89 A5 79 D5 DD   7CF9 0C BA 7D 09   7D69 0A 41 4E 05
7C1D 4A 10 8E 10   7C8D B7 05 8B 07   7CFD 50 52 49 4E   7D6D 8B 04 FD 88
7C21 50 43 2D 31   7C91 AE 79 D5 FD   7D01 54 20 28 59   7D71 9E 17 F2 68
7C25 35 30 30 41   7C95 88 3E 20 B5   7D05 2F 4E 29 3F   7D75 7F 6A 00 4A
7C29 20 4F 4E 4C   7C99 D0 AE 7A 04   7D09 BE ED 3B BE   7D79 05 BE ED 3B
7C2D 43 4E 45 3A   7C9D B5 76 AE 7A   7D0D E2 43 83 10   7D7D B5 24 AE 78
7C31 3E ED 3B 48   7CA1 05 B5 50 AE   7D11 B7 59 8B 13   7D81 75 B5 3F BE
7C35 78 4A 60 58   7CA5 7A 06 B5 05   7D15 B5 FF AE 79   7D85 ED 57 BE E2
7C39 78 5A 50 6A   7CA9 AE 7A 07 B5   7D19 F1 E9 79 FF   7D89 43 C3 42 B7
7C3D 0A F5 88 03   7CAD D0 AE 7A 14   7D1D 00 FD 81 9A   7D8D 59 B3 07 B7
7C41 B5 00 AE 79   7CB1 B5 79 AE 7A   7D21 FD E3 F0 0B   7D91 4E 8B 11 BA
7C45 FF 48 6F 4A   7CB5 15 B5 FA AE   7D25 FD 9E 39 E9   7D95 7D 5C 48 7F
7C49 FF CA 65 CA   7CB9 7A 16 B5 05   7D29 79 F1 00 E9   7D99 4A 00 58 79
7C4D 67 CA 69 BE   7CBD AE 7A 17 B5   7D2D 79 FF 00 FD   7D9D 5A FA 6A 04
7C51 E3 3F F2 FD   7CC1 04 BE D0 F9   7D31 81 9A 48 7F   7DA1 F5 38 03 9A
7C55 58 B5 0B FD   7CC5 8B 17 48 78   7D35 4A 00 FD 83   7DA5 F2 43 7F 4A
7C59 CA FD 6A 4A   7CC9 4A 50 58 78   7D39 F2 FD 58 B5   7DA9 00 FD 88 BA
7C5D 09 BA 7C 6A   7CCD 5A 60 6A 0A   7D3D 0B FD CA FD   7DAD 7D 39 CC 65
7C61 50 41 53 53   7CD1 F5 88 03 BA   7D41 6A 4A 0D BA   7DB1 CA 69 49 00
7C65 57 4F 52 44   7CD5 7C EF FD E9   7D45 7D 54 53 45   7DB5 45 DD 99 04
7C69 3A BE ED 3B   7CD9 F0 0B FD 9E   7D49 54 20 50 41   7DB9 46 CA 67 9A
7C6D 48 76 4A 50   7CDD 66 48 78 4A   7D4D 53 53 57 4F
```

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

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

Several characteristics of the program are worth mentioning. Once access has been gained, the computer is set to DEG, RUN, softkey mode I, and LOCK. If this is not desirable, everything except the setting of LOCK may be changed by replacing the first ten bytes of the program with NOPs. Note too, that the password is not in effect unless the machine is turned off by using CALL G7C01. (The OFF key functions normally.) You might want to place the CALL G7C01 directive under control of a softkey. Also, if the machine sits idle for more than seven minutes during the password entry process, it will automatically shut off. However, upon power-up, nothing will have changed and the user can continue entering a password.

Relocating the Program

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

395F [illegible] to 39
39D5 to 39
39ED to 39
39FB to 3A
3A45 to 3A
3A95 to 3A
3AAD to 3A

The program uses G7F00-4 as a brief stack. If the user does not have a PC-1500A (or has another program stored there), this stack can be relocated to 7650-4 by making the following alterations:

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

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

Memory Map

The main sections of the program are located as follows:

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

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

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

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

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

FOR PC-1350 USERS

TITLE

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

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

POKE 66800,&12,&5F,2,&20,&0B,&DF,&37

Then, place the following BASIC statement in memory:

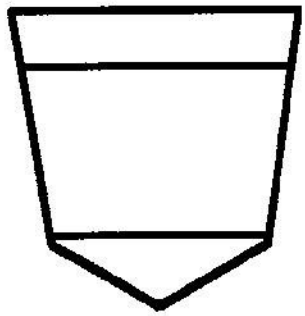
100 "B" CALL 66800 : WAIT 1 : PRINT "*" : GOTO "B"

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

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

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

POCKET COMPUTER NEWSLETTER



Issue 39 - March/April

HAND-HELD COMMUNICATES VIA RADIO WAVES

Motorola's Communications Sector has introduced an industry first: a hand-held computer capable of communicating in real-time using radio waves in the 800 megahertz frequency range!

Dubbed the KDT Computer Terminal, the environmentally rugged unit weighs under 28 ounces and measures 7-1/2 by 4 by slightly over 1-1/4 inches. The unit is battery operated, contains an 800 megahertz data radio, built-in antenna, and sophisticated software. It also sports a 2-line display (liquid-crystal) capable of displaying 27 characters per line. The keyboard contains 59 keys and has full alphabetic capability arranged in QUERTY format, programmable function keys, and a numeric calculator pad.

Two microprocessors split duties to provide multi-functional capability. That is, the unit can support communication functions while it is simultaneously running a user's application program. The unit can accomodate up to 160 kilobytes of ROM and 80 kilobytes of RAM internally, with access to external memory and peripheral devices provided via serial or parallel I/O interfaces. User applications can reside in 32 kilobyte ROM modules which may be plugged into the unit. Up to four such application programs may be installed at one time.

The radio system design that supports the KDT device can manage more than a thousand users on a single radio channel. This is accomplished through multiplexing of the system. When utilizing the built-in data radio, communications occurs at a rate of 4800 bits per second over standard 25 or 12.5 kilohertz channels. If radio coverage is not available, the built-in modem operates over voice-grade phone lines at a 300 bits per second rate.

To obtain more information about the KDT Portable Data Communications System, contact: *Motorola, Inc., 1301 East Algonquin Road, Schaumburg, IL 60196.* Address your inquiries to the Communications Sector.

Photo *Motorola KDT 800 Portable Terminal.*

Another *Personal Information* (tm) product.

FOR HEWLETT-PACKARD HP-71B USERS

APPOINTMENTS PROGRAM WITH ALARM

A feature lacking on the HP-71B (that was provided on its predecessor, the HP-75) is an appointment calendar equipped with an audio alarm. The 71B is capable of providing this service from a hardware point of view, since it has built-in timers and an audible annunciator. But, Hewlett-Packard elected not to include the software routines that would provide such capability in with the basic unit. (After all, if they included *all* the features of the 75 at half the price, it would be a lot harder to appease the early 75 customers!)

The package of routines provided here yield much of the capability for scheduling appointments that is found in the HP-75. You can enter appointment dates and times (through the year 1999) and a reminder message. Appointments may be scheduled as repeating (at a cycle of anywhere between 1 minute to 24 hours) or as single events. You are reminded of appointments through an audible warning. Capability is provided to have an alarm issued even if the HP-71B has been turned off or is executing another program. (This capability is optional and is selected by the user as desired.) If this special capability is not elected, then a past-due appointment is normally announced whenever the HP-71B is turned on. Appointments are acknowledged and/or erased via keyboard control. With slight modification, you can utilize the basic package to have the HP-71B perform other programs when an alarm occurs.

Space considerations do not permit a detailed discussion of the routines. Thus, this article will deal primarily with operational instructions.

The routines are distributed over three files. The major file is named ALARMS. Two associated files are named PWRDNALM (Power Down Alarm) and CHECKALM (Check Alarm). The former is used if you want to have the HP-71B "wake-up" and sound an alarm when it is in the "off" mode. The latter is used whenever it is desired to see whether an appointment time has been reached or is past-due. (This routine can be periodically called by any other program to provide real-time alarm interrupt capability.) Finally, a fourth file named APPOINTS is used as a data file in which appointment information is maintained. (This file is created automatically by the ALARMS routine.) The APPOINTS file contains room for a maximum of 20 appointments. All together the various routines and the data file consume slightly more than 5K of memory.

As a check on your keyboarding accuracy, the amount of space consumed by each program file, as shown by the CAT command, should be:

ALARMS = 3201 bytes
PWRDNALM = 583 bytes
CHECKALM = 430 bytes

Program two "user" keys (using the DEF KEY command) such as the letters "a" and "p" to execute (RUN) the ALARMS and PWRDNALM programs when invoked. The CHECKALM routine should be invoked whenever the HP-71B is turned on. (Use the STARTUP command to "RUN CHECKALM".) It may also be called by any other program(s) if you wish to have alarms announced when you are running other packages. [Note: use the file names exactly as shown because the programs call one-another by those exact titles.]

To begin using the appointment scheduler, execute the ALARMS program. Respond to the initial ENTER/CANCEL query with the appropriate character depending on whether you wish to enter a new appointment or cancel an already existing entry. (Note that the package automatically provides the most common expected response to any prompt. When your response matches this, all you need do is press the END LINE key.)

Respond appropriately to further prompts. When entering an alarm date/time, just edit the current date/time that is provided with the prompt. Note that time is inputted using a 24-hour format. (Also note that you cannot enter an appointment whose date/time is prior to the current date/time.) Additional prompts allow you to direct that an alarm update and repeat at the interval you specify (minimum of 1 minute, maximum of 24 hours) and to enter a message indicating the nature of the appointment. (This reminder message may be up to 22 characters in length.)

Each time an appointment is entered, it is ordered in the APPOINTS file by date/time of occurrence. Thus, at the end of each entry procedure, you will observe a brief "sorting" message on the display. At the conclusion of each entry, the ALARMS program exits by running CHECKALM to see whether any alarms are due. The program then ends with the HP-71B in the command mode. To enter another alarm, you must RUN the ALARMS program again. Hence, the earlier suggestion that you program a user key with this directive.

If you want to be reminded of a pending appointment when the HP-71B is turned off, execute the PWRDNALM module. The HP-71B will go to sleep, but continue checking for the next appointment. Should one occur, the unit "awakens" and sounds the alarm in the "loud" mode. You "acknowledge" an alarm (stop the audible indicator) by pressing any key. Once an alarm has been "acknowledged" you may use the cursor keys to view any appointment reminder message. Pressing the END LINE key after an alarm has been acknowledged results in the following.

If the alarm is a repeating one, its new time will automatically be calculated.

Then, the user is asked whether or not to cancel the alarm. If it is not cancelled, the alarm is maintained in the APPOINTS file. (Note that if it is a non-repeating alarm, it will remain as a "past-due" alarm.)

The user then has the option of viewing any other items in the APPOINTS file.

If the PWRDNALM program sounds an alarm that is not acknowledged within approximately 15 seconds, then the unit simply goes back to "sleep" and ignores the "past-due" alarm condition. Thus, the audio annunciator feature is a one-shot deal when using the PWRDNALM routine. This procedure saves a lot of wear and tear on the batteries in the event that you leave town for a few days and forget about an upcoming appointment stored in the 71B!

Regardless of whether you elect to use the PWRDNALM routine, the CHECKALM routine will be executed any time you power up your HP-71B. (Provided that you program the STARTUP statement to 'RUN CHECKALM'.) If this routine finds a past-due appointment condition, it will sound the audible alarm (in the quieter mode, since someone is obviously near the unit). The alarm can then be acknowledged and the appointment removed from the file, as desired.

Take note of the fact that the system is designed to alert you to the "oldest" past-due appointment in the file named APPOINTS. You must remove it (by cancelling the entry) before a later appointment will be serviced by the audio alarm system. (In summary, the APPOINTS file is kept in chronological order. The CHECKALM routine tests the first entry in the file.)

With a little study of the routines, which are modular by design, you can undoubtably find ways to customize or adapt them to suit your particular needs. For instance, you might elect to have one of your own subprograms executed whenever an alarm went off.

Program *ALARMS*.

```
10 DELAY 1,.5 @ WIDTH 96 @ DESTROY ALL
20 INPUT "ENTER/CANCEL (E/C)? ","E";W$
30 IF UPRC$(W$[1])$"E" AND UPRC$(W$[1])$"C" THEN BEEP @ GOTO 20
40 IF UPRC$(W$[1])="C" THEN 200
50 REM  (C) COPYRIGHT 1985
60 REM  SCELBI C.C., INC.
70 REM  ALL RIGHTS RESERVED
100 GOSUB 1000
110 GOSUB 1100
120 GOSUB 1200
130 GOSUB 1300
140 GOSUB 1400
150 CALL SORT
160 GOTO 210
200 CALL CANCEL
210 DISP "ALARM(S) PROCESSED."
220 RUN "CHECKALM"
1000 W$=DATE$ @ W$=W$[4,8]&"/"&W$[1,2]
1005 ON ERROR GOTO 1095 @ W$=W$[1,8] @ INPUT "ALARM DATE? ",W$;W$
1010 Y=VAL(W$[7,8]) @ IF Y<84 OR Y>99 THEN 1095
1015 M=VAL(W$[1,2]) @ IF M<1 OR M>12 THEN 1095
1020 IF M#2 THEN 1035
1025 IF Y/4=INT(Y/4) THEN D1=29 ELSE D1=28
1030 GOTO 1040
1035 IF M=4 OR M=6 OR M=9 OR M=11 THEN D1=30 ELSE D1=31
1040 D=VAL(W$[4,5]) @ IF D<1 OR D>D1 THEN 1095
1045 W$=W$[7,8]&"/"&W$[1,5] @ IF W$<DATE$ THEN 1095
1050 D$=W$ @ OFF ERROR @ RETURN
1095 BEEP @ DISP "INVALID DATE." @ GOTO 1000
1100 W$=TIME$[1,5] @ ON ERROR GOTO 1195
1105 INPUT "ALARM TIME? ",W$;W$ @ W$=W$[1,5]
1110 Y=VAL(W$[1,2]) @ IF Y<0 OR Y>23 THEN 1195
1115 M=VAL(W$[4,5]) @ IF M<0 OR M>59 THEN 1195
1120 IF D$>DATE$ THEN 1130
1125 IF TIME$[1,5]>W$ THEN 1195
1130 T$=W$ @ OFF ERROR @ RETURN
1195 BEEP @ DISP "INVALID TIME." @ GOTO 1100
1200 W$="" @ ON ERROR GOTO 1295
1205 INPUT "REPEAT ALARM (Y/N)? ","N";W$
1210 IF UPRC$(W$[1])$"Y" THEN 1260
1215 INPUT "INTERVAL? ","24:00";W$ @ W$=W$[1,5]
1220 D=0
1225 D=D+1 @ IF D=3 THEN 1225
1230 IF NUM(W$[D])<48 OR NUM(W$[D])>57 THEN 1295
1235 IF D<5 THEN 1225
1240 Y=VAL(W$[1,2]) @ IF Y<0 OR Y>24 THEN 1295
1245 M=VAL(W$[4,5]) @ IF M<0 OR M>59 THEN 1295
1250 IF Y=0 AND M=0 THEN 1295
1255 IF Y=24 AND M#0 THEN 1295
1260 I$=W$ @ IF I$="N" THEN I$=" "
1265 OFF ERROR @ RETURN
1295 BEEP @ DISP "INVALID INTERVAL." @ GOTO 1200
1300 ON ERROR GOTO 1395 @ W$=""
1305 INPUT "ALARM MESSAGE? ",W$;W$
1310 IF LEN(W$)>22 THEN 1395
1315 M$=W$ @ OFF ERROR @ RETURN
1395 BEEP @ DISP "MESSAGE TOO LONG." @ GOTO 1300
1400 ON ERROR GOTO 1485 @ Y=0
1405 Y=Y+1
1410 W$=CAT$(Y)[1,8]
1415 IF W$="APPOINTS" THEN 1445
1420 IF W$#"" THEN 1405
1425 CREATE DATA APPOINTS,20,52
1430 ASSIGN #1 TO APPOINTS
1435 FOR Y=0 TO 19 @ PRINT #1,Y;"" @ NEXT Y
1440 ASSIGN #1 TO *
1445 ASSIGN #1 TO APPOINTS
1450 Y=-1
1455 Y=Y+1 @ IF Y>19 THEN ASSIGN #1 TO * @ GOTO 1485
1460 READ #1,Y;W$
1465 IF W$#"" THEN 1455
1470 PRINT #1,Y;D$,T$,I$,M$
1475 ASSIGN #1 TO *
1480 GOTO 1495
1485 DISP "APPOINTS FILE ERROR."
1490 DISP "APPOINTMENT CANCELLED."
1495 OFF ERROR @ RETURN
1500 SUB SORT @ GOSUB 1580 @ IF M<2 THEN 1595
1505 ASSIGN #1 TO APPOINTS @ DISP "SORTING APPOINTMENTS."
1510 FOR Y=1 TO M-1
1515 FOR D=Y+1 TO M
1520 READ #1,Y-1;D$,T$,I$,M$
1525 READ #1,D-1;D1$,T1$,I1$,M1$
1530 W$=D$&T$ @ W1$=D1$&T1$
1535 IF W$<=W1$ THEN 1550
1540 PRINT #1,Y-1;D1$,T1$,I1$,M1$
1545 PRINT #1,D-1;D$,T$,I$,M$
1550 NEXT D @ NEXT Y
1555 ASSIGN #1 TO *
1560 GOTO 1595
1580 ASSIGN #1 TO APPOINTS @ M=-1
1585 M=M+1 @ READ #1,M;W$ @ IF W$#"" THEN 1585
1590 ASSIGN #1 TO * @ RETURN
1595 END SUB
1600 SUB CANCEL @ ASSIGN #1 TO APPOINTS @ M=0 @ DIM W$[50]
1605 READ #1,M;D$ @ IF D$="" THEN 1795
1610 READ #1,M;D$,T$,I$,M$
1615 W$=D$[4,8]&"/"&D$[1,2]&" "&T$&" "&I$&" "&M$
1620 DELAY 8,8 @ DISP W$ @ W$=KEY$
1625 INPUT "CANCEL (Y/N)? ","N";W$
1630 IF UPRC$(W$[1])="Y" THEN GOSUB 1700 @ M=M-1
1635 M=M+1 @ IF M>19 THEN 1795
1640 INPUT "LIST NEXT (Y/N)? ","Y";W$
1645 IF UPRC$(W$[1])="Y" THEN 1605 ELSE 1795
1700 D=M+1
1705 READ #1,D;W$ @ IF W$="" THEN 1720
1710 READ #1,D;D$,T$,I$,M$
1715 PRINT #1,D-1;D$,T$,I$,M$ @ D=D+1 @ IF D<20 THEN 1705
1720 PRINT #1,D-1;"" @ RETURN
1795 ASSIGN #1 TO * @ DELAY 1,.5 @ END SUB
1800 SUB UPDATE(D$,I$,M$,T$) @ Y1=VAL(T$[1,2])+VAL(I$[1,2])
1805 M1=VAL(T$[4,5])+VAL(I$[4,5])
1810 IF M1>=60 THEN M1=M1-60 @ Y1=Y1+1
1815 IF Y1<24 THEN 1895
1820 Y1=Y1-24 @ D=D+1
1825 D=VAL(D$[7,8]) @ Y=VAL(D$[1,2]) @ M=VAL(D$[4,5])
1830 IF D=31 THEN M=M+1 @ D=1 @ GOTO 1855
1835 IF D=30 AND (M=4 OR M=6 OR M=9 OR M=11) THEN M=M+1 @ D=1 @ GOTO 1865
1840 IF D=29 AND M=2 AND Y/4=INT(Y/4) THEN M=M+1 @ D=1 @ GOTO 1865
1845 IF D=28 AND M=2 AND Y/4#INT(Y/4) THEN M=M+1 @ D=1 @ GOTO 1865
1850 D=D+1 @ GOTO 1865
1855 IF M>12 THEN M=1 @ Y=Y+1
1860 IF Y>99 THEN DISP "CALENDAR OVER-RANGE." @ STOP
1865 W$=STR$(Y) @ IF LEN(W$)<2 THEN W$="0"&W$
1870 D$=W$
1875 W$=STR$(M) @ IF LEN(W$)<2 THEN W$="0"&W$
1880 D$=D$&"/"&W$
1885 W$=STR$(D) @ IF LEN(W$)<2 THEN W$="0"&W$
1890 D$=D$&"/"&W$
1895 W$=STR$(Y1) @ IF LEN(W$)<2 THEN W$="0"&W$
1900 T$=W$
1905 W$=STR$(M1) @ IF LEN(W$)<2 THEN W$="0"&W$
1910 T$=T$&":"&W$
1915 ASSIGN #1 TO APPOINTS @ PRINT #1,0;D$,T$,I$,M$ @ ASSIGN #1 TO * @ END SUB
```

Program *CHECKALM*.

```
100 DESTROY ALL @ GOSUB 900 @ DIM W$[52] @ DELAY .05,0 @ IF D$="" THEN 155
105 IF D$>DATE$ THEN 195
110 IF D$<DATE$ THEN 120
115 IF T$>TIME$ THEN 195
120 OFF TIMER #1 @ W$=D$[4,8]&"/"&D$[1,2]&" "&T$&" "&I$&" "&M$ @ DISP W$
125 FOR Y=1 TO 60
130 BEEP 1000,.2 @ WAIT .05
135 W$=KEY$ @ IF W$#"" THEN Y=64
140 NEXT Y
145 IF Y<62 THEN 195
150 IF I$=" " THEN 190
155 CALL UPDATE(D$,I$,M$,T$) IN ALARMS
190 DELAY 8 @ CALL CANCEL IN ALARMS @ CALL SORT IN ALARMS
195 CFLAG ALL @ DELAY 1,.5 @ DISP CHR$(27)&"J"&CHR$(27)&"E";")" @ END ALL
900 ASSIGN #1 TO APPOINTS
905 READ #1,0;D$ @ IF D$="" THEN RETURN
910 READ #1,0;D$,T$,I$,M$
915 ASSIGN #1 TO * @ RETURN
```

Program *PURDMALM.*

```
10 ON TIMER #1,59 GOTO 50 @ D1=0
20 T1=TIME @ BYE
30 T2=TIME @ IF T2<59 THEN T2=86400-T1+T2 ELSE T2=T2-T1
40 IF T2<58 THEN CFLAG -25 @ RUN CHECKALM
50 IF D1#0 THEN 20
60 SFLAG -25
70 GOSUB 100
80 IF Y<62 THEN 20 ELSE 10
90 END
100 GOSUB 900 @ DIM W$[52] @ DELAY .05,8 @ IF D$="" THEN 230
110 IF D$>DATE$ THEN 230
120 IF D$<DATE$ THEN 140
130 IF T$>TIME$ THEN 230
140 D1=1 @ W$=D$[4,8]&"/"&D$[1,2]&" "&T$&" "&I$&" "&M$ @ DISP W$
150 FOR Y=1 TO 60
160 BEEP 1000,.2 @ WAIT .05
170 W$=KEY$ @ IF W$#"" THEN Y=64
180 NEXT Y
190 IF Y<62 THEN 230
200 IF I$=" " THEN 220
210 CALL UPDATE(D$,I$,M$,T$) IN ALARMS
220 DELAY 8 @ CALL CANCEL IN ALARMS @ CALL SORT IN ALARMS
230 CFLAG -25 @ DELAY 1,.5 @ RETURN
900 ASSIGN #1 TO APPOINTS
905 READ #1,0;D$ @ IF D$="" THEN RETURN
910 READ #1,0;D$,T$,I$,M$
915 ASSIGN #1 TO * @ RETURN
```

FOR PC-1250/51/60/61 USERS

MEMORY MAP FOR THE PC-1261

Merlin Rober, 407 North 1st Avenue, Marshalltown, IA 50158, is back with more vital information. This time he provides a memory map for the relatively new Sharp PC-1261. You may have to squint a bit to see everything as we reduced the material in order to squeeze as much information as possible into this edition. PC-1261 owners will definitely find any squinting worth the effort!

Table *PC-1261 Memory Map.*

The PC-1261 uses the same CPU as the earlier PC-1250. However, the 1261 runs a good deal faster than the 1250, partly because of the fact that the 1261 uses a "link byte" following each line number (as the PC-1500 does). Also, the FOR/NEXT loops have been speeded up significantly; execution of NEXT takes only about .0075 seconds, twice as fast as the PC-1500! However, most other BASIC operations run only about half as fast as they do in the PC-1500.

Another rather surprising feature of the 1261 is the speed at which it CLOADs and CSAVEs. Not counting the time for the "header", the 1261 loads a kilobyte in just 27 seconds. (Compare with the PC-1500's 73 seconds per kilobyte!)

The 1261, unlike the 1250, uses standard ASCII character codes. Its tokens are also different. Nevertheless, the 1261 can CLOAD programs that were written on (and CSAVEd by) the 1250! While loading such a program, the 1261 uses a table in ROM to translate each code as it is received. It also calculates and inserts a link byte in each program line. It must be kept rather busy at such times!

Athough not supported in the instruction manual, the PC-1261's BASIC includes PEEK, POKE, CALL, CSAVEM, and CLOADM.

For those interested in the internal organization of the PC-1261, a memory map follows. Although it contains a number of "gaps", it includes all major locations. Note that pointers are stored low byte first.

0000-01FF	ROM (Inaccessible by PEEK)
2000-203B	Display Buffer, left half of top line
203C	Mode: 01=RESERVE, 02=PRO, 08=RUN
203D	01=BUSY, 02=PRINT, 08=Katakana, 10=SML, 20=SHIFT, 40=DEF
203E-203F	Not used
2040-207B	Display Buffer, left half of bottom line
207C	01=DEG, 02=RAD, 04=GRAD, 20=ERROR, 40=Unused mark
207D-207F	Not used
2080-20BF	Pending operand stack, 8 floating-point registers
20C0-27FF	No memory
2800-283B	Display Buffer, right half of top line
283C-283D	Not used
283E-283F	Pointer to start of RESERVE Memory
2840-287B	Display Buffer, right half of bottom line
287C-287F	Not used
2880-288F	Pending-operation stack, 16 codes
2890-28B7	Arithmetic workspace
28B8-28BF	Current value of random number
28C0-3FFF	No memory
4000-407F	EASY SIMULATION Area (unexpanded)
4080-64FF	BASIC program area. FF byte in 4080; program begins 4081. Defined variables and arrays begin 64FF, working backward.
6500-6507	Z (Z$)
6508-650F	Y (Y$)
6510-6517	X (X$)
6518-651F	W (W$)
6520-6527	V (V$)
6528-652F	U (U$)
6530-6537	T (I$)
6538-653F	S (S$)
6540-6547	R (R$)
6548-654F	Q (Q$)
6550-6557	P (P$)
6558-655F	O (O$)
6560-6567	N (N$)
6568-656F	M (M$)
6570-6577	L (L$)
6578-657F	K (K$)
6580-6587	J (J$)
6588-658F	I (I$)
6590-6597	H (H$)
6598-659F	G (G$)
65A0-65A7	F (F$)
65A8-65AF	E (E$)
65B0-65B7	D (D$)
65B8-65BF	C (C$)
65C0-65C7	B (B$)
65C8-65CF	A (A$)
65D0-65FF	RESERVE Memory (48 bytes)
6600-6617	?
6618	Current horizontal position of display cursor
6619-661A	Pointer to start of editable (MERGEd) program
661B-661C	Pointer to start of program being executed
661D-6620	?
6621-6622	Pointer to start of EASY SIMULATION area
6623-6624	Pointer to end of EASY SIMULATION area
6625-6626	Pointer to last entry into EASY SIMULATION program
6627-6628	BCD idle-time counter, non-resetting
6629-662A	?
662B	Bit 4=display flash flag
662C	Code of last key used
662D	User-defined-key label sought
662E-6634	?
6635-6636	Pointer to INPUT or assignment statement
6637-6638	?
6639-663A	Pointer, BASIC locations
663B-663F	?
6640-6647	Stored Password
6648-664F	Floating-point accumulator, numeric data or string pointer
6650-667F	Buffer, for output to display
6680-6681	Auto power-off countdown, low byte first
6682	?
6683-6696	GOSUB stack (ten return addresses)
6697-66AF	Numeric results, formatted for display or printing
66B0-66B7	?
66B8-66CF	Buffer, for output to printer
66D0-66D6	?
66D7	Flags: 08=PRINT=LPRINT
66D8	Flags: 04=WAIT
66D9	Flags: 20=password-protected
66DA-66E0	?
66E1-66E2	Pointer to start of BASIC program
66E3-66E4	Pointer to end of BASIC program
66E5-66E6	WAIT time setting, low byte first
66E7	?
66E8	Code of character being flashed
66E9	Flashing cursor location
66EA	Pointer to Input Buffer, low byte
66EB-66ED	Pointer to variable used in assignment statement

66EE-66EF	Pointer to variable used in INPUT statement
66F0-66F2	?
66F3	Pointer to FOR/NEXT stack, low byte
66F4	Pointer to GOSUB stack, low byte
66F5	Pointer to pending operand stack, low byte
66F6	Pointer to pending operation stack, low byte
66F7	Pointer to String Buffer, low byte
66F8	USING, 88=scientific notation
66F9	USING, positions preceding decimal point
66FA	USING, POSITIONS following and including decimal point
66FB	USING, positions in character strings
66FC-66FD	Pointer to start of defined variables and arrays
66FE-66FF	Pointer to DATA item
6700-6701	Pointer to end of BASIC statement just executed
6702-6703	Pointer to BASIC at halt in execution
6704-6705	?
6706-675F	FOR/NEXT stack (5 loops, each consisting of 18 bytes)
6760-67AF	String Buffer, 80 bytes
67B0-67FF	Input Buffer, 80 bytes
7000-7FFF	Not used
8000-8044	ROM Begins: Jump instructions
8045-820B	Table of codes for generating display characters
820C-8391	Table, codes for Katakana, etc.
8392-839F	Codes for "BREAK IN" and "ERROR"
83A0-83D5	Key codes
83D6-840B	Key codes, Katakana mode
840C-8441	Key codes, SHIFTed
8442-8477	Key codes, Katakana mode, SHIFTed
847B-84AD	Key codes, SML mode
84AE-84E3	Key codes, SML mode, SHIFTed
84E4-84FD	Table of codes for keys A to Z, used with DEF key
84FE-A176	ROM routines
A177-A1AA	Table, address of token word, for given initial letter
A1AB-A254	Table, address of token word, for given token
A255-A4CE	Table of token words, tokens, and routine addresses
A4CF-A58D	Code conversion table for PC-1250 cassettes
A58E-D626	ML code, BASIC interpreter
D627-D63B	EASY SIMULATION messages "CLEAR", "YES-1 OR NO-2"
D63C-E7BD	ML Code
E7BE-E88F	Error messages
E890-E955	ASCII Codes, for display with "HELP"
E956-E98A	Table of addresses (by first letter) in "HELP" section
E98B-F456	"HELP" displays, involving BASIC keywords
F457-FFFF	Unused ROM, except possibly for a few short subroutines

MACHINE LANGUAGE UPDATE

Rick Wenger, 6221-18th Avenue, Kenosha, WI 53140, sends the following update on the information that appeared in Issue 38 of ***PCW***:

An exhaustive analysis of PC-1250 ROM code has turned up some interesting new data on the machine language instruction set.

The following instructions do not occur anywhere in ROM and should be considered unsupported:

1. LDA (PC) and LD(P) PC (opcodes 56 and 54).
2. The ZBII and ZCP instruction (codes 76, 77, 72 & 73).
3. CLR (SP) (opcode D8).
4. CLA (opcode 23) and NOP (opcode 33).

The problem with using any of these instructions in machine language routines, is that if Sharp ever decides to upgrade this CPU, these instructions might go by the wayside. Serious software developers will want to stay clear of them. (They are primarily of interest only to hardcore CPU sleuths!)

The following opcodes do not occur in the PC-1250 ROM, but may be considered "valid" opcodes: 21, 4A, 4B, 9D, 9F, A3, AB, B2, B3, B6, BB, BC, BD, BE, C0, C5, C6, CA, CB, F4 and FD. All of these (except C0) were listed in Issue 38 of ***PCW***. All of these fit a pattern within the instruction set. It is understandable in each case, why the ROM programmer's never got around to utilizing them in the PC-1250. (Thus, there are two separate reasons for assuming that the opcodes represent "valid" operations.)

Twenty-three other opcodes were also not used in the ROM and were not listed as instructions in the previous article. Most of these opcodes will crash if an attempt is made to execute them in a program.

The following information serves as a correction or update to the material that appeared on pages 4 and 5 in Issue 38 of ***PCW***:

The explanation for ADECO (P) (P') occurs under the ADECO (P) A instruction.

The instruction ADECO (P) A (opcode 0C) adds the contents of the accumulator to the BCD number pointed to by P. P is decremented and then zero is added with carry. This adding of zero with carry is done the number of times indicated by the contents of C0. (P-C0-1 goes into P.)

The opcode for LD1 (P) (P') is 0A, not the DA shown.

The explanation under EXA (P) should read: Exchange contents of A and (P). (...etc...)

The mnemonic for the opcde 0B should read: EX1 (P)(P').

The LOAD P instructions (such as SLP 00) load register P with immediate data, not with an instruction.

The forward and reverse relative branches are computed by counting from the final byte of the instruction (the nn address), not from the initial byte of the next instruction.

The explanation for the RRO (P) instruction appears on the two lines right above it.

The opcode for INCC1 should be C0, not 60.

The opcode for INC (09) should be C8, not 68.

The opcode for INC (0B) should be CA, not 6A.

The opcode for DEB should be C3, not 63.

To clarify the effects of the increment and decrement instructions on the C and Z flags: C and Z are only affected by increments/decrements on 8-bit registers. Seven- and sixteen-bit increments/decrements do not affect the flags.

The INC IF C0 (opcode 6F) instruction is incorrect and should be deleted from the list.

There is more to come! The analysis made to date indicates there are 9 more valid opcodes available for use by this CPU. Look for a discussion of them in a future issue of ***PCW***.

FOR PC-1350 USERS

MEMORY MAP FOR PC-1350

See the article under the PC-1250/51 column that provides updates to the instruction set (as the same CPU is also used in the Sharp PC-1350).

In addition to his PC-1261 sleuthing work, *Norlin Rober, 407 N. 1st Avenue, Marshalltown, IA 50158,* has also been doing a lot of exploring on the Sharp PC-1350. He notes that, except for its four-line graphic display, the PC-1350 does not differ radically from the PC-1260/61. It uses the same CPU, and its BASIC uses the same tokens (with a few additions). The 1350 is also the first Sharp PC to provide the entire set of standard ASCII characters (although the 1261 came close).

Norlin also reminds readers that the 1350 is capable of loading programs that have been recorded on cassettes by the earlier Sharp PC-1211 (Radio Shack PC-1), Sharp 1250/51 (Radio Shack PC-3 & PC-3A), and Sharp PC-1260/61. Some modifications may be required to the programs themselves (after loading), especially if the programmer used non-approved programming tricks such as utilizing implied multiplication on the 1211 or 1250.

He has been digging out other information that will be passed along in future issues of ***PCW***. Meanwhile, enjoy the fruits of his recent labors by perusing the accompanying memory map for the PC-1350!

Table *PC-1350 Memory Map.*

```
0000-0823  ROM, ML Code.  RESET routine starts 0000
0824-08B3  Constants for computation of transcendental functions
08B4-1F04  ML Code
1F05-1FFF  Unused ROM
2000-5FFF  RAM card expansion area.
           8K Card, 4000-5FFF; 16K Card, 2000-5FFF
6000-602F  System Management Area (with no RAM card in use);
           6006-602F is a copy of 6F00-6F29
6030       BASIC Start Marker (FF byte)
6031-6C2F  BASIC area.  Program starts 6031; defined variables
           and arrays end 6C2F.
6C30-6C37  Z (Z$)
6C38-6C3F  Y (Y$)
6C40-6C47  X (X$)
6C48-6C4F  W (W$)
6C50-6C57  V (V$)
6C58-6C5F  U (U$)
6C60-6C67  T (T$)
6C68-6C6F  S (S$)
6C70-6C77  R (R$)
6C78-6C7F  Q (Q$)
6C80-6C87  P (P$)
6C88-6C8F  O (O$)
6C90-6C97  N (N$)
6C98-6C9F  M (M$)
6CA0-6CA7  L (L$)
6CA8-6CAF  K (K$)
6CB0-6CB7  J (J$)
6CB8-6CBF  I (I$)
6CC0-6CC7  H (H$)
6CC8-6CCF  G (G$)
6CD0-6CD7  F (F$)
6CD8-6CDF  E (E$)
6CE0-6CE7  D (D$)
6CE8-6CEF  C (C$)
6CF0-6CF7  B (B$)
6CF8-6CFF  A (A$)
6D00-6D5F  Buffer, for output to display
6D60-6DFF  Additional buffer space, serial I/O
6E00-6E05  ?
6E06-6E5F  FOR/NEXT stack (Five 18-byte loop registers)
6E60-6EAF  String Buffer, 80 bytes
6EB0-6EFF  Input Buffer, 80 bytes
6F00       Flags
6F01-6F02  Pointer to start of BASIC program (low byte first)
6F03-6F04  Pointer to end of BASIC program
6F05-6F06  Pointer to start of editable (MERGEd) program
6F07-6F08  Pointer to start of defined variables and arrays
6F09-6F10  Stored Password
6F11       Flags: 02=Down-arrow-key execution
6F12       Flags: 08=PRINT=LPRINT
6F13       Flags: 04=WAIT
6F14       Flags: 20=Password-protected
6F15       Flags
6F16       Flags: 40=TEXT Mode; 80=Katakana Mode
6F17       Flags: 02=Display cursor enabled (except for INPUT);
           04=Display cursor enabled; 20=Redisplay expression
6F18-6F1B  ?
6F1C-6F1D  Pointer to start of program being executed
6F1E       Code for MEM$, "B" or "C"
6F1F       ?
6F20-6F21  Pointer to INPUT or assignment statement
6F22-6F23  Pointer to DATA item
6F24-6F26  Pointer to variable used in assignment statement
6F27-6F28  Pointer to variable used in INPUT statement
6F29       ?
6F2A       Pointer to Input Buffer address following INPUT prompt
6F2B       Pointer to FOR/NEXT stack, low byte
6F2C       Pointer to GOSUB stack ,low byte
6F2D       Pointer to pending operand stack, low byte
6F2E       Pointer to pending operation stack, low byte
6F2F       Pointer to String Buffer, low byte
6F30       ?
6F31       User-defined-key label being sought
6F32-6F35  ?
6F36       Code of character being flashed
6F37       ?
6F38       Pointer to Input Buffer, low byte
6F39       USING, 00=scientific notation
6F3A       USING, No. of positions preceding decimal point
6F3B       USING, No. of positions for decimal point and following
6F3C       USING, No. of positions for character strings
6F3D-6F3F  ?
6F40-6F43  GCURSOR coordinates
6F44-6F4B  ?
6F4C       Column for display of immediate-mode result
6F4D       ?
6F4E       Column for next LPRINT
6F4F       Pointer to Input Buffer during INPUT
6F50       ?
6F51-6F52  Codes for name and type of specified array
6F53-6F54  ?
6F55-6F56  Pointer to start of RESERVE memory
6F57       Code of last key used
6F58       ?
6F59-6F5B  JMP instruction, used by system M.L.
6F5C-6F5F  Coordinates of endpoint of previous LINE
6F60-6F61  ?
6F62       Flags for LINE, PSET: 01=set; 02=reset; 04=reverse;
           08=square; 20=specification for line type
6F63-6F66  Coordinates for SET, GPRINT, or start of LINE
6F67-6F6A  Coordinates for end point of LINE
6F6B-6F6E  ?
6F6F-6FFF  RESERVE Memory (6FFF reserved as end marker)
7000-701D  Display Buffer, first fifth of top row
701E-703B  Display Buffer, first fifth of 3rd row
703C-703F  Not used
7040-705D  Display Buffer, first fifth of 2nd row
705E-707B  Display Buffer, first fifth of 4th row
707C-707F  Not used
7080-7087  Floating-point accumulator (numeric or string pointer)
7088-708F  ?
7090-70A3  GOSUB Stack (ten return addresses)
70A4-70A5  Binary-coded-decimal idle-time counter, non-resetting
70A6       Key-repeat timing control
70A7       ?
70A8-70AF  Floating-point register, last calculated result
70B0-70B1  Auto power-off counter, low byte first
70B2       ?
70B3-70B4  WAIT time setting, low byte first
70B5-70B6  Pointer to RETURN statement (or Input Buffer)
70B7-70BB  ?
70BC-70BD  Pointer to VAL or STR$ in expression
70BE-70BF  ?
70C0-71FF  No memory
7200-721D  Display Buffer, 2nd fifth of top row
721E-723B  Display Buffer, 2nd fifth of 3rd row
723C-723F  Not used
7240-725D  Display Buffer, 2nd fifth of 2nd row
725E-727B  Display Buffer, 2nd fifth of 4th row
727C-727F  Not used
7280-7297  Character codes for result of immediate calculation
7298-729F  Current value of random number
72A0-72B8  Numeric results, formatted for display or printing
72B9-72BF  ?
72C0-73FF  No memory
7400-741D  Display Buffer, 3rd fifth of top row
741E-743B  Display Buffer, 3rd fifth of 3rd row
743C-743F  Not used
7440-745D  Display Buffer, 3rd fifth of 2nd row
745E-747B  Display Buffer, 3rd fifth of 4th row
747C-747F  Not used
7480-7497  Buffer, for output to printer
7498-7499  Pointer to end of BASIC statement just executed
749A-749B  Pointer to BASIC at halt in execution
749C-749D  Pointer to error location in BASIC program
749E-749F  ?
74A0-74AF  Pending-operation stack, 16 codes
74B0-74BF  ?
74C0-75FF  No memory
7600-761D  Display Buffer, 4th fifth of top row
761E-763B  Display Buffer, 4th fifth of 3rd row
763C-763F  Not used
7640-765D  Display Buffer, 4th fifth of 2nd row
765E-767B  Display Buffer, 4th fifth of 4th row
767C-767F  Not used
7680-76BF  Pending operand stack, 8 floating-point registers
76C0-77FF  No memory
7800-781D  Display Buffer, last fifth of top row
781E-783B  Display Buffer, last fifth of 3rd row
783C       Flags: 01=SHIFT; 02=DEF; 04=PRINT; 08=RESERVE; 10=RUN;
           20=PRO; 40=Katakana; 80=SML
783D       Flag
783E-783F  Not used
7840-785D  Display Buffer, last fifth of 2nd row
785E-787B  Display Buffer, last fifth of 4th row
787C       Flags: 01=DEGREE; 02=RADIAN; 04=GRAD
787D-787F  Not used
7880-7881  Row and column of flashing cursor
7882-788A  Display control
788B-788C  CURSOR coordinates
788D-788E  Pointer to Display Buffer
788F-789F  Display control
78A0-78AF  ?
78B0       Serial I/O flag: 80=OPEN
78B1       Code for end of text
78B2       Baud rate: 01=300, 02=600, 04=1200
78B3       Endcode: 01=F, 02=L, 04=C; 10=2 stopbits;
           Parity: 40=O, 60=E; 80=8-bit word
78B4-78B5  Console specification
78B6-78BF  ?
78C0-7FFF  No memory
8000-8005  ROM; filled with 01
8006-8089  ML Jump instructions
808A-8269  Table: codes generating display characters
826A-83F4  Table: codes for Katakana characters
83F5-8402  Codes for "BREAK IN" and "ERROR"
8403-8443  Key codes
8444-8484  Key codes, Katakana mode
8485-84C5  Key codes, SHIFTed
84C6-8506  Key codes, Katakana mode, SHIFTed
8507-8547  Key codes, SML mode
8548-8588  Key codes, SML mode, SHIFTed
8589-85A2  Table: codes for A to Z when used with DEF key
85A3-A547  ML code: operating system, cassette routines
A548-A57B  Table: address of token word, for given initial letter
A57C-A637  Table: address of token word, for given token
A638-A91B  Table: token words, tokens, and routine addresses
A91F-A9DE  Code conversion table for PC-1250 cassettes
A9DF-AE9B  ML code
AE9C-AEBF  Codes for "RESERVE MODE", "PROGRAM MODE", "RUN MODE"
AEC0-FF82  ML Code; Routines for BASIC commands and statements;
           Display, printer, graphics, and serial I/O
FF83-FFFF  Unused ROM (filled with code CE)
```

FOR PC-1500 & PC-2 USERS

Program *PC! Solve.*

```
38D0 F2 E9 78 50
38D4 00 BE D0 2B
38D8 E9 78 80 00
38DC E9 76 4E FE
38E0 FD 58 B5 0A
38E4 FD CA FD 6A
38E8 4A 12 8E 12
38EC 20 20 20 20
38F0 20 20 20 20
38F4 50 43 21 20
38F8 53 4F 4C 56
38FC 45 52 BE ED
3900 3B BE E2 43
3904 C3 42 B7 0D
3908 9B 09 FD C8
390C BE D0 2B B5
3910 40 AE 78 80
3914 FD 8A 8E 08
3918 BE E8 CA BE
391C E2 43 C3 42
3920 EB 7B 0E 40
3924 B7 08 89 15
3928 5E B0 9B 14
392C 5E B1 89 03
3930 56 9E 1B 56
3934 56 15 B7 E0
3938 93 22 54 9E
393C 25 B7 0C 89
3940 0D 15 B7 0D
3944 9B 2B B7 E0
3948 81 01 54 54
394C 9E 36 E9 7B
3950 0E BF B7 1C
3954 89 05 BE CD
3958 E6 9E 43 B7
395C 1D 89 0D 15
3960 B7 E0 81 03
3964 BE CE 38 BE
3968 CE 38 9E 54
396C B7 0D 8B 25
3970 B7 1A 89 05
3974 BE D0 2B 9E
3978 61 B7 20 91
397C 62 5E FF 9B
3980 66 FD C8 15
3984 B7 E0 81 08
3988 FD 8A 51 BE
398C CE 38 9E 78
3990 FD 8A 51 9E
3994 7D EB 76 4E
3998 01 BE F9 57
399C BE CC DE 68
39A0 7B 6A FF B5
39A4 58 27 8B 0C
39A8 66 6E B0 99
39AC 08 EB 78 50
39B0 FF BA 3C 02
39B4 68 7B 6A FF
39B8 B5 2C 27 8B
39BC 04 88 05 8E
39C0 14 64 B5 50
39C4 27 8B 0F FD
39C8 28 FD 5A DE
39CC 09 48 79 4A
39D0 A8 BE DC 0C
39D4 8E 08 EC 48
39D8 79 4A A8 8E
39DC 02 9E C7 BE
39E0 DC 0C EC B5
39E4 FC AE 7A 00
39E8 B5 10 AE 7A
39EC 02 48 79 4A
39F0 00 BE DC 0C
39F4 EC B5 FF AE
39F8 7A 00 B5 50
39FC AE 7A 02 48
3A00 79 4A B0 BE
3A04 DC 0C 48 79
3A08 4A A8 BE DC
3A0C 20 48 79 4A
3A10 30 BE DC 0C
3A14 48 79 4A 28
3A18 BE DC 0C EC
3A1C 48 79 4A C8
3A20 BE DC 0C EC
3A24 E6 48 79 4A
3A28 C8 BE DC 20
3A2C B5 04 BE D0
3A30 D2 89 26 8E
3A34 02 9E 5A 48
3A38 79 4A B0 BE
3A3C DC 20 E6 48
3A40 79 4A 30 BE
3A44 DC 20 BE EF
3A48 B6 48 79 4A
3A4C 30 BE DC 0C
3A50 48 79 4A 10
3A54 BE DC 0C 8E
3A58 1B 48 79 4A
3A5C 30 BE DC 20
3A60 48 79 4A 10
3A64 BE DC 0C CD
3A68 54 CD 6A CD
3A6C 56 48 79 4A
3A70 C8 BE DC 0C
3A74 BE 3A 9A 48
3A78 79 4A B0 BE
3A7C DC 20 E6 48
3A80 79 4A 28 BE
3A84 DC 20 F0 48
3A88 79 4A 28 BE
3A8C DC 0C 48 79
3A90 4A 10 BE DC
3A94 0C BE 3A 9A
3A98 9E 41 CD A6
3A9C C9 42 48 79
3AA0 4A 10 BE DC
3AA4 20 48 79 4A
3AA8 B8 BE DC 0C
3AAC BE 3B A1 48
3AB0 79 4A 08 BE
3AB4 DC 20 48 79
3AB8 4A C0 BE DC
3ABC 0C 48 79 4A
3AC0 10 BE DC 20
3AC4 E6 48 79 4A
3AC8 00 BE DC 20
3ACC F0 48 79 4A
3AD0 B8 BE DC 0C
3AD4 BE 3B A1 48
3AD8 79 4A 10 BE
3ADC DC 20 48 79
3AE0 4A 18 BE DC
3AE4 0C 48 79 4A
3AE8 C0 BE DC 20
3AEC E6 48 79 4A
3AF0 00 BE DC 20
3AF4 CD 7E BE DB
3AF8 F5 48 79 4A
3AFC C0 BE DC 20
3B00 E6 48 79 4A
3B04 08 BE DC 20
3B08 BE EF B6 E6
3B0C CD 30 CD 58
3B10 83 9D E6 48
3B14 79 4A 18 BE
3B18 DC 20 BE EF
3B1C B6 48 79 4A
3B20 10 BE DC 0C
3B24 48 79 4A 10
3B28 BE DC 20 E6
3B2C 48 79 4A 18
3B30 BE DC 20 BE
3B34 EF B6 BE F5
3B38 97 E6 EC B5
3B3C F8 AE 7A 00
3B40 B5 10 AE 7A
3B44 02 CD 54 CD
3B48 66 B5 06 BE
3B4C D0 D2 99 B6
3B50 48 79 4A 10
3B54 BE DC 20 BE
3B58 F5 BE E6 48
3B5C 79 4A 10 BE
3B60 DC 20 BE EF
3B64 B6 CD 54 BE
3B68 F7 53 B5 F8
3B6C AE 7A 10 B5
3B70 10 AE 7A 12
3B74 B5 02 BE D0
3B78 D2 8B 09 48
3B7C 79 4A 10 BE
3B80 DC 20 BE 8D
3B84 48 79 4A 10
3B88 BE DC 20 E9
3B8C 76 4E FE B5
3B90 20 AE 78 80
3B94 BE E8 CA BE
3B98 E2 43 C3 42
3B9C EB 76 4E 01
3BA0 9A 58 7B 5A
3BA4 B0 DE 08 48
3BA8 79 4A 08 BE
3BAC DC 0C 9A 58
3BB0 7B 5A B0 BA
3BB4 39 18 48 79
3BB8 4A 10 BE DC
3BBC 20 E6 BE F5
3BC0 BE CD 54 CD
3BC4 66 BE EF B6
3BC8 E6 EC B5 F8
3BCC AE 7A 00 B5
3BD0 10 AE 7A 02
3BD4 CD 54 CD 66
3BD8 B5 02 BE D0
3BDC D2 89 0A 48
3BE0 79 4A 10 BE
3BE4 DC 20 BA 3C
3BE8 11 E9 76 4E
3BEC FE 48 79 4A
3BF0 10 BE DC 20
3BF4 B5 20 AE 78
3BF8 80 BE E8 CA
3BFC BE E2 43 C3
3C00 42 9A 58 7B
3C04 5A B0 DE 02
3C08 8E 06 58 7B
3C0C 5A B0 BA 39
3C10 18 48 79 4A
3C14 60 BE DC 0C
3C18 48 79 4A 60
3C1C BE DC 20 BE
3C20 F5 BE 48 79
3C24 4A 68 BE DC
3C28 0C CD 54 CD
3C2C 6A CD 56 48
3C30 79 4A 70 BE
3C34 DC 0C 48 79
3C38 4A 90 BE DC
3C3C 0C 48 79 4A
3C40 98 BE DC 0C
3C44 EC 48 79 4A
3C48 88 BE DC 0C
3C4C 48 79 4A A0
3C50 BE DC 0C 48
3C54 79 4A 68 BE
3C58 DC 20 E6 48
3C5C 79 4A 60 BE
3C60 DC 20 BE EF
3C64 B6 48 79 4A
3C68 78 BE DC 0C
3C6C 48 79 4A 98
3C70 BE DC 20 E6
3C74 48 79 4A 68
3C78 BE DC 20 CD
3C7C 7E E6 48 79
3C80 4A 88 BE DC
3C84 20 F0 48 79
3C88 4A A8 BE DC
3C8C 0C 48 79 4A
3C90 A0 BE DC 20
3C94 E6 48 79 4A
3C98 68 BE DC 20
3C9C CD 7E E6 48
3CA0 79 4A 90 BE
3CA4 DC 20 F0 48
3CA8 79 4A B0 BE
3CAC DC 0C 48 79
3CB0 4A B0 BE DC
3CB4 20 E6 48 79
3CB8 4A A8 BE DC
3CBC 20 CD 58 E6
3CC0 48 79 4A 60
3CC4 BE DC 20 CD
3CC8 54 CD 66 BE
3CCC EF B6 48 79
3CD0 4A 58 BE DC
3CD4 0C EC B5 F8
3CD8 AE 7A 00 B5
3CDC 10 AE 7A 02
3CE0 E6 48 79 4A
3CE4 58 BE DC 20
3CE8 BE F5 97 B5
3CEC 01 BE D0 D2
3CF0 89 99 CD A6
3CF4 C9 42 48 79
3CF8 4A 78 BE DC
3CFC 20 E6 48 79
3D00 4A 70 BE DC
3D04 20 CD 58 BE
3D08 F5 BE 48 79
3D0C 4A 68 BE DC
3D10 0C 48 79 4A
3D14 78 BE DC 20
3D18 48 79 4A 48
3D1C BE DC 0C 48
3D20 79 4A 78 BE
3D24 DC 20 E6 48
3D28 79 4A 68 BE
3D2C DC 20 CD 7E
3D30 E6 48 79 4A
3D34 70 BE DC 20
3D38 BE EF B6 48
3D3C 79 4A 78 BE
3D40 DC 0C 48 79
3D44 4A 48 BE DC
3D48 20 48 79 4A
3D4C 70 BE DC 0C
3D50 48 79 4A 98
3D54 BE DC 20 48
3D58 79 4A 88 BE
3D5C DC 0C 48 79
3D60 4A A8 BE DC
3D64 20 48 79 4A
3D68 98 BE DC 0C
3D6C 48 79 4A A0
3D70 BE DC 20 48
3D74 79 4A 90 BE
3D78 DC 0C 48 79
3D7C 4A B0 BE DC
3D80 20 48 79 4A
3D84 A0 BE DC 0C
3D88 BA 3C 6C F2
3D8C B5 20 AE 78
3D90 80 48 79 4A
3D94 B0 BE DC 20
3D98 E6 48 79 4A
3D9C A8 BE DC 20
3DA0 CD 58 BE E8
3DA4 CA E9 78 75
3DA8 00 E9 76 4E
3DAC FE 48 79 4A
3DB0 A8 BE DC 20
3DB4 BE D9 CF DC
3DB8 24 FD 6A CD
3DBC 92 B5 2F BE
3DC0 ED 4D 48 79
3DC4 4A B0 BE DC
3DC8 20 BE D9 CF
3DCC DC 24 FD 6A
3DD0 CD 92 BE E2
3DD4 43 C3 42 A5
3DD8 78 50 EB 76
3DDC 4E 01 B7 FF
3DE0 8B 01 9A E9
3DE4 76 4E FE B5
3DE8 40 AE 78 80
3DEC BE D0 2B E9
3DF0 78 50 00 BA
3DF4 39 18
```

PC! SOLVE

Eric Bowman, PEH Box 81, Exeter, NH 03833, submitted this program that he calls: PC! Solve. Here is how he describes its use and operation:

PC! Solve is a versatile equation solving and fractions program. It is written entirely in machine language, thus it is fast. The program can evaluate an expression and display the result as a decimal and fraction or solve an equation with real roots for an unknown.

Preparing the Program for Utilization

First, initialize memory in your PC by entering the command NEW &3DF6. Then, load the program. If loading from the accompanying listing, use POKE statements or utilize a Monitor program. Note that the program is *not* relocatable. Once loaded, start the program by executing: CALL &3800. When started, the program will display "PC! SOLVE" as an indication that it is ready to go. The next step is to enter an equation to be solved or an expression to be evaluated.

Solving An Equation

To solve an equation, enter the equation via the keyboard. All BASIC syntax is applicable here. You must represent the unknown by the symbolic variable "X" or the expression will be evaluated instead of solved. Note that the program is designed to solve for a root so the equation must be set equal to the value zero. Thus, to find the root of 4X=16, you must input 4*X-16 and then press the ENTER key.

If, in fact, you try the above (4*X-16), in about half a second you should see the number 4 appear on the display as 4 is the solution to that simple example.

The solve routine also has the capability for accepting user-defined starting points. If a function is suffixed with ",n" the routine will begin searching for a root at "n" along the X-axis. This capability is particularly valuable if the function to be solved takes the logarithm of X or divides by X or a multiple of X. For instance, to solve 3/X=6, you could input: 3/X-6,1. This causes the search for the root to begin at X=1 (instead of at X=0, which would result in an error). After a root is displayed, the search will continue for another case upon depression of any key except BREAK.

If a root is not an integer, it will be displayed as a fraction (reduced to lowest terms) and a decimal.

The method used to solve equations in this routine is known as Newton's Method. Note that any root located is only accurate to plus/minus 0.00000001. Therefore, this method is only an approximation of the root. For many practical purposes, however, this method is quite adequate.

Converting To A Fraction

If the letter "X" is not used when entering an expression, the expression will be evaluated once. The result will be displayed in both fractional and decimal formats. For instance, to find the fractional form of 0.111111111 . . ., the user would type: 0.11111111 and press the ENTER key. In about half a second, "1/9" will be displayed. Alternately, the user could elect to evaluate an expression such as this: ASN ((1/SQR(2))^2. Pressing the ENTER key following the inputting of these terms would result in "1/2" being shown.

Accuracy is again to 0.00000001. In the first example, if 0.1111111 was inputted (seven 1's after the decimal instead of eight), the result would not be "1/9" but rather 1111111/100000000. For repeating decimals, the entry accuracy should be to 10^(-8).

Errors

If there is an error, either in the solving process or in the initial evaluation of an expression, the inputted data will be re-displayed with the cursor flashing over the first character. If you are trying to solve, look for trouble spots such as division by zero or the extraction of a logarithm of a negative number. When the expression has been corrected, press ENTER to restart the process. Note that the editing features normally found when using BASIC will be in effect during such correction operations. If you want to erase the entire entry, type SHIFT/CL. (Note that as a safety precaution, use of the CL key alone has no affect when in this correction mode.)

Eric indicates that he welcomes inquiries from readers that have any questions or problems using the program or who have suggestions on improving the program.

FROM THE WATCH POCKET

If you ever wished you could have a *lot* more memory in your HP-71B computer, you should know that now you can. An outfit called **Hand Held Products, Inc.**, *P.O. Box 2389, Charlotte, NC 28211,* has announced a line of products that allow you to add as much as 96K (wow!) of memory to the -71B. You can also mix combinations of RAM and ROM. The expansion memory mounts in place of the magnetic strip reader module. Prices are pretty stiff -- it will cost you $995.00 to purchase the 96K module -- but if you are, say, developing programs in Forth or working in a commercial environment, you might find the extra memory well worth the price. Write to the company to obtain literature on these memory expansion products.

Word is floating around that Sharp is close to releasing a model called the PC-2500. The story goes that it will have a multiple-line display, a built-in 4-inch wide combination plotter/printer, and a keyboard large enough to touch-type on. (Sounds like a lap-sized portable!) One other tid-bit: the CPU will be the same one used in the 1250/1260/1350! Sure hope this turns out to be true.

POCKET
COMPUTER
NEWSLETTER

Issue 40 - May/June

WORLD'S FIRST POCKET COMPUTER WITH TOUCH PAD SCREEN

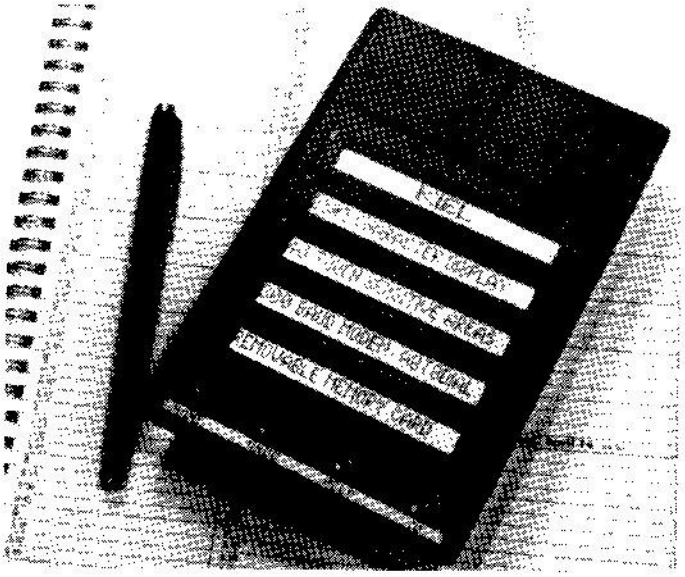
Kiel Corporation, Inc., a one year old company based in Nashua, New Hampshire, has announced a hand-held computer that features a built-in combination display and touch pad.

The model, dubbed the Touch Pad I, features five display lines capable of displaying 25 characters each. Each display line also serves as a touch pad having 15 touch sensitive areas per line.

The unit also has a number of other unusual features. A removable Program Memory Card permits the Touch Pad I to be expanded to a maximum memory capacity of 120 kilobytes. Up to 56 kilobytes of this may be RAM elements, the remaining 64 kilobytes consisting of ROM. The Program Memory Card is available in 8K increments. The Program Memory Card is just 3 millimeters thick and about the size of a standard business card. Memory cards may be programmed using IBM PCs or similar microcomputers as well as by downloading to RAM via the unit's optional modem.

The Touch Pad I can be equipped with an *internal* modem capable of operating at either 1200 or 300 baud. Programs and data may be loaded into or transferred out of the unit via the modem.

Photo *Kiel Corporation's Touch Pad I Hand-Held Computer.*



The hand-held Touch Pad I measures just 6 by 4 by 3/4 inches. It accepts 7.5 to 12 volts direct current (unregulated) from an external source or it may be powered by a removeable battery pack. Current drain is 300 milliamperes (maximum) in the standard configuration. A lower power CMOS version is also available.

The unit utilizes an 8031/51 microprocessor. It comes standardly equipped with 2 kilobytes of RAM and 4 kilobytes of ROM. Kiel Corporation supplies a "program generator" that enables users to develop programs for the package on a larger system such as a desktop computer.

The liquid-crystal display can show variable width characters. Each line contains 960 pixels (120 by 8). Each line is also sectioned into 15 touch sensitive areas that may be used for input. Thus, the display can serve as a keyboard, keypad, function panel, etc., under the control of software.

The unit is also equipped with four user-defined function keys. And, it has a dual tone buzzer for providing audible alerts.

The Touch Pad I can also be optionally equipped with a bar code reader.

The manufacturer considers the product ideally suited for applications such as: building security and control, keyboard and interface to complex instrumentation (such as medical equipment), home banking terminal, weighing and packaging, factory data collection, service reporting, time management activities, robotics, point of sale terminal and a variety of control and monitoring tasks.

The single unit price of this unique touch-sensitive hand-held computer starts at $650.00. In quantity, such as in OEM applications, the price for a basic unit drops down to the $200 range.

Kiel Corporation is developing other products that take advantage of its proprietory touch-sensitive technology. Currently under development are liquid-crystal display touch panels that include "smart drivers" capable of displaying 40 characters per line. These panels can be equipped with 4 or 8 lines with all electronics integral to the panel. Such a panel, equipped with 8 lines of 40 characters, can contain 160 touch-sensitive positions. The company also expects to release a graphics panel that incorporates touch-sensitive features.

To obtain more information on the Touch Pad I, contact: **The Kiel Corporation**, *472 Amherst Street, P.O. Box 6340, Nashua, NH 03063*. The phone number is (603) 881-8666.

FOR PC-1250/51/60/61 USERS

MONITOR PROGRAM FOR 1250/1251

If you really like to dabble with using machine language in your Sharp PC-1250, this program can make life a lot easier than resorting to PEEKs and POKEs. The program was crafted by *Rick Wenger, 6221 - 18th Avenue, Kenosha, WI 53140,* who also supplies the following information about its use and operation.

Loading The Program

Load in the BASIC portion of the monitor from the accompanying listing. Enter the program *exactly* as shown. The program is address-dependent so do not attempt to add or delete anything.

After the BASIC part has been entered, switch to the RSV (reserve memory) mode and set up the following two reserve keys:

SHIFT C: CALL 50312
SHIFT N: CALL 50291 a

Now switch back to the PRO mode (or RUN mode) and use POKE directives to start loading in the machine code portion as shown in the accompanying listing. As indicated in the listing, once you get to a certain point, you can switch over and start using the monitor program itself to key in the balance of the machine code!

Of course, you can store the program on tape. Use a standard CSAVE command to save the BASIC part. Use the statement: CSAVE M "MLMON1";&C40F,&C5CF to store a copy of the machine code portion on tape.

Monitor Instructions

The monitor program provides the following capabilities by selecting the appropriate "defined" key.

DEF M: Preceded by a four-digit hexadecimal address puts you in monitor mode. Thereafter you may move to any address by keying the four digits (of the address) and hitting the ENTER key. Pressing ENTER without an input advances the current address by four. (You can change the number of locations-per-advance within the range 0 through 9 by keying a single digit followed by ENTER. An advance of five is useful when viewing the display memory. An advance of one is nice when entering machine code.) Entering machine language into RAM is accomplished by simply positioning the monitor to the desired address and entering a two-digit hexadecimal code followed by the ENTER key.

DEF SPC: Decrements the monitor address by the current locations-per-advance amount.

DEF F: Preceded by BRK, two digits, fills the temporary CPA with the specified hexadecimal byte. When not preceded by BRK and two digits, it just moves the monitor to the temporary registers area.

DEF B: Puts a breakpoint at the current monitor address.

DEF D: Deletes the last breakpoint reached.

DEF J: Preceded by BRK, four digits, causes a jump to the specified hexadecimal address. When not preceded by BRK and four digits, jumps to the previous DEF J address. Acts as a continue function when used immediately after DEF D. The command DEF J *must* be used if a breakpoint is present in a program. It should *not* be used if a routine ends with a RTS instruction.

DEF S: When preceded by BRK, two digits, searches for the specified hexadecimal byte starting from the current monitor address. If not preceded by BRK, two digits, resumes search for previously indicated search byte from the current monitor address.

DEF N: Provides hardcopy of memory contents via the printer. Output begins at current monitor address unless the command is preceded by BRK and four digits representing the desired starting address. Printing continues until user hits BRK or top of memory (FFFF) is reached.

DEF =: Converts the decimal number in the display to its four-digit hexadecimal equivalent.

SHIFT C: Pseudo-Clear function. Must be used in place of CLEAR in your BASIC programs that reside with the monitor program. Protects the machine language portion of the monitor. Should also be at the head of your program if you intend to use RUN or dimension any variables.

Program ***BASIC Portion of PC-1250/1251 Monitor.***

```
 1:"Y" CALL 50611:D=&00
   00: RETURN
 2:GOSUB "Y": POKE C,D:
   C=C+1
 3:""B= INT (C/256):
   POKE 50426,C-256B,B:
   CALL 50528
 4:B=0: INPUT "0F  0010
   :02 5C 32 4C ";B$:
   ON LEN B$ GOTO 11,2,
   3,6
 5:C=C+A: GOTO "
 6:"M" AREAD B$: GOSUB
   "Y":C=D: POKE 49285,
   194:A=4: GOTO "
 7:"J" AREAD B$: IF LEN
   B$=4 GOSUB "Y":B=
   INT (D/256): POKE 50
   357,B,D-256B
 8:CALL 50329: GOTO 10
 9:"F" AREAD B$: IF LEN
   B$=2 GOSUB "Y": POKE
   50283,D: CALL 50282
10:C=50428: GOTO "
11:A= VAL B$: GOTO 5
12:" "C=C-A: GOTO "
13:"B" CALL 50247: GOTO
   "
14:"D" CALL 50226: GOTO
   "
15:"=" AREAD D:B= INT (
   D/256): POKE 50426,B
   -256B,B: CALL 50394:
   PRINT B$: GOTO "
16:"S" AREAD B$: IF LEN
   B$=2 GOSUB "Y": POKE
   50214,D
17:CALL 50191:C= PEEK 5
   0426+256* PEEK 50427
   : GOTO "
18:"H" AREAD B$: IF LEN
   B$=4 GOSUB "Y":C=D
19:A=5: POKE 49285,159:
   GOTO "
```

Program ***Machine Code Portion of PC-1250/1251 Monitor.***

```
Execute CLEAR <ENTER>
Execute DIM Z$(224)*2 <ENTER>
Check MEM=491 here
POKE in the following hexadecimal code (prefix each value with G sign):
C4DA: 84 10 C4 FA 1A 84 78 C5 A1 82 10 C6 93 1B 85 78
C4EA: C5 A1 82 11 91 1B 02 F5 11 90 52 23 11 95 52 37
C56O: 78 C4 DA 88 00 03 10 C6 91 18 88 10 C0 8B 19 85
C57O: 00 04 63 E0 05 2A 09 82 13 04 0A 88 35 2C 03 88
C58O: 18 88 78 C5 A1 10 C0 87 82 1B 02 C0 87 DB 02 8F
C59O: 86 DB 89 00 03 78 C5 A1 26 DA 26 06 50 41 2B 0A
C5AO: 37 59 78 C5 A8 DA 59 58 64 0F 74 40 67 4A 3A 03
C5BO: 74 07 37 00 04 84 02 40 1E 88 10 C6 91 18 87 50
C5CO: 63 00 29 04 51 51 51 51 10 C0 40 00 03 19 37 FA
At this point you can activate the Monitor to key in the rest of the
machine code. Key in: C40E DEF M 1 <ENTER>
Now use the Monitor to enter the following hexadecimal values. (You do
not need to prefix the values with the G sign when using the Monitor.)
C40F: 10 C4 FA 84 1A 00 00 04 88 55 85 63 40 2A 07 82
C41F: 13 04 0A 88 35 88 63 00 29 12 10 C4 FA 84 1B 37
Key in: C432 <ENTER> (to change address when using the Monitor)
C432: 10 C4 2F 82 00 02 18 10 00 00 82 19 10 C4 3A 1A
C442: 85 11 B5 1B 37 10 C4 FA 86 1A 84 02 B7 DB 02 C4
C452: DA 02 79 82 00 02 07 06 19 10 C4 3B 86 53 11 3A
C462: 87 53 10 C4 2F 82 19 37 02 00
C46C: 00 60 10 C4 FF 1F 37 02 1C 03 C2 82 10 C6 E3 1B
C47C: 02 E0 10 C0 31 52 02 FF 10 C2 1C 52 02 08 03 C4
C48C: 10 C6 FC 82 1B 23 00 AF 10 C5 D0 1F 37 80 00 5F
C49C: 10 C5 00 19 11 02 52 10 C4 FC 57 30 11 FD 57 31
C4AC: 11 FE 57 32 10 C5 02 57 79 00 00 10 C5 02 52 20
C4BC: 10 C4 FC 52 21 11 FD 52 22 11 FE 52 80 10 C5 00
C4CC: 53 11 02 57 11 01 00 5E 81 19 02 58 32 37
To save machine language portion on tape: CSAVE M "MLMON1";6C40F,6C5CF
```

SHIFT N: Pseudo-NEW function. Initializes your BASIC program but retains and protects the monitor program. Can also be used immediately after a hasty NEW or a bad crash to retrieve the monitor from the grave. (You may have to manually key: CALL 50291 ENTER, if the crash blanks the RSV memory.

USER'S MEMORY MAP

Here is how memory is utilized when the monitor is installed in the PC-1250:

C031 - C21C: BASIC portion of the monitor. This part is not relocatable.

C21D - C407: Part of machine language portion of the monitor. Not relocatable.

C4FC - C4FF: Temporary CPU registers (P, P' and SP) and fill byte.

C500 - C55F: Temporary CPA (00 - 5F).

C560 - C5CF: Additional part of machine language portion of the monitor.

C5D0 - C67F: Variables Z through E. Available to user for use as variables or as machine language scratchpad area.

C680 - C69F: Variables D through A. Used by monitor.

Putting The Monitor Through Its Paces

First, try something simple. Key in 49494 DEF =. You should see C156. Now try keying 68888 + 62345 DEF =. Do you see ABCD? You have been using the hexadecimal calculator!

Now try keying CL (the "clear" key) and DEF M. You will be looking at the "hidden" ROM. The first four bytes should be: 4E A0 02 01. Hit ENTER to see the next four bytes: 12 5F DB DF. The byte to the far left in the display is the preceding byte (at address 0003 in this instance). Key DEF SPC to get back to address 0. Now, try a different increment value, say 5. Key the number 5 then hit ENTER several times. Key DEF SPC the same number of times to get back to the starting point.

Here is a simple machine language program. It will double any number in the hexadecimal range 0 through 7F:

ADDRESS	MNEMONIC	MACHINE CODE
C300	LDA (P)	59
C301	ADD(P) A	44
C302	BREAK	79 C4 B7

To enter the program into memory using the monitor, just key: C300 ENTER, 59 ENTER, 44 ENTER, DEF B.

The following discussion assumes that you have an understanding of the machine language instructions being used. (This program is not designed to communicate directly with BASIC.) DEF J will be used to jump to the program. Break will used to get back to the monitor. Thus, we can use the monitor for input and output. A program set up in this fashion is fairly (?) crash-proof, in that you do not need to worry about upsetting registers SP, (01), (16) or (58) through (5B).

Refer to the User's Memory Map provided earlier. Note that the temporary CPU registers at C4FC, C4FD and C4FE are transferred to the real CPU registers P, P' and SP and that the temporary CPA is transferred to the real CPA when the monitor is placed in operation. After execution of the program the monitor enables the temporary CPA to reflect the results of your program (while guarding against a crash that could occur if the CPA was manipulated directly).

To verify operation of the example program and the monitor, key: BRK 13 DEF F. The monitor address should show C4FC. Key 03 ENTER so that P will point to register (03) in the CPA. Key in whatever you want for P' and SP (these will remain unchanged during the following operations). Verify that C4FF through C55F contain 13. Now key: BRK C300 DEF J.

If all is well, the program has now executed and you are back at C4FC. C4FC should still be 03 and C500 through C55F all still contain 13 -- except for C503 -- which contains 26. Key DEF J again and C502 will become 26 while C503 goes to 4C. You can try different values for P and (P). The only registers affected by this program are the accumulator (represented by C502) and (P) which is represented by C500 + (C4FC).

To confirm breakpoint operation, load (P) with 05. Position the monitor to C301. Key DEF B. Verify that C301 - 03 contains 79 C4 B7. Key DEF J. Verify that A and (P) contain 05. Key DEF D. See that C301 - 03 now contains 44 79 C4. Load A with 02. Key DEF J. Verify that (P) now holds 07. If you realize why this is so, you have a good understanding of monitor operations. (This discussion refers to CPA locations. You must, of course, examine the temporary register equivalents.)

You may explore the DEF S and DEF H operations at your leisure.

Installing The Monitor In The PC-1251

You need to make a few modifications to adapt the program as shown for the PC-1250 for the PC-1251. First, change the POKE value in lines 6 and 19 of the BASIC listing from 49285 to 47237. Next, in the machine code listing, change all the underlined values of C0 to B8 and the underlined values of C2 to BA. That's all it takes!

A Bare Bones Monitor

You can make a skeleton version of the monitor which lacks the functions DEF F, DEF SPC, DEF B, DEF D, DEF =, DEF $ and DEF H. But, you can still view memory and enter machine code. Why, you can even enter breakpoints by hand! It might come in handy in situations where memory is really at a premium. To make the "bare bones" version, do the following:

1. Delete lines 9 through 19 from the BASIC listing.
2. Change line 8 to read: CALL 50329:C=50428:GOTO "
3. In line 4, (after the ON GOTO), change the number 11 to a quotation mark (").
4. Change locations C46C - C471 to: EA 01 61 AE 00 02.
5. Change C474 to 21, C476 to C1, C486 to 21, C485 to C1 and C489 to 6C.

PC-1500 TO PC-1250 THROUGH PC-1261 PROGRAM CONVERSIONS

Quite a few readers have asked whether programs previously published in *PCM* for the model PC-1500 can be converted to run on a PC-1250/51/60/61. The fact is, many of the programs written in BASIC for the PC-1500 can also be run on the PC-1250 series with little change. The limiting factors are generally the amount of memory needed, whether or not plot capabilities are being utilized (as the equivalent functions are not available on the 1250 series), and the amount and type of variable usage. Let's take a look at each of these factors to try and gauge their effect on adapting a program from one type of machine to the other.

Memory: Remember, a PC-1500 can be expanded by an extra 8 to 16 kilobytes. Large programs written for the PC-1500 simply won't fit in a 1250 series PC that doesn't have sufficient memory. As a rule of thumb, allowing some margin for handling conversion problems, you can fit a 1500 byte PC-1500 program into a PC-1250, a 3000 byte PC-1500 program into a PC-1251 (or Radio Shack PC-3A) and a 9000 byte PC-1500 program into a PC-1261. These byte counts include storage used by any arrays being utilized in the program(s).

Printer/Plotter: The CE-150 printer/plotter interface typically used with the PC-1500 has capabilities that can not be duplicated on the CE-125 or CE-126P printer. Examine programs you are thinking of converting to see if they use plotter capabilities. If so, can the program still be useful if those capabilities are dropped? If not, look for another program to convert!

Variable Usage: A major difference between the PC-1500 and the 1250 series is how the primary variables (standard variables A - Z and string variables A$ - Z$) are allocated. In the PC-1500 these variables are completely independent. There are two sets (one for variables A - Z, the other for string variables A$ - Z$). Furthermore, the string set in the PC-1500 has room for 16 characters in each element. Such is not the case in the PC-1250 series. There is one set of memory locations that serves to hold the variables A - Z or A$ - Z$. If a variable is being used to hold a numerical value, it cannot simultaneously serve to hold a string value and vice versa. Furthermore, when a variable is assigned to serve as a string element, only 7 characters can be stored. This reduced capability on the PC-1250 series can make it difficult to convert a program that utilizes the standard string registers extensively in a PC-1500 application. However, such adaptations can be made by assigning two-character variables (such as A1$, B1$, C1$,... etc.) or array elements (such as A$(0), A$(1), A$(2),... etc.) in the PC-1250 series in place of the standard string variables (A$, B$, C$,... etc.) in the PC-1500.

Here are a few other tips you might want to consider when adapting programs from one model to the other: Watch out for "implied" multiplication. While the PC-1250 will interpret the statement AB as A times B, the PC-1260 would think you meant the two-character variable named AB. It is a good idea to always specifically invoke multiplication by using the multiplication (*) sign such as: A*B. Don't forget you can simply invoke printer operation on the PC-1250 series by giving the manual command PRINT = LPRINT. You can go back to using the display (for all print statements) by using the statement PRINT = PRINT. Thus, you do not have to complicate your PC-1250 programs with a whole bunch of flags and two sets of PRINT and LPRINT directives.

Simultaneous Equations

For those of you who like programming in BASIC (instead of machine language) here is a small but powerful program. It was originally developed by *Norlin Rober* for the Sharp PC-1211. But, as you shall see, it runs fine on a PC-1250, a 1251, 1260 or 1261. You use it to solve a system of equations having up to 12 unknowns. It only takes a little over 400 bytes for the program itself. Another 1300+ bytes are used by an array.

Use DEF/A to start the program. Respond to the prompts to input the number of equations and the coefficients for each one. For example, if you wanted to solve the system:

```
3W  +  2X  +  6Y  +  2Z  =  2
 W  +  4X  +  3Y  +  2Z  =  6
2W  +   X  +  9Y  +  3Z  =  2
4W  +  3X  +  6Y  +   Z  =  8
```

You would respond to the first prompt with 4 (for the number of equations) and then enter the coefficients in four sets as: (3, 2, 6, 2, 2), (1, 4, 3, 2, 6), (2, 1, 9, 3, 2) and then (4, 3, 6, 1, 8). If you have loaded the program properly you should obtain the solution: -2, 3, 2, -5.

If the coefficients entered are not acceptable, you will see the message REJECTED appear on the display. If so, enter any remaining sets of coefficients and then try re-entering the troublesome set. If the REJECTED message comes up again then the algorithm cannot solve the particular problem due to inconsistency or dependence. (The algorithm employs Gaussian elimination.)

Program *Simultaneous Equations*.

```
1:"A" DIM A(172):
  INPUT "NO. OF EQUATI
  ONS? ";A:A=A+1:B=0
2:"C" BEEP 1: FOR C=7
  TO A+6: INPUT A(C+A*
  B): NEXT C: IF B=0
  THEN 5
3:FOR D=1 TO B:E=A(A+A
  *B-D+6):F=A(A*D-D+6)
4:FOR C=7+A*B TO A+C-1
  :A(C)=A(C)-A(C+A*D-A
  -A*B)*E/F: NEXT C:
  NEXT D
5:IF A(C-B-1)=0 BEEP 1
  : PRINT "REJECTED":
  GOTO 2
6:B=B+1: IF A-B-1 GOTO
  2
7:FOR B=1-A TO -1:F=6-
  A*B:E=A(F)/A(F+B):A(
  F)=E: IF B+2 GOTO 9
8:FOR C=1 TO -B-1:F=A*
  C+6:A(F)=A(F)-A(F+B)
  *E: NEXT C: NEXT B
9:BEEP 1: FOR B=1 TO A
  -1:E=A(A*A-A*B+6):
  PRINT B;"  ";E: NEXT
  B: END
```

FOR HEWLETT-PACKARD HP-71B USERS

CURVE-FITTING PROGRAM

Engineers and scientists frequently need to fit a series of x,y data points to curves. What curve should you try? Linear? Logarithmic? Exponential or power? This powerful program can help you out!

This is a curve-fitting package that has been adapted specifically for the HP-71B. (The original version, designed for a Sharp PC-1500, was submitted by *Thomas S. Cox* and published in Issue 15 of ***PCN***.) Special thanks for assisting in the adaptation go to *Robert Findley*.

Load the program into your HP-71B from the accompanying listing. As a check on your accuracy, look for a byte count of 2329 upon keying in the program exactly as shown. Start the program using the RUN key.

Once running, you will see a six-item menu flashing on the display. Press the number key corresponding to the selection of your choice. Normally, you will start with item number 1 to enter a group of x,y points. You also make this selection if you want to add additional points to your curve later. (Select the CLEAR option if you are inputting a new set of data points. Respond negatively [N] to this prompt if you wish to add data points to previously entered data.)

To enter data points, respond appropriately to the prompts for point number (PT#), x- and y- values. Enter as many points as you want to obtain a decent fit. Input point number zero (PT# 0) to conclude the data entry operation. This will return you to the primary rotating menu.

Select menu item number 2 if you want to change data. You can eliminate or alter your latest entry point using this option. Just respond appropriately to the program's queries.

Menu item number 3 will automatically determine the best fit of your data to the four supported curves: linear, log, exponential or power. You will see the "fitting" procedure take place for each type of curve followed by the computer's selection of the best fit. The program then displays in order the constant equation term A, variable equation term B, coefficient of correlation R and coefficient of determination R^R for the curve selected. (Press the END LINE key to obtain each term.) Once these have been presented

Program *HP-71B Implementation of Curve-Fitting.*

```
10 DELAY .8,0 @ WIDTH 96 @ DISP "CURVE FIT MENU"
20 DISP "1-ENTER     2-CHANGE" @ IF KEYDOWN THEN 40
25 DISP "3-BEST      4-CURVE FIT" @ IF KEYDOWN THEN 40
30 DISP "5-EST X     6-EST Y"
40 IF NOT KEYDOWN THEN 20 ELSE Z$=KEY$
50 IF Z$<"1" OR Z$>"6" THEN 20 ELSE ON NUM(Z$)-48 GOSUB 80,480,380,60,420,550 @
60 DISP "CURVE MENU"
65 DISP "1-LINEAR    2-EXPON." @ IF KEYDOWN THEN 70 ELSE DISP "3-LOG.      4-POWER
70 IF NOT KEYDOWN THEN 65 ELSE Z$=KEY$
75 IF Z$<"1" OR Z$>"4" THEN 60 ELSE ON NUM(Z$)-48 GOSUB 180,230,280,340
80 DISP "ENTER DATA" @ INPUT "CLEAR (Y/N)? ","N";Z$
90 IF Z$="Y" THEN DESTROY ALL
100 C=1 @ Z=N+1 @ INPUT "(0 TO STOP) PT# ";Z
105 Z$="X"&STR$(Z)&"= " @ IF Z=0 THEN 10 ELSE DISP Z$; @ INPUT "","?";L @ IF L<=
110 X=L @ Z$="Y"&STR$(Z)&"= " @ DISP Z$; @ INPUT "","?";Y @ IF Y<=0 THEN T=1
120 GOSUB 130 @ GOTO 100
130 D=D+C*X @ E=E+C*X*X @ F=F+C*Y @ G=G+C*Y*Y @ H=H+C*X*Y @ N=N+1*C @ IF J*T=1 T
140 IF J#1 THEN M=M+C*LOG(X) @ O=O+C*LOG(X)*LOG(X) @ K=K+C*Y*LOG(X)
150 IF T#1 THEN P=P+C*LOG(Y) @ Q=Q+C*LOG(Y)*LOG(Y) @ I=I+C*X*LOG(Y)
160 IF J+T=0 THEN S=S+C*LOG(X)*LOG(Y)
170 RETURN
180 IF W=1 THEN 210
190 U=1 @ DISP "LINEAR Y=B*X+A"
200 B=(H-D*F/N)/(E-D*D/N) @ A=F/N-B*D/N
210 R=(N*H-D*F)/SQR((N*E-D*D)*(N*G-F*F)) @ IF W=1 THEN RETURN
220 GOTO 500
230 IF W=1 THEN 260
240 U=2 @ DISP "EXP. Y=A*EXP(B*X)" @ IF T=1 THEN 430
250 B=(I-P*D/N)/(E-D*D/N) @ A=EXP(P/N-B*D/N)
260 R=(N*I-D*P)/SQR((N*E-D*D)*(N*Q-P*P)) @ IF W=1 THEN RETURN
270 GOTO 600
280 IF W=1 THEN 310
290 U=3 @ DISP "LOG. Y=B*LOG(X)+A" @ IF J=1 THEN 430
300 B=(K-M*F/N)/(O-M*M/N) @ A=(F-B*M)/N
310 R=(N*K-F*M)/SQR((N*O-M*M)*(N*G-F*F)) @ IF W=1 THEN RETURN
320 GOTO 600
330 IF W=1 THEN 360
340 U=4 @ DISP "PWR. Y=A*X^B" @ IF T+J>0 THEN 430
350 B=(S-M*P/N)/(O-M*M/N) @ A=EXP(P/N-(B*M/N))
360 R=(N*S-M*P)/SQR((N*O-M*M)*(N*Q-P*P)) @ IF W=1 THEN RETURN
370 GOTO 600
380 IF J+T>0 THEN 430
390 W=1 @ V=0 @ U=0 @ FOR Z=1 TO 4 @ ON Z GOSUB 180,230,280,330 @ DISP Z;R @ C=R
395 IF C>V THEN GOSUB 650
```

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

the program returns to the main menu.

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

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

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

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

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

FOR PC-1500 & PC-2 USERS

SPEED COMPARISONS -- PC-1500 STILL TOPS

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

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

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

Table *Comparisons of PC Execution Times.*

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

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

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

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

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

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

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

Table *Comparisons of PC Tape Operation Times.*

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

A ROUTINE TO VERIFY MACHINE LANGUAGE SAVES

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

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

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

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

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

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

```
7DBD F2 BE D0 2B
7DC1 FD 58 B5 08
7DC5 FD CA 6A 05
7DC9 8E 06 54 49
7DCD 54 4C 45 3A
7DD1 58 7B 5A B0
7DD5 F5 88 03 B5
7DD9 40 AE 78 80
7DDD BE E8 CA BE
7DE1 E2 43 C3 42
7DE5 B7 18 9B 2C
7DE9 EB 7B 0E 40
7DED B7 08 89 07
7DF1 5E B6 9B 15
7DF5 56 9E 1B B7
7DF9 0C 89 08 B5
7DFD 0D 17 9B 21
7E01 54 9E 27 B7
7E05 0D 8B 1D E9
7E09 7B 0E BF B7
7E0D 1C 89 05 8E
7E11 CD E6 9E 38
7E15 B7 1D 89 05
7E19 BE CE 38 9E
7E1D 41 B7 20 91
7E21 42 51 9E 48
7E25 B5 00 BE BB
7E29 D6 48 7B 4A
7E2D B6 58 7B 5A
7E31 69 05 B7 0D
7E35 8B 04 51 44
7E39 9E 09 B5 C0
7E3D AE 78 79 CD
7E41 B0 83 40 A5
7E45 7B AA 08 A5
7E49 7B AB 0A A5
7E4D 7B AC 28 A5
7E51 7B AD 2A CD
7E55 AA 83 2C CD
7E59 B4 F2 FD 58
7E5D B5 0A FD CA
7E61 FD 6A 4A 15
7E65 8E 15 56 45
7E69 52 49 46 49
7E6D 43 41 54 49
7E71 4F 4E 20 43
7E75 4F 4D 50 4C
7E79 45 54 45 BE
7E7D ED 3B BE D0
7E81 2B CD 46 B7
7E85 23 F2 BE D0
7E89 2B CD B4 F2
7E8D FD 58 B5 0A
7E91 FD CA FD 6A
7E95 4A 0B 8E 0B
7E99 44 49 53 43
7E9D 52 45 50 41
7EA1 4E 43 59 BE
7EA5 ED 3B CD 46
7EA9 BE D0 2B CD
7EAD B4 F2 FD 58
7EB1 B5 0A FD CA
7EB5 FD 6A 4A 0E
7EB9 8E 0E 43 48
7EBD 45 43 4B 53
7EC1 55 4D 20 45
7EC5 52 52 4F 52
7EC9 BE ED 3B BE
7ECD E2 43 CD 46
```

FOR PC-1350 USERS

USING POKES ON THE PC-1350

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

Memory Reallocation

In an unexpanded PC-1350, BASIC normally starts at address 66030. This address is stored in a START OF BASIC pointer. This pointer is located at 66F01-02. Note that 66F01 contains 630, the low byte of this starting address. 66F02 contains 660, the high byte of this starting address.

BASIC can be relocated by changing the contents of this pointer. For example, execution of POKE 66F01,630,664, followed by execution of NEW (in the PRO mode) will set the START OF BASIC to 66430. Now any BASIC program, whether entered by CLOAD or from the keyboard, will be stored beginning at 66430. The area from 6030 to 642F is then available for machine language. BASIC will not interfere with it.

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

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

Correction of the Printer Bug

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

printer to the left margin. Thus, when printing is resumed, it may begin part way through the line. The left margin may be reset by execution of POKE 66F4E,0. (You might want to assign this POKE to a RESERVE key.)

Katakana Mode

The PC-1350 ROM contains codes for generating Katakana characters. Katakana mode is set by execution of POKE 66F16, 680. It is cancelled by switching the computer off then on. Or, by executing POKE 66F16,0. In Katakana mode, keying SML will display the Katakana annunciator rather than SML. The alphabet keys will then generate codes for Katakana characters. The SHIFT key prefix as well as the keys for plus, minus, equals, comma, colon and semicolon, allow keying in 63 Katakana characters. Note that the codes for each of these characters is a two-byte code, with 6FE as the first byte. Hence, string variables will hold fewer than the normal number of characters of this kind. The Katakana alphabet is also obtainable using CHR$ 161 to 223. Also, CHR$ 241 to 244 will return some characters that apparently are not Katakana. Shift * will produce the symbol for Yen. All of these characters are printable with the CE-126P.

An Apostrophe Key

If you would like a key for the apostrophe, which may be displayed or printed, here is how it can be done. First, clear RESERVE memory by executing NEW in the RESERVE mode. Then POKE 67F7F,241,39. Now the key sequence SHIFT/SPC will key in the apostrophe character. (Additional RESERVE keys may be assigned in the normal way.)

How to TAB with the Printer

The PC-1350 BASIC lacks TAB. And, if you have tried it, you will have found that the CURSOR statement applies to the display, but not to the printer. Consider the printed line as consisting of 24 columns, numbered from 0 to 23. To begin the next following LPRINT in column n, execute POKE 66F4E,n. However, some precautions are in order. Do not use a value of n greater than 23 or you will have unexpected garbage printed. Also, do not set a column that precedes characters already specified for printing -- or some of the data will not get printed.

You might want to try this example: 10 LPRINT 1;:POKE 66F4E,7:LPRINT 2;:POKE 66F4E,15:LPRINT 3

Specification of Format

It is possible to specify PRINT (and LPRINT) formats with POKE rather than with USING. The advantage is that calculated values may be used to determine the desired format. The number of spaces to precede the decimal point, including a space for the sign, should be POKEd into 66F3A. The number of spaces to include and follow after the decimal pointer, should be POKEd into 66F3B. For character string lengths, POKE the number of spaces into 66F3C.

Keeping the Display Cursor in the Right Line

To set the display cursor to column n (with n ranging from 0 to 23) within the same line of the display that has already begun, execute CURSOR n, PEEK 67880. (The current line of the display being used by PRINT is stored in 67880.)

Some Fancy INPUT Prompts

When an INPUT statement follows a PRINTed prompting message, the INPUT will begin on the next line of the display. It is possible to accept input in the *same* display line as the PRINTed prompt by executing POKE 66F17,4. This enables the flashing cursor at the current location (rather than on the next line). A CURSOR statement is required after INPUT has been made, to advance to the next line. The following example should help clarify this procedure:

```
10 DIM A(10)
20 WAIT 0
30 FOR I=1 to 10: PRINT "A(";STR$ I;")=";
   POKE 66F17,4: INPUT A(I)
40 CURSOR: NEXT I
```

Clearing Defined Variables and Arrays

You can clear the defined variables and arrays *without* clearing the fixed variables by executing POKE 66F07,48,108.

Finally, if you forget your password, POKE 66F14,0 will reset the password protection feature.

The above information was supplied by: *Norlin Rober, 407 North 1st Avenue, Marshalltown, IA 50158.*

FROM THE WATCH POCKET

Is Sharp going to sell the PC-2500 in the United States? This is a portable "notebook" size version of the PC-1350. It has a touch-typing keyboard and a built-in printer/plotter along the lines of the CE-150 unit. The CPU is the same as that used in the 1350. It even uses the same memory expansion modules (CE-201 -- 8K and CE-202 -- 16K) as the PC-1350. There is a lot of potential interest in this unit, primarily because it might serve as a tool for PC software developers. Seems the machine is being delivered in Europe and other countries now, but is not yet available in the United States.

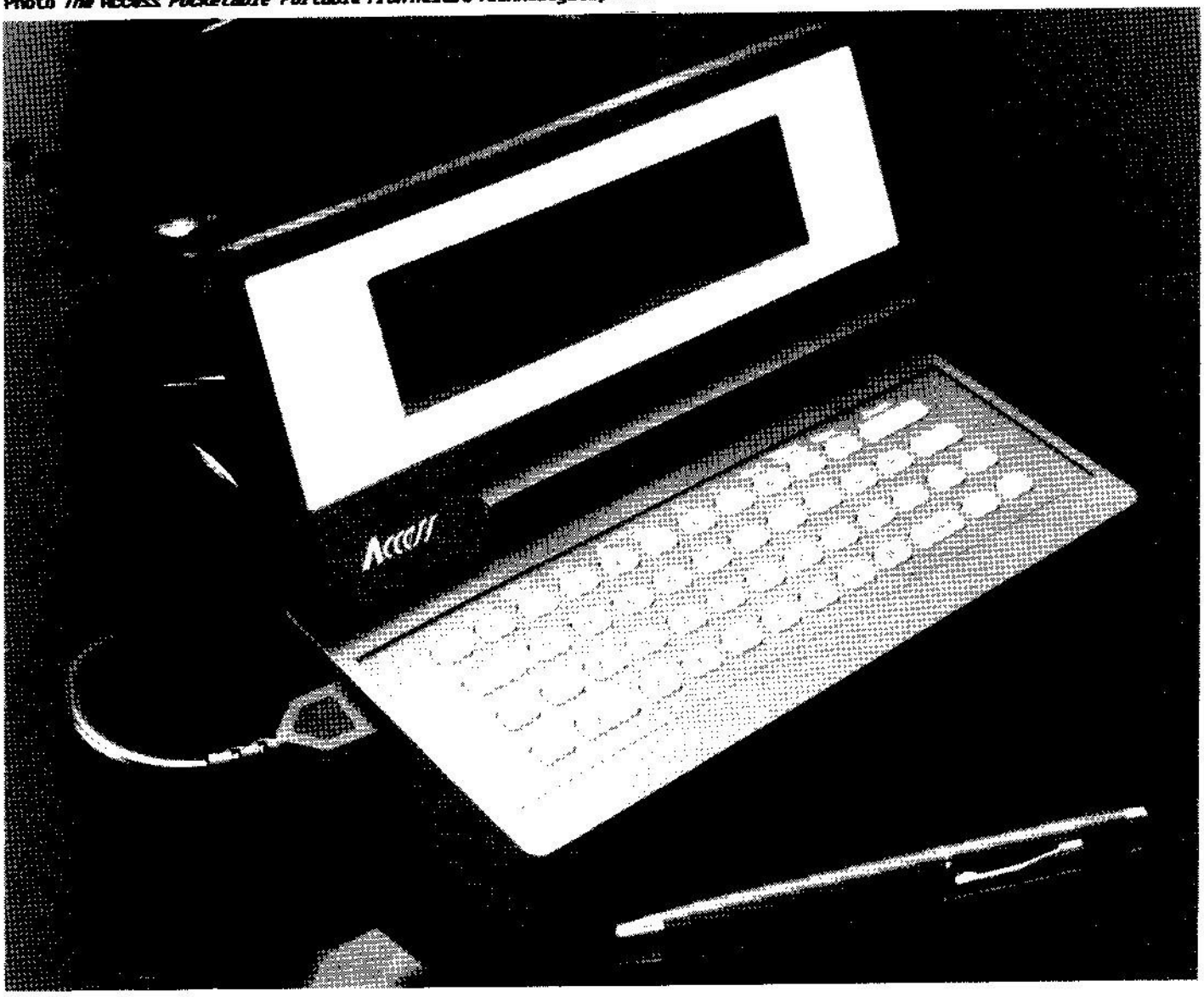
And, where is the technical manual for the PC-1350? We heard rumors months ago that Sharp would have a nice manual for the techies available soon.

If you are looking for PC-1500 supplies -- such as CE-161 (16K memory modules), printer/plotter paper & pens, etc., try contacting: **Atlantic N.E. Marketing,** *P.O. Box 921, Marblehead, MA 01945.* The phone number is: (617) 745-7707.

POCKET COMPUTER NEWSLETTER

Issue 41 - July/August

Photo *The Access Pocketable-Portable from Melard Technologies, Inc.*



Another *Personal Information* (TM) product.

NEW POCKET-SIZED COMPUTER DEBUTES

Melard Technologies, an Elmsford, New York, company has introduced a new pocket-sized portable computer and remote terminal. The new machine, which the firm claims is unlike any other portable personal computer currently available, is small (8-1/4 by 3-3/4 by 1-7/8 inches) and said to be more powerful than hand-held terminals. The company states that elegant software design makes it easy for the computer novice to use. The company has named the new machine *Access* and is marketing it for customized applications in selected vertical markets through value-added resellers and systems integrators.

Small and lightweight (at 24 ounces), the computer packs a lot of power. It is capable of handling up to 120K of memory including as much as 64K of removable, cartridge-based memory that may be used for program or data storage. The machine can also serve as an interactive terminal that connects to remote computers. It can handle electronic mail by tapping into networks such as EasyLink, MCI Mail, and ITT Dialcom. It can access database services such as Dialog, BRS or Medlars. And, it connects to informations services such as Dow-Jones, CompuServe, The Source, and hundreds of others. Connection is said to be quickly and easily accomplished using just a few keystrokes. The user doesn't have to worry about number of bits, baud rates, and other such barriers to immediate communication.

When operating "on-line" *Access* can use its micro-computer to selectively capture, edit, and store transmitted data (up to the equivalent of twenty-four 8-1/2 by 11 pages at a time). Communication with remote computers or databases is possible anywhere there is a phone. This is accomplished using the unique *Access* acoustic-cup modem. This device performs all the functions of a bulky modem, yet it can be slipped into a pocket.

Once the unit is turned on the user is presented with the main application-oriented program, communications options and a host of personal productivity programs. The unit's screen displays a selection of: personal scheduler, address book/phone directory, calculator, and time alarm. Software also provides for word processing and information management.

The basic *Access* unit can be customized as to function, electronics, and appearance. Flexibility in configuring the amount of built-in ROM and RAM as well as the plug-in data/program cartridges, enable the unit to be tailored to specific application requirements. Thus, the portable unit can become an integral part of a value-added reseller's basic system. With appropriate software, for example, the machine can become one of the smallest, easiest to use computer terminals for home banking and financial services. By utilizing custom software/firmware combinations, it can also serve medical, real estate, insurance, and a variety of other fields as a portable database, used to transport records from office to the field and back. Records can be updated as desired. Data can be fed directly into the home office computer for storage. Alternately, the data may be printed out locally using a printer connected through the serial port on the *Access* machine.

Access carries a large display screen for its size: 40 characters by 8 lines. This makes it easy to read data. The screen can serve as a window on a full 80-character by 24-line screen that is easily scrolled using cursor keys. The screen can be adjusted for optimum viewing through a combination of software contrast control and a continuously adjustable display angle control. The screen can also serve as a bit-mapped display.

The unit offers a full ASCII keyboard plus programmable function keys. Silicon rubber keys provide tactile feedback and silent, soft-touch operation.

The computer/terminal is battery-powered to provide full portability. Expected battery life is 12 hours of operating time. The unit has a low battery indicator as well as automatic shutoff for the preservation of data and programs. An AC adaptor is also provided.

The *Access* computer is currently available in various configurations starting at a price of $599.00 in the United States. For more information on the machine, contact: *Melard Technologies, Inc., 5 Westchester Plaza, Elmsford, NY 10523*. The telphone number is: (914) 592-3044.

FOR PC-1250/51/60/61 USERS

MORE MACHINE LANGUAGE INSTRUCTIONS

Rick Wenger, 6221 - 18th Avenue, Kenosha, WI 53140, wants to tie up some loose ends and revamp some of the terminology from his earlier machine code article. (This refers to the information provided by him in issues 38 and 39 of *PCW*).

1. Change the mnemonic for opcode 4C from "INA D" to "INA KBD" and delete all references to the 16-bit "D" register.

2. Rename CPA registers 5C, 5D, 5E and 5F as output ports C, D, E and F respectively.

3. Opcode DF (because of item 2 above) thus will be renamed as "OUTF" instead of the original "OUT1".

4. The purpose of the XX byte in the 4-byte 7A instruction has been determined. Change the mnemonic to read "PUSH -cc- nnnn. This will be explained further in the discussion that follows.

These minor revisions will improve the system while providing the flexibility that may be required by new Sharp models. You may now add the following nine machine code directives to the previously published instruction set.

69 (JCA): Jump Conditional on Accumulator. This is a variable length instruction somewhat akin to the "ON ... GOTO" instruction in BASIC. It is always associated in the ROM code of the PC-1250 with the "7A" (PUSH -cc- nnnn) instruction. The purpose of the -cc- byte is to set an internal counter which determines the length of the subsequent JCA instruction. Here is an example taken from the PC-1250 ROM that will help clarify the concept:

Address	Disassembled Code
1C4F	PUSH -04- 1C62
1C53	JCA 37 0B74
1C57	38 0B7D
1C5A	39 0B86
1C5D	85 10B0
1C60	10A5
1C62	RTS

Thus, if the content of the accumulator is 37, program flow branches to address 0B74. If the accumulator is 38, the branch is to 0B7D. If the accumulator is 39, the branch is to 0B86. When the accumulator contains 85, the program branches to 10B0. And, if the accumulator does not contain any of those values then the program goes to 10A5. In all cases, the program will return to 1C62 (the address that was pushed onto the stack before the JCA instruction was invoked) when an RTS is encountered. In this instance, 1C62 itself contains an RTS instruction, but it could be any other type of directive.

Experiments have shown that these two types of instructions (PUSH and JCA) can be used in a less restrictive way than what appears in the ROM coding:

1. PUSH -cc- nnn is a viable instruction on its own.

2. The value pushed onto the stack does not have to be the address located right after the JCA instruction.

3. The PUSH instruction and the JCA instruction can be separated by machine language code provided not interative instructions (those with a 0 or a 1 in the mnemonic) are invoked.

4. The length of a JCA instruction can be determined by the formula: (cc + 1) * 3.

The rest of the instructions to be presented herein relate to I/O operations.

5D (OUTC)
0D (OUTD)
5F (OUTE)

It is interesting to note that ports C and D on the Sharp PC-1250 relate to the keyboard strobe. Port E relates to printer operations. Budding CPU sleuths may wish to work out the details on their own. PC-1350 owners are advised that it appears to sense the keyboard in a manner different from the PC-1250.

CC (INA PORT): This directive senses the status of various devices through lines opened by OUTD. (It is possible that such lines are opened in a different manner on the PC-1350). For example, if 08 is output through D, the instruction INA PORT will: set bit 0 in the accumulator if the slide switch is set to RSV, set bit 1 if the slide switch is at PRO, set bit 2 if the slide switch is OFF, and clear bits 0, 1 and 2 if the slide switch is set to the RUN mode.

4F (0 SIGNAL)
6F (0 NO SIGNAL): These instructions are dedicated to reading signals from the cassette interface. The incoming signal affects the timing of these instructions as well as effecting a change in the P pointer register.

First of all, in order for these instructions to work properly, a line must be opened to the cassette interface. On the PC-1250 this can be accomplished by sending the byte 70 out through port F. Opcode 4F (as well as 6F) will then put out some sort of "resistance" to the incoming signal, the purpose apparently being to block out the normal background noise that exists on any cassette tape. An increment is then done on the P register. When using 4F, this increment of the P register will be repeated as long as the incoming electrical signal *is sufficient* to overcome the resistance. When using 6F, this increment of P will be repeated as long as the signal is *not sufficient* to overcome the resistance.

Both 4F and 6F use the value in the C0 counter as an upper limit to the number of increments that will be attempted. Again, an example from coding found in the PC-1250 ROM as part of the CLOAD routine should help clarify the use of these instructions:

Address	Disassembled Code	Comments
7392	IN 08	If BRK key is
7394	FZC 2E <73C3>	pressed abort CLOAD
7396	SLP 00	
7397	0 NO SIGNAL	If change in
7398	0 SIGNAL	P is less
7399	LDA P	than 9,
739A	CPA 09	the restart
739C	JCS 7392	waiting loop
739F	REFR 02	
73A1	SLP 00	
73A2	0 NO SIGNAL	If change in
73A3	0 SIGNAL	P is less
73A4	LDA P	than 9,
73A5	CPA 09	then restart
73A7	JCS 7392	waiting loop
73AA	REFR 02	
73AC	SLP 00	
73AD	0 NO SIGNAL	
73AE	0 SIGNAL	
73AF	LDA P	got it??
73B0	CPA 09	
73B2	JCS 7392	

In the PC-1250, information is stored on tape a nibble (in 4-bit blocks) at a time. The purpose of the above stretch of code is to wait for a low-pitched beep that signals the start of a new nibble. Each 0 NO SIGNAL, 0 SIGNAL sequence measures a substantial portion of a single wave of recorded sound. The wave will be longer than the wave of a regular high-pitched beep. The above code will only allow execution to proceed when three waves in a row register a change in P of at least nine. Since 0 NO SIGNAL and 0 SIGNAL are sensitive to the timing of a wave, the above ROM code serves to accurately kick-off the timing for each nibble being CLOADed.

0 SIGNAL (code 4F) also has an influence on the Z flag. It clears the Z flag only if the "ready to listen" signal has been sent through OUTF.

The last two instructions I will present are somewhat speculative. The hardest instructions to test are those that apparently do nothing. The following interpretations are consistent with all findings of these opcodes in ROM as well as every test I could come up with.

CE (LNOP): Long "no operation" instruction.
4D (SNOP): Short "no operation"instruction.

The PC-1250 will perform approximately 63,800 CE instructions per second. Three 4D instructions execute in the same amount of time as two CE directives.

DISASSEMBLER FOR PC-1250/1251

Would you believe a disassembler for the *Wenger Mnemonics* that resides in just 1.1K? Believe it! Rick Wenger himself came up with this gem. The version shown in the accompanying listings is for the Sharp PC-1251.

To adapt it to the PC-1250, just POKE 6C490,6C0 after you have loaded the program into memory! (Remember, when this program is installed in a 1250 you will only have 316 bytes of memory left over.)

The memory map of this program (for a 1251) is as follows:

Address	Comments
B831-B8AB	BASIC portion of disassembler.
B8AC-C1E8	Available to user.
C1E9-C5CF	Machine language part of disassembler.

Also, remember that BASIC variables A, B and C are used by the disassembler. Additionally, since no attempt has been made to protect the machine language portion of the program, you should not attempt to use dimensioned variables when this program is in memory.

Once the program has been loaded (in two parts, a short

Program *BASIC Portion of the PC-1251 Disassembler.*

```
1:"D" AREAD A$:B=1
2:CALL &C591
3:CALL &C488: PRINT "1
  234: 123456789012345
  6"
4:IF PEEK &C058=&13
  BEEP 4: INPUT "MONIT
  OR? ";C$: IF C$="Y"
  LET B=0
5:GOTO 3
6:"M" AREAD A$:B=0:
  GOTO 2
```

BASIC portion and the rest in machine language), the program is a cinch to use. Just enter the starting address for disassembly and press DEF/D (the DEFine key and the letter D key). This will give you a mnemonic disassembly. You can also get a memory dump in machine code by entering an address followed by DEF/M.

Need hardcopy? No problem. Just connect your printer and issue the directive PRINT = LPRINT. If you want to go back to using the display just hit the SHIFT and CL keys.

The program does have a few idiosynchracies. The power sign (^ for exponentiation) is used instead of * since the 1250 does not have the latter character in its vocabulary. And, PP is used instead of P+ for technical reasons. Also, the JCA directive (see preceding article in this issue) is not included in the disassembler. (You would need to do a backwards trace just to discover the length of the instruction.) Hence, the disassembler program stops and asks whether you want to go into monitor mode (memory dump). Any response to this query other than "Y" results in the disassembler continuing operation.

Radio Shack PC-3A users should find the program useful as should PC-A owners (who will need to perform the POKE &C490,&C0 procedure after loading the program as indicated above for the PC-1250 version).

Now you can explore the ROM in your PC-1250/51 PC-3/A at your leisure, enjoying the ease of *Wenger Mnemonics*. Enjoy!

Program *Machine Language Portion of the PC-1251 Disassembler.*

C455 14 1A 198B

```
C1E9: 80 31 09 08
C1ED: 82 31 09 08
C1F1: 83 31 09 86
C1F5: 31 09 87 0E
C1F9: 13 08 9E 09
C1FD: 0A 9E 0E 13
C201: 08 9D 09 0A
C205: 9D 11 09 02
C209: 75 00 55 A1
C20D: 0A 1D 02 75
C211: 00 55 A1 11
C215: 09 03 75 00
C219: 55 A1 0A 1D
C21D: 03 75 00 55
C221: A1 06 09 0A
C225: 08 02 75 00
C229: 86 18 09 0A
C22D: 08 02 75 00
C231: 86 06 09 0A
C235: 08 02 75 00
C239: 55 A1 18 09
C23D: 0A 08 02 75
C241: 00 55 A1 51
C245: 09 9C 31 09
C249: 1C 91 31 09
C24D: 95 31 09 15
C251: 81 06 09 09
C255: 55 35 00 06
C259: 87 18 1A 07
C25D: 55 35 00 06
C261: 87 80 80 11
C265: 09 02 75 00
C269: FC 0A 1D 02
C26D: 75 00 FC 11
C271: 09 03 75 00
C275: FC 0A 1D 03
C279: 75 00 FC 17
C27D: 17 02 F5 17
C281: 11 02 F5 11
C285: 09 02 75 00
C289: 86 11 09 02
C28D: 7C 00 86 11
C291: 09 06 00 95
C295: 11 09 06 00
C299: 15 81 11 09
C29D: 06 00 18 95
C2A1: 80 11 09 0E
C2A5: 06 00 FE 11
C2A9: 09 09 06 00
C2AD: FE 11 09 0E

C2B1: 7D 00 86 11
C2B5: 09 09 7D 00
C2B9: 86 2B 1F 88
C2BD: 37 1F 88 2B
C2C1: 08 88 37 08
C2C5: 88 2B 1C 89
C2C9: 37 0A 9B 80
C2CD: 37 09 97 11
C2D1: 09 15 00 86
C2D5: 11 09 15 01
C2D9: 00 86 11 09
C2DD: 18 15 00 86
C2E1: 80 15 1A 18
C2E5: 0D 86 11 09
C2E9: 02 75 00 0D
C2ED: 97 80 17 19
C2F1: 98 2B 1F 98
C2F5: 37 1F 98 2B
C2F9: 08 98 37 08
C2FD: 98 80 80 80
C301: 80 0E 13 08
C305: 08 82 09 0A
C309: 08 82 0E 13
C30D: 08 86 09 0A
C311: 86 06 09 09
C315: 75 00 86 18
C319: 1A 07 75 00
C31D: 86 06 13 09
C321: 75 00 86 14
C325: 17 75 00 86
C329: 0E 13 08 42
C32D: A4 09 0A 42
C331: A4 0E 13 08
C335: 42 A6 09 0A
C339: 42 A6 0E 13
C33D: 06 00 10 07
C341: 89 18 13 14
C345: 95 37 0A 0B
C349: 97 02 18 0E
C34D: 0C 13 06 91
C351: 0E 13 08 95
C355: 09 0A 95 11
C359: 09 7C 00 86
C35D: 11 09 7C 00
C361: F5 80 11 09
C365: 75 00 FC 80
C369: 11 09 06 00
C36D: FC 17 09 86
C371: 11 09 06 00
C375: F5 17 11 86

C379: 15 14 15 86
C37D: 80 14 1A 19
C381: 88 80 14 1A
C385: 19 8A 26 13
C389: 09 F5 34 17
C38D: F5 27 0E 19
C391: F5 28 15 F5
C395: 26 13 09 86
C399: 34 17 86 27
C39D: 0E 19 86 28
C3A1: 15 86 80 80
C3A5: 80 2E 93 80
C3A9: 80 80 02 13
C3AD: 14 00 18 0E
C3B1: 0C 13 06 91
C3B5: 26 09 09 F5
C3B9: 38 1A 87 F5
C3BD: 80 80 26 09
C3C1: 09 86 38 1A
C3C5: 07 86 80 80
C3C9: 4F 18 97 4F
C3CD: 12 95 75 1A
C3D1: 18 8D 80 4F
C3D5: 1F 88 4F 08
C3D9: 88 4F 1F 98
C3DD: 4F 08 98 0E
C3E1: 13 08 08 83
C3E5: 09 0A 08 83
C3E9: 0E 13 08 87
C3ED: 09 0A 87 06
C3F1: 09 08 75 00
C3F5: 86 18 07 08
C3F9: 75 00 86 07
C3FD: 0E 19 75 00
C401: 86 08 15 75
C405: 00 86 0E 13
C409: 08 42 A5 09
C40D: 0A 42 A5 0E
C411: 13 08 42 A7
C415: 09 0A 42 A7
C419: 0E 13 06 00
C41D: 15 14 17 99
C421: 80 11 13 14
C425: 95 80 18 0A
C429: 19 88 08 11
C42D: 17 83 17 17
C431: 86 80 26 13
C435: 09 FC 34 17
C439: FC 27 0E 19
C43D: FC 80 08 11

C441: 17 58 B5 80
C445: 0A 1D 06 00
C449: 87 0A 1D 06
C44D: 00 F5 80 14
C451: 1A 19 89 80
C459: 02 63 26 26
C45D: 02 11 26 88
C461: 71 E0 03 01
C465: 2C E4 02 63
C469: 26 02 5C 26
C46D: 02 60 26 02
C471: 11 26 90 71
C475: 80 78 C5 BC
C479: 04 2C D8 02
C47D: 13 26 88 78
C481: C5 BC 02 13
C485: 26 2C A8 10
C489: C5 CE 84 1A
C48D: 02 51 03 B8
C491: 13 02 0A 85
C495: 78 C5 B8 06
C499: 06 4D 82 13
C49D: 04 0A 00 03
C4A1: 88 35 85 63
C4A5: E0 3A 05 05
C4A9: 04 88 18 10
C4AD: C6 92 D6 10
C4B1: 28 16 88 03
C4B5: 04 78 C5 BC
C4B9: 02 11 26 04
C4BD: 50 C3 29 0A
C4C1: 10 C5 CE 84
C4C5: 1B 2C C1 83
C4C9: 59 90 DB 63
C4CD: E0 2B 76 63
C4D1: C0 3A 05 71
C4D5: 40 2C 05 63
C4D9: 80 2B 74 02
C4DD: E8 03 C1 84
C4E1: 13 02 0A 90
C4E5: 59 04 D6 80
C4E9: 39 04 43 2B
C4ED: 07 A0 00 0A
C4F1: 18 A1 62 80
C4F5: 29 7A 2C 08
C4F9: 62 40 38 04
C4FD: 02 30 26 59
C501: 64 1F 74 11
C505: 67 11 38 15
C509: 74 27 67 39
C50D: 38 0F 74 06

C511: 67 42 3A 09
C515: 74 06 67 4A
C519: 3A 03 74 07
C51D: 26 20 67 21
C521: 38 08 62 20
C525: 38 04 02 31
C529: 26 62 80 50
C52D: 39 35 02 11
C531: 26 10 C5 CE
C535: 84 1A A1 59
C539: D2 58 64 03
C53D: 67 03 89 DA
C541: 28 04 02 36
C545: 26 04 C3 3A
C549: 0A 78 C5 BC
C54D: 50 DA 67 02
C551: 2D 12 10 C5
C555: CE 84 1B 88
C559: 63 2F 38 11
C55D: 63 28 3A 28
C561: 63 2E 3A 09
C565: 63 38 3A 20
C569: 63 3C 2A 1C
C56D: 02 11 26 02
C571: 33 26 04 89
C575: 59 03 00 88
C579: 62 01 84 28
C57D: 04 14 2C 02
C581: 15 78 C5 B8
C585: 02 32 26 02
C589: 11 86 26 63
C58D: 67 3B 04 37
C591: 10 C6 9B 90
C595: 1A 11 99 1A
C599: 90 63 51 3A
C59D: 03 71 07 71
C5A1: 40 50 20 67
C5A5: 14 29 0D 93
C5A9: 59 92 DB 58
C5AD: 44 90 59 58
C5B1: 91 44 10 C5
C5B5: CE 1B 37 78
C5B9: C5 BC 51 59
C5BD: 58 78 C5 C2
C5C1: 59 64 0F 74
C5C5: 40 67 4A 3A
C5C9: 03 74 07 26
C5CD: 37 C0 C5 00
C5D1: 00 00 00 00
C5D5: 00 00 00 00
C5D9: 00 00 00 00
```

FOR HEWLETT-PACKARD HP-71B USERS

CALENDAR PROGRAM

The HP-71B has powerful time (TIME$) and date (DATE$) functions. These may be used to perform a number of useful tasks, such as keeping track of appointments (see *PCN* Issue 39), determining the number of days between dates, and so forth. However, it is often desirable to be able to quickly review the structure of a month, whether past, present or future, to determine what day of the week a particular day falls upon. This can be particularly useful to check dates of weekends, paydays(!), holidays, and so forth.

The short BASIC program provided in the accompanying listing may be used to produce a monthly calendar for any period between March, 1700, (for checking the past) on up to February, 2200, (for checking the future). The version shown, crafted by tailoring a program originally submitted by *Emerich Auerswachter*, takes just 816 bytes of memory space in the HP-71B.

Once loaded, execute the package and respond to the two prompts for the "number" of the month (where 1 = January through 12 = December) and the year. Soon the layout of the desired month will appear on the display. Use the END LINE key to view each week of the month after the calendar heading (S M T W T F S) appears.

Naturally, if you want to obtain a hardcopy of a monthly calendar, just connect to your printer and issue the DISPLAY IS PRINTER command.

It is a mighty handy program to have tucked away in your HP-71B.

Example *Output from HP-71B Calendar Program.*

```
>
MONTH (1-12): 7
YEAR: 1985
 S  M  T  W  T  F  S
 -  1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31
>
>
MONTH (1-12): 8
YEAR: 1985
 S  M  T  W  T  F  S
 -  -  -  -  1  2  3
 4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30 31
>
```

Program *Calendar for the HP-71B.*

```
5 DELAY 8,.1 @ WIDTH 96 @ DESTROY ALL @ DIM A$(68)[2]
10 DATA " 1"," 2"," 3"," 4"," 5"," 6"," 7"," 8"
15 DATA " 9","10","11","12","13","14","15","16"
20 DATA "17","18","19","20","21","22","23","24"
25 DATA "25","26","27","28","29","30","31"
30 FOR N=1 TO 31 @ READ A$(N) @ NEXT N
35 INPUT "MONTH (1-12): ",'1';A @ INPUT "YEAR: ",'1985';B @ E=621048 @ C=365.25
40 IF A<=2 THEN D=INT(30.6*(A+13))+INT(C*(B-1))-E @ GOTO 55
45 D=INT(30.6*(A+1))+INT(B*C-E)
50 IF D>146097 THEN D=D-1
55 IF D<=73047 THEN D=D+1 ELSE IF D<=36522 THEN D=D+1
60 D=D/7 @ D=INT(7*(D-INT(D))+1.49)
65 IF A=4 OR A=6 OR A=9 OR A=11 THEN C=30.
70 IF A=1 OR A=3 OR A=5 OR A=7 OR A=8 OR A=10 OR A=12 THEN C=31
75 IF A=2 THEN C=28 ELSE IF (B/4=INT(B/4))+(B/100<>INT(B/100))=2 THEN C=29
80 IF A=2 AND B=2000 THEN C=29
85 FOR E=1 TO D-1 @ A$(E+31)=" -" @ NEXT E
90 FOR E=1 TO C @ A$(D+E+30)=A$(E) @ NEXT E
95 DISP " S  M  T  W  T  F  S"
100 FOR N=1 TO 7 @ DISP A$(N+31);" "; @ NEXT N @ DISP
105 FOR N=1 TO 7 @ DISP A$(N+38);" "; @ NEXT N @ DISP
110 FOR N=1 TO 7 @ DISP A$(N+45);" "; @ NEXT N @ DISP
115 FOR N=1 TO 7 @ DISP A$(N+52);" "; @ NEXT N @ DISP
120 FOR N=1 TO 6 @ DISP A$(N+59);" "; @ NEXT N @ DISP A$(N+59); @ DISP A$(67);"
125 END
```

FOR PC-1500 & PC-2 USERS

SOFTWARE CHESS CLOCK

The serious chess player needs a chess clock to time moves in tournament play, but an electronic clock can cost $60 or more. If you own a Radio Shack PC-2 or Sharp PC-1500 pocket computer, however, you already own a chess clock thanks to the built-in time function. All you need to do is to copy the accompanying program into your machine!

This chess clock has many extras: (1.) electronic accuracy, (2.) automatic move counter, (3.) graphics to show whose move it is, (4.) beep to signal key presses and time defaults (can be turned off) and (5.) clock can be halted at any point (automatic power-off saves batteries).

Operation

Operation is like traditional chess clocks, but with a modern twist. Once the program is loaded (save a copy on tape, of course) proceed as follows.

1. Position the computer so that it rests between the players lengthwise along the side of the board with the name plate on the White player's side.
2. When ready, type RUN and press the ENTER button.
3. After your move, press any key to start opponent's timer.

Exceptions:

a. Press BRK key to stop the program.

b. Press "P" (pause) to stop both clocks. To restart press ENTER button. If the delay is long, the display will turn off. Do not be alarmed. When ready, press the ON button and then the ENTER key to take up where you left off.

Reading the Display

The diagram below illustrates the display. An arrow serves as an indicator that points to the player whose turn it is to move.

	H.MMSS Move X	*	(White)
*	H.MMSS Move... X		(Black)

H = hours
M = minutes
S = seconds
X = move number
* = flag indicating < 5 minutes to a time default

If you go over the time limit, the computer beeps and indicates the winner on the display.

Program Explanation

The following is a line-by-line explanation of the program listing. Use it if you want to modify the program or learn exactly how it operates.

10 REMark that identifies program.

20 DIMension statement that defines size of characters used to warn of impending time default (under 5 minutes). WAIT 0 directive keeps the screen updating.

30 Sets variables to initial values.

40 Hold graphics characters for left arrow.

50 Hold graphics characters for right arrow.

60 H stands for header. This includes month and day which is subtracted out in this application.

70 T stands for base time. It is converted to decimal and subtracted from current time to give elapsed time.

80 W is White's elapsed time. To get this you take the current time, subtract header, convert to decimal, subtract start time (T), subtract time used by Black (B) and subtract the length of any pauses (P).

906100 These are the time controls. You may need to change statements 90, 100, 105, 106 and 170, 180, 185, 186 if different time controls are used. For example, if time control is 40 moves in 90 minutes and 30 moves per hour thereafter, then these lines would become:

```
90  IF W>1.5AND M<=40 GOTO 400
100 IF W>2.5AND M<=70 GOTO 400
105 IF W>1.4167AND M<=40 THEN LET X$="*"
106 IF W>2.4167AND M<=70 THEN LET X$="*"
170 IF B>1.5AND M<=40 GOTO 500
180 IF B>2.5AND M<=70 GOTO 500
185 IF B>1.4167AND M<=40 THEN LET Y$="*"
186 IF B>2.4167AND M<=70 THEN LET Y$="*"
```

1056106 Sets warning flag if down to five minutes or less.

110-125 Formats and prints White's display.

130 Checks to see if you wish to stop clocks.

140 If any key is pressed then loop is ended for White.

150 This statement is used to prevent keyboard bounce. Otherwise, pressing a button on the computer might cause the display to switch several times before you released the key. Not only would this be frustrating, it would also destroy the move count.

155 Beep used to give feedback that button was pressed.

160-230 Same as 80-150 except for Black side.

240 Increase move count. Clear warning flags and beep.

250 Continue looping (calculating elapsed time) until break key is pressed, program is paused, or a time fault occurs.

300 REMark.

310 Q = base time used to calculate length of a pause. WAIT causes display to freeze until ENTER button is pressed.

320 Display White side and Black side elapsed times.

330 Return to automatic update mode.

340 Compute time of accumulated pauses.

350 Prevents keyboard bounce.

360 Returns to timing loop.

400-450 Sound and visual cue that White side has used all its time.

500-550 Sound and visual cue that Black side has used all its time.

Note: If you want to silence the beep sound, enter the directive BEEP OFF. Use BEEP ON to restore audio capability.

Additional Notes

Here are some general comments about the program.

1. No extra memory expansion module is required. The program will reside in a PC having minimum memory.
2. "a" signs in the listing indicate required spaces. Do not type the a sign, use the "space"button each time this sign appears.
3. Warning! Do not use the clock past midnight. A false time fault will occur.
4. If tournament directors are concerned with the possibility of cheating, they can compare an entrant's program to a master listing.
5. A shortcoming of this clock is the liquid-crystal display which is hard to read in poor lighting. Results are best when the playing area is well illuminated.
6. Execute a STATUS 1 to check the accuracy of your loading. If the answer is not equal to 1053, check your work.

The Author

 It is reprinted here by permission of the author. Hope all you chess fans will find it useful.

Program *Chess Clock for the PC-1500 & PC-2.*

```
 10 REM Chess Clock, 27 Jan. 84 JRG
 20 DIM X$(1)*1,Y$(1)*1:WAIT 0
 30 CLEAR:M=1:Y$="@":REM Initialize
 40 L$="081C3E7F1C1C1C1C"
 50 R$="1C1C1C1C7F3E1C08"
 60 H=INT(TIME/100)*100
 70 T=DEG(TIME-H)
 80 W=DEG(TIME-H)-T-B-P
 90 IF W>2AND M<=40GOTO 400
100 IF W>3AND M<=60GOTO 400
105 IF W>1.9167AND M<=40THEN LET X$="*"
106 IF W>2.9167AND M<=60THEN LET X$="*"
110 GPRINT L$;
120 PRINT "@@@";USING "##.####";DMS W;"@Move@@";USING "###";M;
125 CURSOR 25:PRINT X$
130 IF INKEY$ ="P"THEN GOSUB 300
140 IF INKEY$ =""THEN GOTO 80
150 IF INKEY$<> ""THEN GOTO 150
155 BEEP 1,250,4
160 B=DEG(TIME-H)-T-W-P
170 IF B>2AND M<=40GOTO 500
180 IF B>3AND M<=60GOTO 500
185 IF B>1.9167AND M<=40THEN LET Y$="*"
186 IF B>2.9167AND M<=60THEN LET Y$="*"
187 PRINT Y$;
190 PRINT "@@@";USING "##.####";DMS B;"@Move...";USING "###";M;"@@";
200 GPRINT R$
210 IF INKEY$="P"THEN GOSUB 300
220 IF INKEY$=""THEN GOTO 160
230 IF INKEY$<>""THEN GOTO 230
240 M=M+1:X$="@":Y$="@":BEEP 1,250,4
250 GOTO 80
300 REM Pause, stop timing
310 Q=DEG(TIME-H):WAIT
320 PRINT "@White";USING "##.####";DMS W;"@Black";USING "##.####";DMS B
330 WAIT 0
340 P=DEG(TIME-H)-Q+P
350 IF INKEY$<>""THEN GOTO 350
360 RETURN
400 FOR I=0TO 155
410 BEEP 1,I,4
420 GCURSOR I:GPRINT RND 127
430 NEXT I:WAIT
440 PRINT "*Time default, Black wins!"
450 END
500 FOR I=0TO 155
510 BEEP 1,I,4
520 GCURSOR I:GPRINT RND 127
530 NEXT I:WAIT
540 PRINT "*Time default, White wins!"
550 END
```

FOR PC-1350 USERS

INSIDE THE PC-1350

Merlin Raber **provides more information on the operation of the PC-1350 pocket computer:**

BASIC executes slightly faster (by about 4%) than in the 1260 series, but, except for loops, the 1350 is still no match for the PC-1500. The four-line display, as one might expect, is a bit sluggish in operation compared to one-liners.

A 1350, with 16K RAM expansion, can be expected to have a slightly larger program capacity than an expanded 1500A, since the 1350's tokens are only one byte each.

The 1350 lacks BEEP OFF, so you'll have to put up with the chirping while performing cassette operations. It also lacks display annunciators for "BUSY", trigonometric mode, and PRINT ON. PC-1500 devotees might miss TIME, sixteen-byte fixed string variables, and the RESERVE Mode "menu", but will appreciate being able to insert and delete characters without having to constantly SHIFT.

The PC-1350 contains PEEK, POKE, CALL, CSAVE M, and CLOAD M, although their operation is not discussed in the instruction manual.

I have found a few rather nice features of the 1350 that the manual neglects to mention. If SHIFT is keyed prior to use of the cursor left/right keys, the cursor will instantly move to the beginning or end of the displayed line. Commas separating items in a GPRINT statement, as with the PC-1500, will put blank columns between dot columns. And the "&" prefix may be used with hexadecimal numbers as large as &2540BE3FF!

I do have some minor gripes concerning the printer. If BREAK is keyed while the CE-126P is busy printing, the next print will not necessarily begin at the left margin. Lack of TAB complicates formatting with the printer. And those thermal paper rolls are so terribly short--not to mention expensive!

STORAGE OF BASIC PROGRAM LINES

Each line of BASIC begins with a two-byte line number, in hexadecimal form. This is followed by a link byte, which specifies the number of bytes in the remainder of the line, including the terminating 0D byte. A stop byte, FF, follows the 0D of the last line of the program. (This stop byte is retained when another program is MERGEd, marking the boundary between programs.)

STORAGE OF VARIABLES

Each numeric variable consists of 8 bytes. The first three nybbles represent the exponent, in ten's-complement form. The fourth nybble specifies the sign of the mantissa (0=+, 8=-). The ten digits of the mantissa are represented (in binary-coded decimal) by the third to seventh bytes, and the final byte is always zero.

A fixed string variable (A$ to Z$) begins with F5, which identifies the variable as containing a string. As many as seven character codes may follow, with 00 used to terminate shorter strings.

Each variable with a two-character name, and each array variable, is preceded by a seven-byte header, constructed as follows:

Byte 0: ASCII code for first character of name.
(For the extended fixed-variable array A( ), 40 is used.)
Byte 1: ASCII code for second character of name; zero if none.
If an array, 80 is added. If a string variable with one-character name, 20 is added; with two-character name, 40.
Byte 2: High byte, size of remainder of variable or array
Byte 3: Low byte, size of remainder of variable or array
Byte 4: Dimension; second dimension if two. Zero if not an array
Byte 5: First dimension, if two-dimensional; zero otherwise.
Byte 6: Length of variable; if an array, length of each element

STORAGE INTO RESERVE MEMORY

The characters assigned to a RESERVE key follow a code identifying that key. The unused portion of RESERVE memory is filled with zeros. The codes for RESERVE keys are as follows:

A: 81	F: 86	K: 8B	SPC: F1	X: F8
B: 82	G: 87	L: 8C	$: F3	Z: FA
C: 83	H: 88	M: 8D	=: F4	
D: 84	J: 8A	N: 8E	V: F6	

SEARCHES FOR PROGRAM LINES

When a programmed GOTO or GOSUB is executed, a search for a higher-numbered line will begin at the current location in BASIC. Searches for line numbers lower than (or the same as) that of the line containing the GOTO or GOSUB begin at the start of the program. Searches for labels always begin at the start of the program.

PC-1350 BASIC CODES

00	33 3	65 e	97 TAN	C9 CLEAR
01	34 4	66 f	98 INT	CA USING
02	35 5	67 g	99 ABS	CB DIM
03	36 6	68 h	9A SGN	CC CALL
04	37 7	69 i	9B DEG	CD POKE
05	38 8	6A j	9C DMS	CE CLS
06	39 9	6B k	9D ASN	CF CURSOR
07	3A :	6C l	9E ACS	D0 TO
08	3B ;	6D m	9F ATN	D1 STEP
09	3C <	6E n	A0 RND	D2 THEN
0A	3D =	6F o	A1 AND	D3 ON
0B	3E >	70 p	A2 OR	D4 IF
0C	3F ?	71 q	A3 NOT	D5 FOR
0D ENTER	40 @	72 r	A4 ASC	D6 LET
0E	41 A	73 s	A5 VAL	D7 REM
0F	42 B	74 t	A6 LEN	D8 END
10	43 C	75 u	A7 PEEK	D9 NEXT
11	44 D	76 v	A8 CHR$	DA STOP
12	45 E	77 w	A9 STR$	DB READ
13	46 F	78 x	AA MID$	DC DATA
14	47 G	79 y	AB LEFT$	DD PAUSE
15	48 H	7A z	AC RIGHT$	DE PRINT
16	49 I	7B {	AD INKEY$	DF INPUT
17	4A J	7C ¦	AE PI	E0 GOSUB
18	4B K	7D }	AF MEM	E1 ABPAD
19	4C L	7E ~	B0 RUN	E2 LPRINT
1A	4D M	7F	B1 NEW	E3 RETURN
1B	4E N	80	B2 CONT	E4 RESTORE
1C	4F O	81	B3 PASS	E5 CHAIN
1D	50 P	82	B4 LIST	E6 CURSOR
1E	51 Q	83	B5 LLIST	E7 GPRINT
1F	52 R	84	B6 CSAVE	E8 LINE
20 SPC	53 S	85	B7 CLOAD	E9 POINT
21 !	54 T	86	B8 MERGE	EA PSET
22 "	55 U	87	B9	EB PRESET
23 #	56 V	88	BA	EC BASIC
24 $	57 W	89	BB OPEN	ED TEXT
25 %	58 X	8A	BC CLOSE	EE OPEN$
26 &	59 Y	8B	BD SAVE	
27 '	5A Z	8C	BE LOAD	F5 spade
28 (	5B [	8D	BF CONSOLE	F6 heart
29)	5C ¥	8E	C0 RANDOM	F7 diamond
2A *	5D]	8F	C1 DEGREE	F8 club
2B +	5E ^	90	C2 RADIAN	F9 ■
2C ,	5F _	91 LN	C3 GRAD	FA □
2D -	60 `	92 LOG	C4 BEEP	FB π
2E .	61 a	93 EXP	C5 WAIT	FC √
2F /	62 b	94 SQR	C6 GOTO	FD
30 0	63 c	95 SIN	C7 TRON	FE
31 1	64 d	96 COS	C8 TROFF	FF
32 2				

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

__ Enroll me as a 1985 Subscriber (Issue numbers 37-44). $24.00 in U.S. (U.S. $30.00 to Canada/Mexico. Elsewhere U.S. $40.00 payable in U.S. funds against a U.S. bank.)
__ Enroll me as a 1984-85 Subscriber (Issue numbers 31-44). $42.00 in U.S. (U.S. $51.00 to Canada/Mexico. Elsewhere U.S. $70.00 payable in U.S. funds against a U.S. bank.)
__ Enroll me as a 1983-85 Subscriber (Issue numbers 21-44). $78.00 in U.S. (U.S. $93.00 to Canada/Mexico. Elsewhere U.S. $120.00 payable in U.S. funds against a U.S. bank.)
__ Enroll me as a 1982-85 Subscriber (Issue numbers 11-44). $102.00 in U.S. ($125.00 to Canada/Mexico. Elsewhere U.S. $160.00 payable in U.S. funds against a U.S. bank.)
__ Check here if paying by MasterCard or VISA. Please give credit card information below.

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

Name: _____

Addr: _____

City: _____ State: ____ Zip: _____

MC/VISA #: _____

Signature: _____ Exp. Date: _____

mail this order form to:

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

POCKET COMPUTER NEWSLETTER

Issue 42 - September/October

Photo *The New Sharp PC-2500 Portable Notebook Computer.*

Another *Personal Information*™ product.

SHARP ANNOUNCES PC-2500 PORTABLE NOTEBOOK COMPUTER

Sharp Electronics Corporation has officially introduced a new portable notebook computer dubbed the model PC-2500.

The unit measures approximately 11-3/4 x 8-1/4 x 1-3/4 inches. It sports a full-sized typewriter-style keyboard, augmented by eight special function keys. A liquid-crystal display provides four lines of 24 characters per line or a graphics mode containing 150 x 32 pixels. There is also a built-in printer/plotter that can produce text and graphics in four colors. Its weight is under three pounds.

The PC-2500 is said to contain ROM software enabling it to be programmed in BASIC. It also is reported to contain two applications programs: a spreadsheet-style "business" package and a database-style "phone" directory.

Rechargeable nickel-cadmium batteries are said to be capable of supplying the unit for up to 100 hours at a time. Additionally, a built-in serial input/output port enables the unit to be connected to other computers, a printer, or external modem.

The PC-2500 comes equipped with 5 kilobytes of user RAM. This can be expanded to a total of 13 kilobytes or 21 kilobytes by installing optional 8K (CE-201M) or 16K (CE-202M) memory expansion cards. These battery-backed memory modules are said to be easily inserted or removed from a slot in the side of the computer.

The PC-2500 may prove to be of special interest to value added retailers and software vendors. This is because custom software applications can be provided on the CE-201M and CE-202M memory expansion cards. Additionally, since the PC-2500 apparently utilizes the same CPU as the pocket model PC-1350, the PC-2500 may be capable of serving as a more convenient development station for PC-1350 applications.

The price of the new computer in the United States of America is said to be pegged at $395.00. For further information contact: *Sharp Electronics Corporation, Systems Division, 10 Sharp Plaza, Paramus, NJ 07652.*

MEMORY EXPANSION MODULES FOR THE HP-71B

Want a lot more RAM in your HP-71B? Contact: *Hand Held Products, Inc., P.O. Box 2388, Charlotte, NC 28211.* (Telephone (704) 541-1380.) They have a series of modules that permit you to add as much as 96K of CMOS RAM or mix RAM and EPROM. The memory installs in place of the optional card reader.

Photo *RAM Expansion Modules for the HP-71B.*

FOR PC-1250/51/60/61 USERS

KEYBOARD AND DISPLAY I/O ON THE PC-1250

Rick Wenger, 6221 - 18th Avenue, Kenosha, WI 53140 is the provider of this article. He wishes to precede the article with the following disclaimer:

Although all routines presented here have been tested and run (sometimes for several days) on a PC-1250, there can be no guarantee that your PC-1250 will be the same. It is thus possible that you might do harm to your computer or connected peripherals by trying these routines. It is a healthy precaution, when experimenting with I/O, to be ready to use the RESET button at a moments notice.

In particular, you should be careful about sending output signals to a device at any rate faster than it is normally required to process. As an example, I once set up a loop to turn the display on and off as rapidly as possible. Upon execution, the top four rows of the left half of the display became extremely dark, while the rest of the display remained off. I quickly issued a RESET, but a "shadow" remained in the upper left-hand corner of the display for more than a week. If I had not reacted so quickly, I might well have lost some segments of the display.

Be careful! Apply the information contained herein at your own risk.

About the Keyboard

You can sense the keyboard from within machine language programs. The accompanying chart may be viewed as the electrical arrangement of the keyboard. As an example, if &04 is output through port C and &00 is output through port D, and an INA KBD instruction is invoked while the letter H key is pressed, the value &40 will be loaded into the CPU's accumulator. The following is a brief program, (in two parts, a BASIC portion and a machine language section) that will enable you to verify this information on your unit.

BASIC portion of the "key sense" program:

```
100 "KEY SENSE" INPUT "C? ";C,"D? ";D:POKE &C2FE,C,D
110 CALL &C300:PRINT "A= ";PEEK &C2FC:GOTO 100
```

Machine language portion of the "key sense" program:

	MNEMONICS	COMMENTS	ADDRESS	MACHINE CODE
	LDW C2FE	Load ports C	C300	10 C2 FE
	LDP 5C	and D with	C303	12 5C
	LD1(P)(W)	values from BASIC.	C305	1A
	OUT C	Send	C306	5D
	OUT D	strobe signals.	C307	DD
LOOP	INA KBD	Wait for	C308	4C
	CPA 00	key to be	C309	67 00
	RZS LOOP	pressed.	C30B	39 04
	LDW C2FC	Store contents	C30D	10 C2 FC
	LD(W) A	of accumulator to	C310	52
	RTS	(C2FC) & exit.	C311	37

Note that this "key sense" routine communicates with the user in decimal rather than hexadecimal. (All you really need to know is that 16d = 10h, 32d = 20h, 64d = 40h and 128d is equal to 80h.)

Once we know how to sense the keyboard, we can write a routine to return the character code of any key that is

Chart *Port C and D Outputs When PC-1250 Keys Are Pressed.*

	00-04	00-02	00-01	01-00	02-00	04-00	08-00	10-00	20-00	40-00
10	DEF	SHIFT	↑	↓	←	→				
20	Q	W	E	R	T	Y	U	I		
40	A	S	D	F	G	H	J	K	L	
80	Z	X	C	V	B	N	M	SPC	ENTER	0
02	8	9	CL	7						
08	5	6	/	4	P	0				
04	2	3	*	1	=					
01	.	+	-							

pressed. This is not necessary, however, since such a routine in already available in the ROM of the PC-1250. The routine that I call "KSCAN" begins at address 1F44. Chances are that a similar routine exists in the PC-1350 and other pocket computers that use the same kind of CPU chip. The character codes returned by this KSCAN routine are not in ASCII format. They are the codes shown in the accompanying table. (Most of these codes were originally published in Issue 26 of *PCW*.)

About the Display

The display memory of the PC-1250 is split into two parts. The left half of the display is controlled by bits in addresses F800 - F83B. The right half by bits in addresses F840 - F87B. The right half is ordered in reverse fashion to the left as far as memory addressing is concerned. See the accompanying diagram for a mapping illustration.

To facilitate discussion, I have numbered the columns of LCD dots from 1 to 120 decimal going from left to right. You can complete your understanding of the display using the simple (two-part) program provided next. The first portion of this routine is in BASIC:

```
200 "DISPLAY TEST" INPUT "ADDR? ";A
210 INPUT "BYTE? ";B
220 WAIT 0:PRINT "":POKE A,B:CALL &C320:GOTO 200
```

The second part is in machine language:

	MNEMONICS	COMMENTS	ADDRESS	MACHINE CODE
	LDP 5F	Turn	C320	12 5F
	LDA 01	on	C322	02 01
	EXA (P)	the	C324	DB
	OUTF	display.	C325	DF
	SNOP	Do nothing	C326	4D
	SNOP	for a shot time.	C327	4D
L1	IN 08	Wait for BRK	C328	6B 08
	RZS L1	key to be pressed.	C32A	39 03
L2	IN 08	Wait for BRK	C32C	6B 08
	RZC L2	key to release.	C32E	29 03
	RTS	Exit to BASIC.	C330	37

To try this routine execute RUN 200. Enter a valid display address and a byte in the range &00 - &7F. Use the BRK key to exit the display loop. After you have tried several display addresses, you might want to see if you can modify the BASIC part of the program to allow input of the display column numbers instead of absolute addresses.

You may notice, when running this program, that the

Diagram *Memory Map of the Display Bits for the PC-1250.*

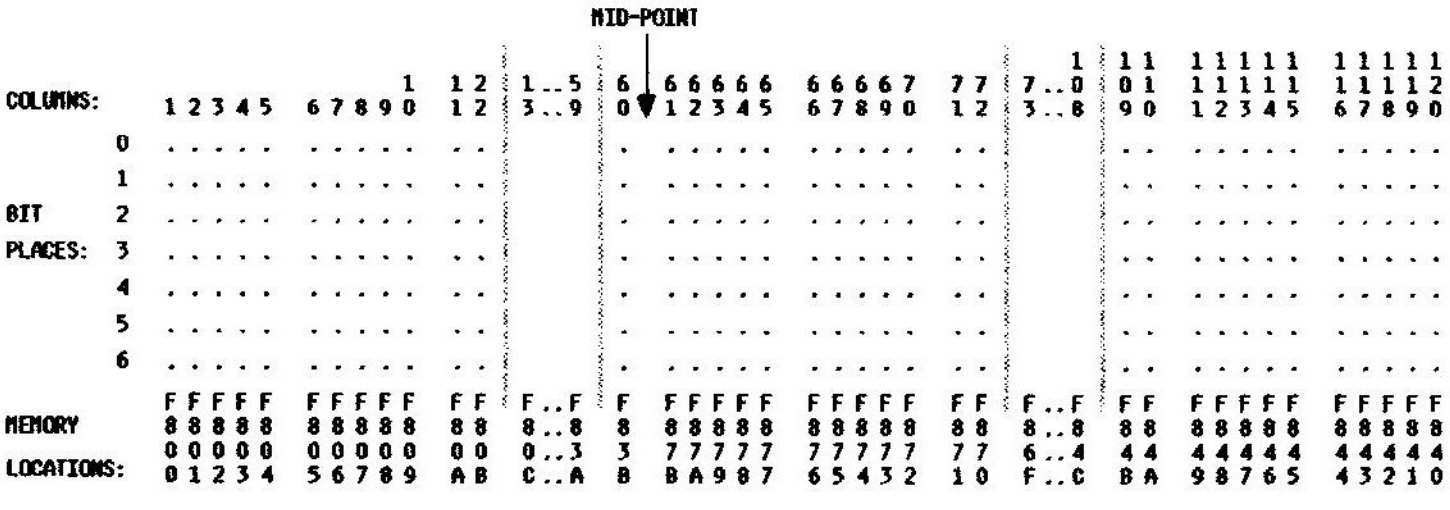


Table ***PC-1250 Character Codes.***

00	Enter	43	3
02	CL	44	4
07	BRK	45	5
09	Shift	46	6
0A	DEF	47	7
0C	↑	48	8
0D	↓	49	9
0E	►	4A	.
0F	◄	4B	E
10	⌴ (insertion)	4E	_ (underscore)
11	Space		
12	"	51	A
13	?	52	B
14	!	53	C
15	#	54	D
16	%	55	E
17	¥	56	F
18	$	57	G
19	π	58	H
1A	√	59	I
1B	'	5A	J
1C	;	5B	K
1D	:	5C	L
1E	@	5D	M
1F	&	5E	N
30	(	5F	O
31	)	60	P
32	>	61	Q
33	<	62	R
34	=	63	S
35	+	64	T
36	-	65	U
37	*	66	V
38	/	67	W
39	^	68	X
40	Ø	69	Y
41	1	6A	Z
42	2	FF	no key

display is noticeably weaker than the one you normally see. This will be particularly so if you are using a weak set of batteries.

One way to perk up the display when running the routine is to insert a REFR nn instruction in the display loop. Change the machine language portion of the routine by doing the following: POKE &C326,&4E,&A0 and then POKE &C32B,&05. Try executing the program now. Do you like what you see? Can you figure out why the display dims when you press the BRK key?

Even this improved display did not satisfy the people who developed the ROM in the PC-1250. Make a few more pokes in the test routine as follows:

```
POKE &C323,&05
POKE &C326,&6B,&01
POKE &C32B,&06
```

This will cause the machine language portion of the display to be as follows:

	MNEMONICS	COMMENTS	ADDRESS	MACHINE CODE
	LDP 5F	Prepare	C320	12 5F
	LDA 05	port F.	C322	02 05
	EXA (P)	Turn slow	C324	DB
DISPL	OUTF	display on.	C325	0F
	IN 01	Super-slow mode.	C326	6B 01
	IN 08	Stay in display	C328	6B 08
	RZS DISPL	until BRK occurs.	C32A	39 06
L2	IN 08	Wait for BRK	C32C	6B 08
	RZC L2	key to release.	C32E	29 03
	RTS	Exit to BASIC.	C330	37

This routine puts the display loop into a "super-slow" mode that crawls along at just 7.4 loops per second. At this rate the display appears to wink about every half a second. If you replace the first IN 08 directive with a couple of SNOPs, you will see the phenomena disappear.

The display routine provided in the ROM alternates between using the "super-slow" mode and the REFR nn method of display maintenance. The method used depends on whether or not a key is being pressed.

Here are a few other useful ROM entry points that can be of use in handling the display:

At address &11A1 is a routine that dumps the contents of the display buffer (at addresses &C7B0 - &C7FF) to the display memory. It utilizes the character bit maps that are at addresses &4464 - &45DA.

At address &1D06 is a routine that maintains a nice display while scanning the keyboard. When a key is pressed, A is loaded with the appropriate code and control returns to the calling program.

The following short program can be used to fill the screen with the character specified by touching any key:

	MNEMONICS	COMMENTS	ADDRESS	MACHINE CODE
	LDW C6DA	Supress	C340	10 C6 DA
	OR(W) 01	cursor	C343	D5 01
	LDA "SPC"	Use to clear bfr	C345	02 11
FILBF	LDW C7B0	Fill buffer with	C347	10 C7 B0
	LDCO 23	23 decimal (con-	C34A	00 17
	LDO(W) A	tents of the acc.)	C34C	1F
	DEA	If ENTER key is	C34D	43
	FCC CONT	pressed, then	C34E	2A 02
	RTS	exit this routine.	C350	37
CONT	SS BTD	Call ROM routine	C351	F1 A1
	SS DHOLD	Call ROM routine	C353	FD 06
	REV FILBF	Loop back	C355	20 0F

To start this routine, simply call &C340. To exit, hit the ENTER key. Implementing SHIFT status will cause it to sustain until SHIFT is pressed a second time. Be careful though, some non-displaying shifted character can cause a crash. (Use the RESET button while depressing a key to recover from such a crash.)

If you want to do a little creative work on your own, why not see if you can write a program that allows the user to design a unique character and have it move around the display under control of the left/right cursor movement keys? Be sure to include provision for escaping from the program (without having to use the RESET button)!

Some Additional Information for Machine Language Buffs

The information in this section provides additional understanding of the PC-1250 from a machine language point of view.

The operation of the ON/RSV/PRO/RUN/OFF switch has been discussed previously. It is also important to realize that flipping this switch to the OFF position defeats the BRK key. Thus, if the computer is in a machine language loop and you flick this switch to OFF, the loop will keep repeating. But, the instruction IN 08 will no longer sense the BRK key (by clearing Z when BRK is pressed). If you subsequently turn the

switch back on, a brief BRK pulse will be sent out. This pulse can be recognized by an IN 08 instruction.

Pushing in the RESET button on the back of the PC-1250 interrupts normal sequential operations and restarts the CPU from address 0. Analysis of the start up routine indicates that the directive IN 40 checks for continued pressing of the RESET button. It does this by clearing the Z flag when RESET is closed (down). However, this is not possible to test experimentally.

A BRK pulse will also restart operations from address 0 when the CPU has been shut down by outputting 08 through port F. The CPA as well as RAM memory is preserved through power-down/power-up.

Once the RESET button is released, the PC-1250 has no apparent way of knowing whether start up was caused by the RESET button, cycling of the OFF/ON switch, or using BRK after automatic power-off. Thus, it appears to be the method of power-down that the PC-1250 flags (to itself) by leaving the CPA in various states, that determines whether the display comes back, whether program memory is preserved, and so forth. You can verify this by using the RESET button after an automatic power-off condition. There does not appear to be any need to hold down any other keys. Program, data and display are retained.

You might learn more about the methods of powering down by studying the following ROM entry points:

61EA3 - automatic power-off routine.

64A57 - OFF switch power-down routine.

The Appendix in the PC-1250 Manual gives the following "last resort" RESET procedure: "Push the ALL RESET button in for about 10 seconds. This should clear any problem..." In fact, you can hold the RESET for 10 seconds or 10 minutes in some instances and it will have little effect. A better procedure is as follows. Makes sure the slide switch is set to the RUN position. Push the RESET button firmly and then release. Push the RESET button a second time and release. If this does not clear up any problems, then check your power source.

The regular high pitched BEEP sound can be generated by outputting the value 30 through port F. This produces a high concert B pitch. Sending the value 20 will provide a tone that is about an octive lower.

It is possible to create many other tones by tightening the beeper (10 out through port F) and slackening the beeper (00 out through port F) within a precisely timed program (machine language) loop.

The CPU is capable of sensing two different internal clock signals. IN 01 clears the Z flag approximately every half-a-second. IN 02 clears the Z flag approximately every two milliseconds.

FOR HEWLETT-PACKARD HP-71B USERS

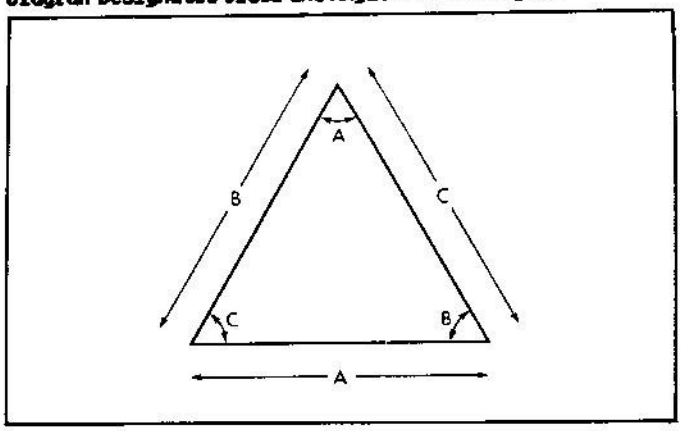
PROGRAM FOR SOLVING TRIANGULATION PROBLEMS

Surveyors and architectural engineers often need to find the solutions to problems involving triangles. This program can assist those dealing with such tasks. If you give it three parts of a triangle, including at least one side (see the accompanying diagram), it can generally find the remaining side(s) and angle(s).

Once the program has been loaded into your HP-71B, just respond to the prompts to utilize it. Unknown parts are designated by using the self-prompting value of zero. You can check that you keyed the program in correctly by verifying that the program has a byte count of 1319. Then confirm proper operation by executing the program using the inputs shown in the accompanying example. You should get the same output results.

This program was adapted for the HP-71B from a program originally submitted by *Norlin Rober*.

Diagram ***Designated Sides and Angles of a Triangle.***



Example ***Operation of the Triangles Program.***

```
SIDE A? 5
SIDE B? 0
SIDE C? 8
ANGLE A? 35
ANGLE B? 0
ANGLE C? 0
1st OF 2 SOLUTIONS:
SIDE A=  5.0000
SIDE B=  8.5393
SIDE C=  8.0000
ANGLE A=  35.0000
ANGLE B=  78.4047
ANGLE C=  66.5953
AREA=  19.5918
2nd SOLUTION:
SIDE A=  5.0000
SIDE B=  4.5671
SIDE C=  8.0000
ANGLE A=  35.0000
ANGLE B=  31.5953
ANGLE C=  113.4047
AREA=  10.4783
```

Program *Solving Triangles on the HP-71B.*

```
100 DELAY 8,.1 @ WIDTH 96 @ DESTROY ALL @ DEGREES @ FIX 4 @ DIM A(6)
110 INPUT "SIDE A? ",'0';A(1) @ IF A(1)=0 THEN 120 ELSE I=1
120 INPUT "SIDE B? ",'0';A(2) @ IF A(2)=0 THEN 130 ELSE I=I+2
130 INPUT "SIDE C? ",'0';A(3) @ IF A(3)=0 THEN 140 ELSE I=3-I
140 IF I=0 THEN 500
150 INPUT "ANGLE A? ",'0';A(4) @ IF A(4)=0 THEN 160 ELSE J=1
160 INPUT "ANGLE B? ",'0';A(5) @ IF A(5)=0 THEN 170 ELSE J=J+2
170 INPUT "ANGLE C? ",'0';A(6) @ IF A(6)=0 THEN 180 ELSE J=3-J
180 IF A(I)=0 THEN 300
200 L=90-A(4)-A(5)-A(6)+90 @ IF L<0 THEN 700
210 A(J+3)=L @ FOR K=1 TO 3 @ IF K<>I THEN A(K)=SIN(A(K+3))*A(I)/SIN(A(I+3))
220 NEXT K @ GOTO 600
300 IF J=I THEN 400
310 G=A(J) @ H=A(6-I-J) @ L=A(J+3) @ IF H*SIN(L)>G THEN 700
320 M=ASN(H*SIN(L)/G) @ A(9-I-J)=M @ IF L+M>=180 THEN 700
330 A(I+3)=180-L-M @ A(I)=G*SIN(L+M)/SIN(L)
340 IF G>H*SIN(L) AND H>G THEN PRINT "1st OF 2 SOLUTIONS:" @ N=1
350 GOTO 600
360 A(9-I-J)=180-M @ A(I+3)=M-L @ A(I)=G*SIN(M-L)/SIN(L) @ N=0
370 PRINT "2nd SOLUTION:" @ GOTO 600
400 L=A(I+3) @ FOR K=4 TO 6
410 IF K<>I+3 THEN G=A(K+3) @ H=A(9-I-K) @ A(K)=90
420 IF H<>G*COS(L) THEN A(K)=ATN(G*SIN(L)/(H-G*COS(L)))
430 IF -A(K) THEN A(K)=180+A(K)
440 NEXT K @ A(I)=SQR(G*G*SIN(L)*SIN(L)+(G*COS(L)-H)*(G*COS(L)-H)) @ GOTO 600
500 IF (A(1)+A(2)-A(3))*(A(1)+A(3)-A(2))*(A(2)+A(3)-A(1))<=0 THEN 700
510 A(4)=ACS((A(2)*A(2)+A(3)*A(3)-A(1)*A(1))/(2*A(2)*A(3)))
520 A(5)=ACS((A(1)*A(1)+A(3)*A(3)-A(2)*A(2))/(2*A(1)*A(3)))
530 A(6)=ACS((A(1)*A(1)+A(2)*A(2)-A(3)*A(3))/(2*A(1)*A(2)))
600 K=.5*A(1)*A(2)*SIN(A(6)) @ PRINT "SIDE A= ";A(1)
610 PRINT "SIDE B= ";A(2)
620 PRINT "SIDE C= ";A(3)
630 PRINT "ANGLE A= ";A(4)
640 PRINT "ANGLE B= ";A(5)
650 PRINT "ANGLE C= ";A(6)
660 PRINT "AREA= ";K
670 IF N THEN 360
680 END
700 PRINT "NO SOLUTION"
```

FOR PC-1500 & PC-2 USERS

BOWLING SCORE KEEPER

Here is a program that will take care of score keeping during a bowling game. When started, the program prompts for the number of players and their names. Use the main player menu thereafter to select a player whose score is to be updated. If you make a player selection error, enter a score greater than 10 to return to the menu. You need at least an 8K memory module and a printer/plotter to utilize this program.

The program was provided by *Robert Sincick, 9610 S.W. Davies Road, Beaverton, OR 97005.*

BOWLING

SUE	30	60	90	120	150	180	210	240	270	300	300	
JOE	19	38	57	76	95	114	133	152	171	190	190	
JIMMY	7	26	35	43	51	59	68	87	96	103	103	
ROBERT	9	18	36	44	53	60	24	82	102	122	122	
SAM	18	22	41	50	20	89	98	102	116	125	125	
JANE	20	40	60	80	100	120	140	160	180	200	200	

Program *Bowling Score Keeper (PC-1500/PC-2).*

```
1000 "BOWL"REM BOWLING SCORES GRAPHICS VERSION 6/13/84, BY BOB SINCICK
1005 "B"
1010 GRAPH :CLEAR
1020 "IN"INPUT "HOW MANY PLAYERS? ";N:IF N>6GOTO "IN"
1030 DIM N$(6):DIM P(6,14):DIM B(6,2):DIM W$(0)*26
1040 FOR X=1TO 6:FOR Y=11TO 12:P(X,Y)=1:NEXT Y:NEXT X
1050 FOR X=1TO 6:FOR Y=13TO 14:P(X,Y)=0:NEXT Y:NEXT X
1060 WAIT 0:FOR Z=1TO N:PRINT "PLAYER NO.";(Z);" IS:";
1070 INPUT N$(Z):CLS :NEXT Z
1080 CSIZE 5:LPRINT "BOWLING":GLCURSOR (0,-150):SORGN :CSIZE 2
1090 FOR X=0TO 216STEP 72
1100 LINE (X,100)-(X,-410):LINE ((X+36),-410)-((X+36),100):NEXT X
1110 FOR Y=0TO -360STEP -72:LINE (0,Y)-(216,Y)
1120 IF ABS Y>=360GOTO 1140
1130 LINE (216,(Y-36))-(0,(Y-36))
1140 NEXT Y
1150 Z=1:FOR X=182TO 2STEP -36
1160 GLCURSOR (X,100):ROTATE 1:LPRINT USING "&&&&&&&&";N$(Z):Z=Z+1:NEXT X
1170 FOR Y=-36TO -360STEP -72:FOR X=18TO 198STEP 36
1180 LINE (X,Y)-(X,(Y+18))-((X+18),(Y+18)):NEXT X
1190 FOR X=216TO 36STEP -36
1200 LINE (X,(Y-18))-((X-18),(Y-18))-((X-18),(Y-36)):NEXT X:NEXT Y
1210 Y=-360:FOR X=18TO 198STEP 36
1220 LINE (X,Y)-(X,(Y-18))-((X+18),(Y-18))
1230 LINE (X,(Y-18))-(X,(Y-36))-((X+18),(Y-36)):NEXT X:GLCURSOR (0,0)
1240 W$(0)=" "+LEFT$ (N$(1),3)+" "+LEFT$ (N$(2),3)+" "+LEFT$ (N$(3),3)+" "
1250 W$(0)=W$(0)+LEFT$ (N$(4),3)+" "+LEFT$ (N$(5),3)+" "+LEFT$ (N$(6),3)
1260 CLS :USING :WAIT 0:PRINT W$(0);
1270 C$=INKEY$
1280 C=ASC (C$):IF (C<17)+(C>22)THEN GOTO 1270
1290 C=C-16:IF P(C,0)=1BEEP 4:GOTO 1260
1300 WAIT 00:CLS :PRINT N$(C);" F-";P(C,11);" BALL-";P(C,12);" ";
1310 INPUT B(C,(P(C,12)))
1320 IF P(C,12)=1LET B(C,2)=0
1330 IF (B(C,1)+B(C,2))>10OR (B(C,1)+B(C,2))<0BEEP 1:CLS :PAUSE "WRONG INPUT !"
GOTO 1260
1340 IF P(C,12)=1GOTO "FF"
1350 GOTO "PFST"
1360 "FF"IF P(C,11)=1GOTO "ST?"
1370 GOTO "PFSP"
1380 "PFSP"IF P(C,13)=1GOSUB "ACB":GOSUB "PLF":GOSUB "RSM":GOTO "ST?"
1390 GOTO "PPFST"
1400 "ACB"P(C,(P(C,11)-1))=P(C,(P(C,11)-2))+10+B(C,1):RETURN
1410 "RSM"P(C,13)=0:RETURN
1420 "ST?"IF B(C,1)=10GOSUB "SSM+1":GOSUB "PST":GOSUB "SBM1":GOTO 1260
1430 P(C,12)=2:GLCURSOR (0,-400):GOTO 1260
1440 "PFST"IF P(C,14)=1GOSUB "CF-1":GOSUB "PLF":GOSUB "SSM-1"
1450 GOTO "SP?"
1460 "SP?"IF B(C,1)+B(C,2)=10GOSUB "SSM":GOSUB "PSP":GOSUB "SBM1":GOTO 1260
1470 GOSUB "CF":GOSUB "PFS":GOSUB "SBM1":GOTO 1260
1480 "PPFST"IF P(C,14)=2GOSUB "CPPF":GOSUB "PPPF":GOSUB "SSM-1"
1490 GOTO "ST?"
1500 "PST"COLOR 1:Z=18:X=216-(((C-1)*36)+18):IF P(C,11)=11LET Z=36
1510 IF P(C,11)=12LET Z=54
1520 FOR Y=((-36*P(C,11)+Z))TO ((-36*P(C,11))+(Z-18))STEP -2
1530 LINE (X,Y)-((X+18),Y):LINE ((X+18),(Y-1))-(X,(Y-1))
1540 NEXT Y:GLCURSOR (0,Y):RETURN
1550 "CF-1"P(C,(P(C,11)-1))=P(C,(P(C,11)-2))+B(C,1)+B(C,2)+10:RETURN
1560 "PLF"GLCURSOR ((217-(36*C)),(83-(36*P(C,11))))
1570 LPRINT USING "####";P(C,(P(C,11)-1)):IF P(C,11)-1=10GOSUB "LF"
1580 RETURN
1590 "CF"P(C,(P(C,11)))=P(C,(P(C,11)-1))+B(C,1)+B(C,2):RETURN
1600 "PFS"GLCURSOR ((217-(36*C)),(47-(36*P(C,11))))
1610 LPRINT USING "####";P(C,P(C,11)):IF P(C,11)=10GOSUB "LF"
1620 RETURN
1630 "SBM1"P(C,12)=1:P(C,11)=P(C,11)+1:GLCURSOR (0,-400):COLOR 0:RETURN
1640 "SSM+1"P(C,14)=P(C,14)+1:RETURN
1650 "SSM"P(C,13)=P(C,13)+1:RETURN
1660 "SSM-1"P(C,14)=P(C,14)-1:RETURN
1670 "PSP"COLOR 2:X=216-(((C-1)*36)+18)
1680 Y=((-36*P(C,11)+18)):Z=18:FOR G=1TO 18:LINE (X,Y)-((X+Z),(Y-18)):Z=Z-1
1690 LINE ((X+Z),(Y-18))-(X,(Y-G)):NEXT G:GLCURSOR (0,Y-18):COLOR 0:RETURN
1700 "CPPF"P(C,(P(C,11)-2))=P(C,(P(C,11)-2))+P(C,(P(C,11)-3))+20+B(C,1):RETURN

1710 "PPPF"GLCURSOR ((217-(36*C)),(119-(36*P(C,11))))
1720 LPRINT USING "####";P(C,((P(C,11))-2)):IF P(C,11)-2=10GOSUB "LF"
1730 GLCURSOR (0,(72-(36*P(C,11)))):RETURN
1740 "LF"COLOR 3:GLCURSOR ((217-(36*C)),-355):LPRINT USING "####";P(C,10)
1750 P(C,0)=1:GLCURSOR (0,-400):COLOR 0:RETURN
```

FOR PC-1350 USERS

DISPLAYING GRAPHS

Using this program, the Sharp PC-1350 can display the graph of an equation given in either rectangular or parametric form.

The equation to be graphed must be entered into the program, beginning at line 100. As an example (included in the accompanying program listing), to graph Y = SIN X, line 100 would be: 100 Y = SIN X.

Be sure the PC is set to the DEGREE mode. Then RUN the program. In response to the query "TYPE (R/P)?", enter R, since the equation to be graphed will use rectangular coordinates. Use 0 and 360 as values for MIN X and MAX X. For MIN Y and MAX Y use -1 and 1 respectively. (These inputs determine the portion of the plane that is to be included in the graph.) In response to the query "RATIO, W/H?", enter 2. This will make the width of the graph twice its height. (A ratio of 5 would widen the graph to fill the entire display.) The display will then plot a sine curve on the display, with the X- and Y- axes included.

The ellipse determined by the parametric equations expressed as X = 2 cos t, y = sin t may be plotted by entering:

```
100 X = 2*COS T
110 Y = SIN T
```

For this case, enter P in response to "TYPE R/P?". Enter MIN T and MAX T values of 0 and 360. Use 30 for "NO OF INCR" (the number of increments to be made to the value of T). Use -2 and 2 as minimum and maximum values of X. Use -1 and 1 as the values for Y. Specify a W/H ratio of 2. Observe the graph.

Polar-coordinate graphs may be treated as parametric forms. To plot the three-leaf rose having the polar equation r = cos (theta), consider T as the symbol for theta and enter the lines:

```
100 R = COS (3*T)
110 X = R*COS T, Y = R*SIN T
```

(Line 110 converts polar to rectangular coordinates for plotting.)

Set T minimum to 0, maximum to 360, with 90 increments. Set X to go from -1 to 1 and the same for Y. Use a W/L ratio of 1. Watch the graph plot on the screen.

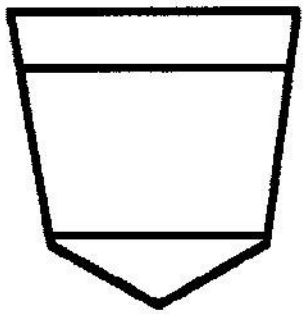
You can experiment to you heart's content with various equations, using different numbers of increments, W/H ratios, and so forth. In many cases you will find that the limited resolution of the display prevents curves from being as smooth as one might desire. In spite of this, the PC-1350 and this program is able to supply an approximately accurate graph!

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

Program *Graphs on the Sharp PC-1350.*

```
10:INPUT "TYPE (R/P)? "
   ;G$
11:IF G$="P" INPUT "MIN
    T? ";E,"MAX T? ";F
12:IF G$="P" INPUT "NO
   OF INCR? ";N
13:INPUT "MIN X? ";A,"M
   AX X? ";B
14:INPUT "MIN Y? ";C,"M
   AX Y? ";D
15:INPUT "RATIO, W/H? "
   ;L
20:K=30*L,HX=K/(B-A),VY
   =30/(C-D),HK=-A*HX+.
   5,VK=-D*VY+.5: CLS :
   WAIT 0
30:LINE (HK,0)-(HK,30):
   LINE (0,VK)-(K,VK)
40:IF G$="P" LET H=(F-E
   )/N,T=E: FOR I=0 TO
   N: GOSUB 100:T=T+H:
   NEXT I: WAIT :
   GPRINT : CLS : END
50:H=(B-A)/K,X=A: FOR I
   =0 TO K: GOSUB 100:X
   =X+H: NEXT I: WAIT :
   GPRINT : CLS : END
100:Y= SIN X
200:PSET (HX*X+HK,YY*Y+V
   K): RETURN
```

POCKET
COMPUTER
NEWSLETTER



Issue 43 - November

FOR PC-1250/51/60/61 USERS

AN ADDENDUM TO THE PC-1250 MANUAL

Rick Wenger, 6221 - 18th Avenue, Kenosha, WI 53140, thinks the "Student Computer Handbook" packed with the PC-1250 could use the following updating.

Strings and Stuff

Subscripts for one or two dimensional arrays may go from the value 0 to 255, not just 1 to 255. Thus, a statement such as DIM C(4,3) sets up an array of 5*4 elements. It consequently uses up a total of 5*4*8 bytes plus 6 bytes for the array header information, yielding a total of 166 bytes. This is far more than the 96 bytes indicated on page 96 of the manual.

The "extended memory" A-subscripted variable can be dimensioned in the implied manner of: A(29)=expression. In fact, just asking the computer to PRINT A(29) has the effect of creating three new variables: A(27), A(28) and A(29). Implied dimensioning of this type also has the effect of creating six header bytes. Thus, the number of bytes used in this example would be 3*8+6=30. At this point (unlike a regular B through Z dimensioned variable) the A-subscripted variable can still be re-dimensioned in the upward direction. A statement such as PRINT A(36) or A$(40)="TEST" is all that it takes. But, if at this point in time you make a regular DIM statement (such as DIM B$(10)), you set an upper limit to the A-subscript equal to its current value. This upper limit can not be revised without first invoking the CLEAR statement or RUN command.

It is interesting that the PC-1250 is a bit too easily confused as to the work space required to concatenate a string. Most *POW* readers are aware that there is an 80 byte limit to the length of a concatenated string imposed by PC BASIC. But, suppose dimensioned variables D$(0) and D$(1) each have a length of 50 characters. The PC-1250 will refuse to print LEFT$(D$(0),31))+LEFT$(D$(1),1)). This statement brings ERROR 5 because 31+LEN(D$(1))>80, even though only 32 bytes of work space are actually required!

Cassette Operations

A feature not mentioned in the manual is the ability of a password to be associated with the CSAVE command. The statement CSAVE "TITLE","PASSW" will save the current program in memory onto tape with the filename "TITLE". On CLOADing the program back into a PC-1250, however, the user will need to use the statement PASS "PASSW" in order to obtain a listing.

Another fact not mentioned in the manual is that CSAVE may be used in the RSV mode to store the contents of RSV memory on tape. You may, at your option, give your RSV key assignments a file name and CSAVE them on tape with the PC in the RSV mode. You can then CLOAD these assignments back into the computer at a later date using the CLOAD command with the PC in the RSV mode!

Any password is ignored if it is associated with CSAVE in the RSV mode. If you attempt to CLOAD a regular program while in the RSV mode, you will obtain an ERROR 6 message. If you attempt to CLOAD a RSV file into program memory, you will obtain useless data.

The descriptions of the PRINT# and INPUT# statements in the manual appear to be incorrect and misleading. For example, PRINT# "FILENAME" does *not* save all data in the computer onto tape. It only saves fixed variables A through Z plus any A-subscripted variables. There is no single statement that will save all data onto tape.

Additionally, the statement PRINT# "FILENAME";VARIABLE NAME does not save just the contents of the specified variable. It saves all A-subscripted variables starting from the named variable and continuing on up. This statement format does not work at all for regular dimensioned variables.

However, regular dimensioned variables may be saved using the form PRINT# "FILENAME";V(*) or PRINT# "FILENAME";V$(*). The statement INPUT# "FILENAME";X(*) or INPUT# "FILENAME";X$(*) will then load the data into the appropriate X or X$ array elements provided that X has been dimensioned appropriately and that the data types match.

Fixed, extended and dimensioned variables can be sent to tape together under one file name when using the PRINT# statement by separating the variable names with commas. They can later by loaded into the computer with a similar compound INPUT# statement. However, the pseudo-variable name A(*) is not permitted (as it is on the PC-1500) to store all dimensioned variables with a single specification.

The CSAVE, CSAVE M, PRINT# and INPUT# statements may all be used in a BASIC program. CLOAD, CLOAD M and MERGE may not be used with a program. CHAIN must also be invoked from within a program.

The MERGE command works differently in the PC-1250 from the way it works in the PC-1500. When the 1250 merges programs from tape, it does not flag the fact that a program has been merged. Thus a user "normally" has complete flexibility in running and editing merged programs. However, if line numbers are not in a natural sequence (such

as could normally be used in a single program), editing functions will decide that the unnatural sequence that occurs last in memory marks the first line of a program. RUN followed by a line number searches for the line number starting at the true start of program memory, but gives up the search when it reaches the first unnatural line sequence. RUN followed by a label, on the other hand, or a DEFine key sequence, will not cause the search to halt when an unnatural sequence is found. In this case, the entire length of memory will be examined for the appropriate label.

When a merged program is executing, GOTO a line number or GOSUB a line number causes the interpreter to search up or down program memory (depending on whether the desired line number is greater than or less than the current line being executed). It marks the first unnatural sequence encountered as the end of search point. Using GOTO or GOSUB with a label, on the other hand, causes the entire program memory to be examined.

An analysis of this methodology indicates that Sharp displayed wisdom in their handling of merged programs in this fashion. Merged programs that have non-conflicting labels can always be executed. And, the option to go in and edit conflicting line numbers starting with the last program that was merged, enables the user to integrate programs.

Other Features

Reservable key sequences that end with the ↵ symbol will execute automatically when called into the display, just as though the ENTER key had been pressed.

The & prefix causes the next four digits (excluding leading zeros) to be interpreted as a hexadecimal value. No decimal point is permitted in this format.

Within the range &2000<=address<=&FFFF, the statement PEEK(address) returns the decimal representation of the value residing in that RAM or ROM address. This statement has the same priority as SIN, COS, etc. Thus parenthesis are required around the address value unless it is a simple constant or variable.

POKE address,expression1,expression2,expression3,... will POKE the values corresponding to the expressions into successive RAM memory locations starting at the address location. The syntax only requires the presence of expression1 following the address.

CALL address will call a machine language routine that starts at the indicated address. Parameters can not be passed directly to/from the CPU registers as can be done on the PC-1500.

CSAVE M "TITLE";address1,address2 causes machine code to be saved on tape from address1 to address2. The title is optional. Hidden ROM (addresses 0 through &1FFF) can not be saved on tape using the CSAVE M command.

CLOAD M "TITLE";address searches for a machine language program having the indicated title on tape. When found, it loads the code into the computer starting at the indicated address. Both the title and address parameters are optional. If a title is not specified, the next machine language program encountered on the tape will be loaded. When the address value is not specified, the program will be loaded into the addresses from which it was stored on the tape.

The PC-1250 is able to distinguish between BASIC programs, data (variables) files and machine language programs stored on tape. Thus, duplicate titles may be used for the various types of files as long as the keyword (CSAVE, PRINT# or CSAVE M) indicating the type of storage is different.

No mention is made in the manual of using the CA key after hitting the shift key. To date I have discovered the following uses for the Shift/CA sequence:

1. Clears the program cursor pointer so that CL followed by the up-arrow or down-arrow brings up the first/last line of a BASIC program when in the PRO mode.
2. Two stroke substitute for PRINT=PRINT command.
3. Two stroke substitute for TROFF command.

Note that the use of CA disables the CONT operation.

Contrary to what is stated on page 64 of the manual, the RND function does not appear to repeat a sequence of numbers each time a program is executed. In fact, it seems that the only way to repeat a RND sequence of numbers is to press the reset button. This works whether or not a key is held down to preserve program status.

Contrary again to the manual (page 81), commas and semi-colons may not be mixed within a single PRINT statement. If a comma is used it must stand alone as a separator. If not, the message ERROR 1 results.

Finally, I want to add my two cents worth to the discussion of the 1200 series' unconventional truth value assignments.

NOT must *not* be used in a conditional IF statement. NOT, OR and AND are all true Boolean operators. But, in this BASIC dialect, a statement is considered true, if and only if, it has a positive truth value. (In most BASICs, the value -1 is considered true, 0 as false.) This convention works alright for AND and OR, but not for NOT. For example:

NOT(1=2) equals NOT(00000000) which equals 11111111 (binary). In the strange yet wonderful world of two's complement arithmetic in a closed 8-bit register, this is the equivalent of -1. Therefore, IF NOT(1=2) BEEP 1 will not produce a beep when it really should.

It is better in this dialect to use IF 0=(1=2) BEEP 1.

Indeed, if 0= is substituted for NOT everywhere in a program, the desired effect can be obtained. You can use this method to translate complex logic statements from BASIC programs used on other computers.

Alternately, you could add 2 after every NOT statement. But, this method prevents the use of + in place of OR or * in place of AND (as on the old PC-1211/PC-1).

Using the 0= system saves one byte per use and gives you complete interchangeability between + and OR and * and AND. Additionally, statements will translate to any of the various PC-1500 models!

FOR HEWLETT-PACKARD HP-71B USERS

HISTOGRAM PROGRAM FOR USE WITH THINKJET PRINTER

The program presented in this article may be used to collect, organize, and analyze data for presentation as a histogram. If you do not have a Hewlett-Packard Thinkjet printer, you can use the output from the display to manually sketch a histogram chart. This is often satisfactory when "out in the field." But, if you have a Thinkjet (or similar) printer that you can connect to your HP-71B, then this program will also produce a histogram that neatly summarizes your data.

Load the program from the accompanying listing and check your work by looking for a byte count of 2981 when you CATalog the file. When you execute the program you will get a "revolving" menu that advances to the next choice each time you elect not to choose the current option.

Basically, the program expects you to input a set of values. Statistical information is compiled from this data set (by selecting the REVIEW STATS option). This statistical information includes: the minimum, maximum, mean, standard

Listing ***Histogram Program for the HP-71B.***

```
5 DELAY 1,.1 @ WIDTH 96 @ DESTROY ALL @
PRINTER IS * @ PWIDTH 80
10 G=1 @ DIM X(255),F(45),S$[80]
20 DISP "HISTOGRAM"
40 DELAY 1,.1 @ FIX 2 @ DISP "HISTOGRAM
MENU" @ A$="" @ B$=""
45 B=1 @ INPUT "ENTER DATA? ",'N';B$ @ I
F UPRC$(B$[1,1])="Y" THEN 55
46 B=2 @ INPUT "REVIEW STATS? ",'N';B$ @
 IF UPRC$(B$[1,1])="Y" THEN 55
47 B=3 @ INPUT "CALCULATE CELLS? ",'N';B
$ @ IF UPRC$(B$[1,1])="Y" THEN 55
48 B=4 @ INPUT "PRINT HISTOGRAM? ",'N';B
$ @ IF UPRC$(B$[1,1])="Y" THEN 55
49 B=5 @ INPUT "REVIEW DATA? ",'N';B$ @
IF UPRC$(B$[1,1])="Y" THEN 55
50 B=6 @ INPUT "HELP? ",'N';B$ @ IF UPRC
$(B$[1,1])="Y" THEN 55
51 GOTO 45
55 ON B GOTO 60,90,220,410,710,780
60 DELAY 2,.1 @ PRINT "ENTER DATA    Q=Qu
it"
70 FIX 0 @ G$=STR$(G) @ G$=G$[1,LEN(G$)-
1] @ PRINT "DATA ("&G$&")? "; @ INPUT ""
,'Q';Z$
75 IF Z$="Q" THEN 40 ELSE X(G)=VAL(Z$)
80 G=G+1 @ GOTO 70
90 GOSUB 930 @ DELAY .05,.05 @ PRINT @ P
RINT
95 DELAY 2,.1 @ PRINT "STATISTICS" @ DIS
P "CALCULATING STATS" @ DELAY 8,.5
100 FOR J=1 TO G @ FOR Q=1 TO G-J
110 IF X(Q)>X(Q+1) THEN T=X(Q) @ X(Q)=X(
Q+1) @ X(Q+1)=T
120 NEXT Q @ NEXT J @ J=J-1
130 Z=0 @ S=0 @ FOR I=2 TO J @ S=S+X(I)
@ NEXT I @ M=S/(J-1)
140 FOR I=2 TO J @ Z=Z+X(I)*X(I) @ NEXT
I
150 V=(Z-(J-1)*M*M)/(J-2)
160 PRINT "DATA MIN=";X(2)
170 PRINT "DATA MAX=";X(J)
180 PRINT "MEAN     =";M @ PRINT "STD DEV
 =";SQR(V)
190 PRINT "VARIANCE=";V
200 PRINT "ENTRIES =";G-1
210 PRINTER IS * @ GOTO 40
220 INPUT "CELL WIDTH? ",'1';W @ W=ABS(I
NT(W)) @ DELAY 2,.1 @ DISP "CALCULATING
CELLS"
225 IF W=0 THEN BEEP @ DISP "WIDTH=0 IS
RIDICULOUS" @ GOTO 220
230 Z=INT(X(J)+1)-INT(X(2)) @ Y=Z/W @ IF
 MOD(Z,W)<>0 THEN Y=Y+1
235 IF Y>28 THEN BEEP @ DISP "TOO MANY C
ELLS!" @ GOTO 220
240 FOR I=0 TO Y-1 @ F(I)=0 @ NEXT I
250 FOR I=0 TO Y-1
260 FOR Q=2 TO J
270 R=INT(X(2))
280 IF X(Q)>=R+I*W AND X(Q)<R+(I+1)*W TH
EN F(I)=F(I)+1
310 NEXT Q @ NEXT I
320 A=0 @ M=0
330 FOR I=0 TO Y
340 IF A<F(I) THEN A=F(I)
350 M=M+F(I)
360 NEXT I
370 D=A
380 GOTO 40
410 DISP "WATCH THE PRINTER!" @ PRINTER
IS :PRINTER
420 PRINT CHR$(12) @ PRINT
430 PRINT "                - - * * - -   H
I S T O G R A M   - - * * - -"
440 PRINT
450 U1=0 @ GOSUB 640 @ N=0 @ N2=(56-Y)/Y
460 FOR I=Y TO 1 STEP -1
465 S$=STR$(INT(X(J)+1)-N*W) @ N=N+1 @ F
OR L=LEN(S$)+1 TO 11 @ S$=S$&" " @ NEXT
L
470 S$=S$&"I" @ PRINT S$; @ S$=""
475 IF I>0 THEN U=INT(60*F(I-1)/D)
480 IF U<U1 THEN U2=U1 ELSE U2=U
485 U1=U
490 GOSUB 620
500 FOR N1=1 TO N2 @ PRINT "            I
"; @ GOSUB 650 @ NEXT N1
510 NEXT I
520 S$=STR$(INT(X(2))) @ FOR L=LEN(S$)+1
 TO 11 @ S$=S$&" " @ NEXT L
525 S$=S$&"I" @ PRINT S$; @ S$="" @ U2=U
1 @ GOSUB 620
530 U1=0 @ GOSUB 640
540 FIX 0 @ PRINT "             0%";
550 FOR I=2 TO 10 STEP 2 @ G$=STR$(INT(D
/M*I*10+.5)) @ G$=G$[1,LEN(G$)-1]
560 IF LEN(G$)<2 THEN G$="0"&G$
570 G$=G$&"%" @ PRINT "          ";G$;
580 NEXT I @ PRINT
600 PRINT @ GOSUB 700 @ PRINT "MAX FREQ.
 =";D;"    CELL WIDTH=";W
610 PRINT CHR$(12) @ PRINTER IS * @ GOTO
 40
620 FOR K=1 TO U2 @ S$=S$&"-" @ NEXT K @
 PRINT S$ @ S$=""
630 RETURN
640 PRINT "            I";
650 J1=1
660 FOR I1=2 TO 10 STEP 2
665 IF J1=U1 THEN PRINT "I"; @ GOTO 680
670 IF J1<6*I1 THEN PRINT " "; @ GOTO 68
0
675 IF J1=6*I1 THEN PRINT ":";
680 J1=J1+1 @ IF J1>6*I1 THEN 690 ELSE 6
65
690 NEXT I1 @ PRINT @ RETURN
700 FOR I=1 TO 20 @ PRINT " "; @ NEXT I
@ RETURN
710 GOSUB 930 @ DELAY .05,.05 @ PRINT @
PRINT
715 DELAY 2,.1 @ PRINT "    CELL DATA" @
 DELAY .05,.05 @ PRINT
720 DELAY 8,.5 @ FOR I=0 TO Y-1 @ FIX 0
@ G$=STR$(I+1) @ G$=G$[1,LEN(G$)-1]
725 PRINT "CELL  ";G$;" =";F(I);R+I*W @
 NEXT I
730 DELAY .05,.05 @ PRINT @ PRINT @ DELA
Y 8,.5
740 PRINT "RAW DATA LISTING" @ DELAY .05
,.05 @ PRINT @ DELAY 8,.5
750 FOR I=2 TO G @ FIX 0 @ G$=STR$(I-1)
@ G$=G$[1,LEN(G$)-1]
760 PRINT "DATA PT. # ";G$; @ FIX 2 @ PR
INT " =";X(I) @ NEXT I
770 GOTO 40
780 DELAY 8,.5 @ DISP "ENTER UP TO 255 D
ATA" @ DISP "POINTS. THEN SELECT"
790 DISP "'REVIEW STATS' BEFORE" @ DISP
"YOU ASK TO 'PRINT"
800 DISP "HISTOGRAM'. USE 'REVIEW" @ DIS
P "DATA' OPTION LAST."
810 DISP "YOU CAN ADD MORE DATA" @ DISP
"BY SELECTING THE"
820 DISP "'ENTER DATA' OPTION" @ DISP "W
HENEVER DESIRED."
830 GOTO 40
930 INPUT "USE PRINTER (Y/N)? ",'N';Z$
940 IF UPRC$(Z$[1,1])="N" THEN PRINTER I
S * ELSE PRINTER IS :PRINTER
950 RETURN
```

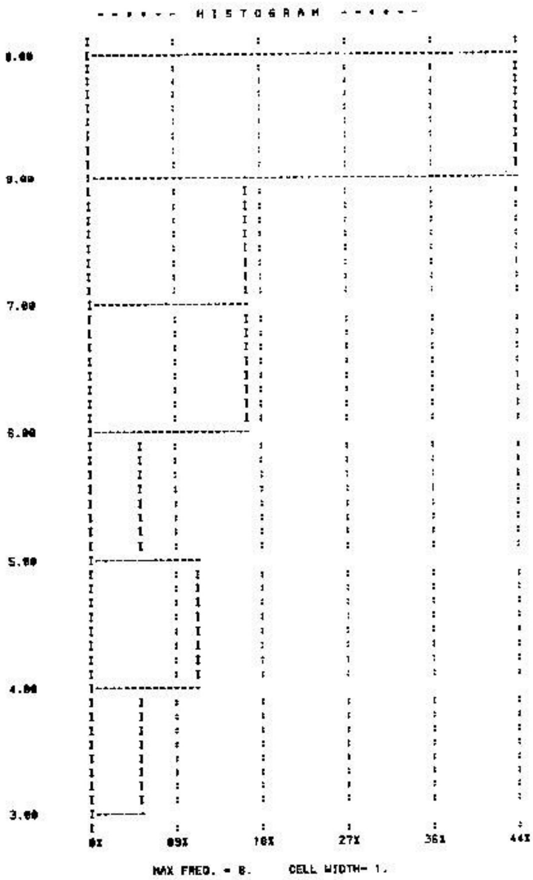
```
STATISTICS
DATA MIN= 3.90
DATA MAX= 8.50
MEAN     = 6.96
STD DEV  = 1.51
VARIANCE= 2.28
ENTRIES = 18.00


    CELL DATA

CELL   1 = 1.  3.
CELL   2 = 2.  4.
CELL   3 = 1.  5.
CELL   4 = 3.  6.
CELL   5 = 3.  7.
CELL   6 = 8.  8.


RAW DATA LISTING

DATA PT. # 1 = 3.90
DATA PT. # 2 = 4.20
DATA PT. # 3 = 4.40
DATA PT. # 4 = 5.70
DATA PT. # 5 = 6.30
DATA PT. # 6 = 6.40
DATA PT. # 7 = 6.60
DATA PT. # 8 = 7.10
DATA PT. # 9 = 7.70
DATA PT. # 10 = 7.90
DATA PT. # 11 = 8.00
DATA PT. # 12 = 8.00
DATA PT. # 13 = 8.10
DATA PT. # 14 = 8.10
DATA PT. # 15 = 8.10
DATA PT. # 16 = 8.10
DATA PT. # 17 = 8.20
DATA PT. # 18 = 8.50
```



deviation, variance, and number of data points. The data can then be organized by sorting the data into equally spaced zones called cells (by selecting the CALCULATE CELLS option). You define the "width" of these cells or bands. The computer counts the number of data points that fall into each cell. You can have this information diagrammed as a histogram by selecting the PRINT HISTOGRAM option, provided you have a Thinkjet or similar printer connected via an HP-IL interface. If you do not have a printer, use the REVIEW DATA option to have the cell information shown on the display. There is even a HELP option that provides a summary of how to proceed, for first-time users of the program.

Note that you should proceed in the fashion: enter data, review statistics, and calculate cells before attempting to obtain a histogram printout or review the histogram data.

A nice feature of the program is that you can put in additional data points at any time by re-selecting the ENTER DATA option! You can have a maximum of 255 data points (provided you have enough RAM available in your system). Be sure, however, to redo the statistics and cell calculations, if you do add in new data points!

The accompanying program listing includes a small sample data base you can use for checking purposes and a reduced illustration of how the histogram will appear.

This program was adapted to the HP-71B by drawing upon a version for the Sharp PC-1500 crafted by *Russ Doughty*.

FOR PC-1500 & PC-2 USERS

VERSATILE ROOT FINDER PROGRAM

Rick Wenger, 6221 -18th Avenue, Kenosha, WI 53140, usually concentrates on the PC-1250/1350 models. However, the program he provides here is so useful and general-purpose, that he has provided versions for the PC-1500/PC-1500A as well as the PC-1250/60/1350 series.

This short program emulates the "SOLVE" function on some Hewlett-Packard top of the line programmable calculators. It is so useful that many mathematicians, scientists, engineers and students may will likely find it worthwhile to keep it handy in memory all the time.

To use the program, you define F as a function of X using any of the line numbers from 100 through 198. This function *must* assign a unique value to F for any X within the valid range of the function.

Here are a few examples of such functions:

1. F=(X*X-2)*X-5
2. F=((((8*X-75)*X+275)*X-485)*X+387)*X-90
3. F=X*SIN(1/X)
4. F=((2*X-11)*X+16)*X-7
5. F=SQR(ABS(X))

Functions 1, 2 and 4 above are polynomials that have been written in nested form to facilitate speed and accuracy during computation. Function 1 is included in the list for historical reasons. It was the original equation used to illustrate Newton's method of approximating real roots. It has one real root at 2.094551482. Equation 2 is a fifth degree polynomial with three real roots at 0.375, 2 and 3. Fifth degree polynomials can not, in general, be solved by algebraic means. Some sort of iterative process is required.

Equation 3 is a tough function for most root-finding programs. The algorithm used in this program, however, will handle it nicely, provided the user specifies some appropriate initial guesses. The function is continuous everywhere except at X=0. It has an infinite number of solutions in any interval which contains 0. When in the degree mode, 1/(180*K) is a solution for K=1,2,3... to infinity!

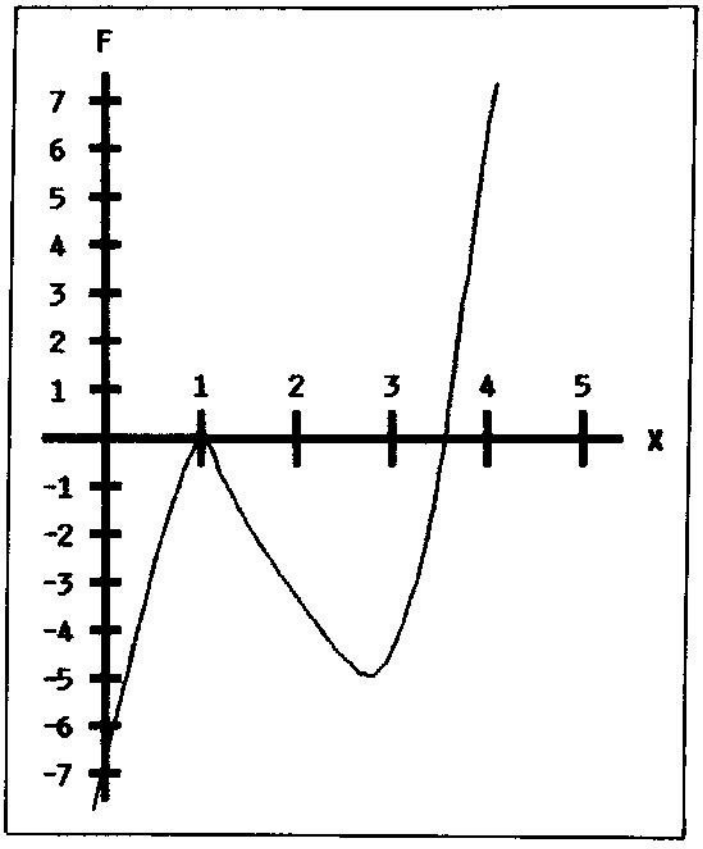
Equation 4 will be used to demonstrate some unique features of the algorithm used in this program. To try it out, enter the equation as a statement within the program having a line number between 100 and 198 (it is included as line 100 in the accompanying program listing). Switch your PC to the RUN mode and use the program to evaluate the function for several values of X. Your results should generally agree with the accompanying illustrative graph.

This root finder program combines the certainty of the bisection method with the ultimate speed of the secant method (a.k.a. Regula Falsi) for smooth functions. The program does not require that the two initial guesses bracket the root. (Indeed, this is not always recommended.) However, if a root can be bracketed (with F(x1) opposite in sign to F(x2)), the program will always converge in the region of the desired root. Accuracy is typically the best that is attainable using floating point firmware.

For example, when evaluating equation 4, the starting guesses X=5 and X=1.5 will cause the root finder to converge on the root X=3.5 after 9 iterations. The bisection method would have required 32. The secant method takes 24 iterations to arrive at the "wrong" root at X=1.

It is possible to use this program to find the root at X=1 as well, if desired. However, since the X-axis is tangential to the curve at this juncture, it is better not to bracket the root. Instead, choose two guesses to one side of the root such as 0.5 and 0.75. Otherwise the program may converge on the already calculated root of X=3.5. It will be observed

Graph *F(x)=2X^3-11X^2+16X-7.*



that a loss of accuracy occurs because of the flatness of the curve in the vicinity of X=1. Any X in the range plus/minus 1E-5 will yield F(x)=0.

It should also be noted that there is no such thing as a foolproof root-solving program. As a direct consequence of the flexibility and speed afforded by the algorithm used in this program, false roots can occur in certain rare instances. As an example of this, try the guesses 0.51835 and 1.5 on the equation labeled number 2 in the list of example functions.

Equation number 5 is given as an example of a function that the program does not help solve. Notice that the function lies completely on one side of the X-axis and that F'(x) is undefined at X=0, which is where the root resides!

A few features worth noting have been incorporated into the program to provide convenience in its operation. You may elect not to make any guesses, in which case the default inputs will be X=0 and X=1. Alternately, you may make just a single guess, or set the last 2 guesses equal to one another without disturbing the program.

The variable C has been set aside as a function evaluation counter. This, incidentally, is the proper criterion for judging the efficiency of any equation solver. Newton's method, for example, often requires few iterations to converge, but is not a good choice for a general root-finder. This is because three function evaluations are required per iteration in order to obtain each new estimate of the root. Newton's method also has problems with convergence in many instances.

There are two program listings shown. PC-1250/60 users should eliminate the WAIT statements in lines 10 and 50 of the PC-1250/60/1350 version. Users who want to watch a function as it converges can add a line such as: 198 PAUSE X,F to the program. Enjoy!

Program *Root-Finder for the PC-1500.*

```
 10:CLEAR
 20:INPUT "x? ";Z:
    GOSUB 99:PRINT
    "f(x)= ";F:
    GOTO 20
 30:IF (ABS (F-E)<
    =.1*ABS F)OR (
    C<2)LET Z=W+W-
    X+C:GOSUB 99:
    GOTO 30
 40:IF SGN F<>SGN
    ELET V=W:D=E
 50:Z=(E*X-F*W)/(E
    -F):IF Z=X
    PRINT "x= ";X:
    PRINT "f(x)= "
    ;F:GOTO 10
 60:IF (D=0)OR (
    SGN (X-Z)<>SGN
    (V-Z))GOSUB 99
    :IF F<>EGOTO 4
    0
 70:IF D=0GOTO 30
 80:Z=V/2+X/2:
    GOSUB 99:IF
    SGN F=SGN ELET
    W=V:E=D
 90:GOTO 40
 99:W=X:E=F:X=Z:C=
    C+1
100:F=2*X^3-11*X^2
    +16*X-7
199:RETURN
```

Program *Root-Finder for the PC-1250/60/1350.*

```
 10:CLEAR [: WAIT 0]
 20:INPUT "x? ";Z:
    GOSUB 99: PRINT "f(x
    )= ";F: GOTO 20
 30:IF ABS (F-E)<=.1*
    ABS F OR C<2 LET Z=W
    +W-X+C: GOSUB 99:
    GOTO 30
 40:IF SGN F<> SGN E
    LET V=W:D=E
 50:Z=(E*X-F*W)/(E-F):
    IF Z=X PRINT "x= ";X
    [: WAIT] : PRINT "f(x)
    = ";F: GOTO 10
 60:IF D=0 OR SGN (X-Z)<
    > SGN (V-Z) GOSUB 99
    : IF F<>E GOTO 40
 70:IF D=0 GOTO 30
 80:Z=V/2+X/2: GOSUB 99:
    IF SGN F= SGN E LET
    W=V:E=D
 90:GOTO 40
 99:W=X:E=F:X=Z:C=C+1
100:F=2*X^3-11*X^2+16*X-
    7
199:RETURN
```

FOR PC-1350 USERS

PC-1350 MONITOR

Hey, look what happens with teamwork! *Norlin Rober* adapted *Rick Wenger's* PC-1250 Monitor so that it would run on the PC-1350. At the time this originally occurred, Rick did not have a PC-1350 of his own. Now he does. This article he submitted goes a long ways towards introducing newcomers to the use of a monitor program. Rick's address is *6221 - 18th Avenue, Kenosha, WI 53140.*

About This Machine Language Monitor

This program was originally written for a PC-1250 having only 2K of RAM. It has been adapted to the PC-1350 by Norlin Rober. He requested that I write this article that describes its use and application.

Since the original program had to fit in such a small space, there are no elaborate prompts or esoteric, seldom-used functions. It operates in an essentially modeless environment. Most of the user defined keys have dual functions through the convention of preceding defined key sequences by BRK and an argument to the function. It is a dandy little program, however, especially for PC-1350 owners who do not have any RAM card expansion memory. On the other hand, those with the interest should find it easily upgradeable to include capabilities such as word search, block moving, etc.

Note the simple method used to protect the machine language portion of the program. We simply move up the value in the "start of BASIC" pointer and execute NEW. This is good news! The PC-1350 is not as hostile to the machine language programmer as the PC-1250 seems to be.

Loading and Saving the Monitor

To key in the program from the accompanying listings:

1. POKE &6F01,&D0,&65
2. Execute NEW
3. Key in the machine language portion using POKE for the address range &64DA through &64F9.
4. Key in the BASIC part of the program. When through keying in the BASIC portion, MEM should yield 1107.
5. The rest of the machine language for the monitor can be entered using the DEF M capability of the monitor itself.

Once the program has been entered, you can save a copy of it on tape:

1. CSAVE M &640F,&65CE
2. CSAVE

To reload the program from tape in the future:

1. POKE &6F01,&D0,&65
2. Execute NEW
3. CLOAD M
4. CLOAD

To restore the PC-1350 to its normal memory configuration when through using the monitor: CALL 0.

Here is how the PC-1350 Monitor utilizes memory:

&6031-640E: Area for user machine language routines, protected from BASIC (990 bytes).

&640F-65CE: Machine language portion of the Monitor. Includes the area &64FC-FE used for temporary CPU registers and &6500-5F used as a temporary CPA.

&65D0-67DC: BASIC area used by BASIC portion of the Monitor program.

&67DD-6C2F: Remaining BASIC area (1107 bytes) available for BASIC routines to the user.

Fixed variables A, B, C and D are used by this Monitor program and should not be disturbed by user routines.

Monitor Instructions

Monitor operations are invoked by use of DEFined key sequences. That is, pressing of the DEF key followed by one of the character keys shown in the listing below. Most of these operations are preceded by entering a value into the display.

DEF M: Preceded by a 4-digit hexadecimal address (note that this address value is *not* prefixed with the & sign) puts the program into the monitor mode. Thereafter you may move to any address by simply keying a 4-digit address and hitting the ENTER key. Enter without any input increments the monitor address by four. You can change this increment within the range 0 through 9 by keying a single digit followed by ENTER. An increment of 5 is useful when viewing the contents of memory. An increment of 1 is useful when entering machine code into memory. Entering machine code into RAM is accomplished by merely positioning the monitor at the desired starting address, then keying in each 2-digit hexadecimal value (without the & prefix) followed by ENTER.

DEF SPC: Decrements the monitor address by the current increment value.

DEF F: Preceded by BRK and 2 digits, this fills the temporary CPA with the specified hexadecimal value. When not preceded by BRK and 2 digits, it simply moves the monitor to the temporary registers area.

DEF B: Puts a breakpoint at the current monitor address.

DEF D: Deletes just the last breakpoint encountered.

DEF J: Preceded by BRK and 4 digits, causes a jump to the specified hexadecimal address. When not prefixed by BRK and an address value, it causes a jump to the previously entered jump address unless used immediately after DEF D. In the latter case its serves as a continue function. You *must* use DEF J if a breakpoint is present a program that will be executed by the monitor. You *must not* use DEF J if the routine ends with an RTS instruction!

DEF S: Preceded by BRK and 2 digits, it searches for the specified hexadecimal byte starting from the current monitor location. If not preceded by BRK and 2 digits, it resumes the search from the current location for the last specified data byte.

DEF H: Provides a hardcopy of memory from the current monitor location to an external printer. Output begins at the current monitor location. Unless this option is preceded by BRK and 4 digits representing an ending address, printing will continue until the BRK key is pressed or the top of memory (address &FFFF) is reached.

DEF =: Converts the decimal number currently in the display into a 4-digit hexadecimal equivalent.

CALL &6473: This is a pseudo-NEW function. It effectively "news" the BASIC programming area while retaining the Monitor program. It can also be used immediately after an inadvertent regular NEW or a bad "crash" to resurrect the Monitor program from the graveyard!

Putting Monitor Through Its Paces

First, let's become familiar with moving around in memory using the Monitor. Key &5D0 DEF M (note that the term DEF stands for using the DEFine key, not the letters D..E..F!) Remember you do not key in the & prefix when entering values to the monitor program. What you will now see appear on your display is the first four bytes of the BASIC language portion of the Monitor program, in the left-hand portion of the display. Preceding that is the address of the first byte in this 4-byte group. To the far left is the antecedent byte, from address &5CF. You will find, when entering machine code, that it is often helpful to be able to see what you have just done.

You may wish at this point to explore the computer's method of storing BASIC programs. The increment value (for the number of bytes advanced/displayed) is initially set to four. You can revise this within the range 0 to 9 by keying a single digit before hitting the ENTER key. Do *not* enter two digits, however, or you will start modifying the BASIC portion of the Monitor program itself. You can move backwards in memory by using the DEF SPC option.

Another interesting section of memory to explore is within the address range 0000 through 1FFF. This is the area of the so-called "hidden" ROM.

Note that you can move to any new address when in the monitor mode by just keying in a 4-digit address and then pressing the ENTER key.

Try A Simple Program

Since most microprocessors do not have multiplication capabilities in their instruction set, a good first program to develop is a subroutine that accomplishes this task. We will attempt to write such a program using the Monitor to illustrate both code entry and debugging operations. The method demonstrated may be referred to by some as the "brute

Program *BASIC Portion of the PC-1350 Monitor.*

```
1:"Y" CALL &65B3:D=&??
  ??: RETURN
2:GOSUB "Y": POKE C,D:
  C=C+1
3:""B= INT (C/256):
  POKE &64FA,C-256*B,B
  : CALL &6560
4:B=0: INPUT "12 1234:
  12 12 12 12 ";B$:
  ON LEN B$ GOTO 11,2,
  3,6
5:C=C+A: GOTO "
6:"M" AREAD B$: GOSUB
  "Y":C=D: POKE &662A,
  &DF:A=4: GOTO "
7:"J" AREAD B$: IF
  LEN B$=4 GOSUB "Y":B
  = INT (D/256): POKE
  &64B5,B,D-256*B
8:CALL &6499: GOTO 10
9:"F" AREAD B$: IF
  LEN B$=2 GOSUB "Y":B
  = INT (D/256): POKE
  &646B,D: CALL &646A
10:C=&64FC: GOTO "
11:A= VAL B$: GOTO 5
12:" "C=C-A: GOTO "
13:"B" CALL &6447:
  GOTO "
14:"D" CALL &6432:
  GOTO "
15:"=" AREAD D:B= INT (
  D/256): POKE &64FA,D
  -256*B,B: CALL &64DA
  : PRINT B$: GOTO "
16:"S" AREAD B$: IF
  LEN B$=2 GOSUB "Y":
  POKE &6426,D
17:CALL &640F:C= PEEK &
  64FA+256* PEEK &64FB
  : GOTO "
18:"H" AREAD B$: IF
  LEN B$=4 GOSUB "Y":C
  =D
19:A=5: POKE &662A,&E2:
  GOTO "
```

force" technique. The operands will be restricted to the range 00 - FF with the product thus limited to the range 0000 through FFFF hexadecimal.

The first step we will perform (following a top-down structure) is to write a BASIC routine to handle I/O for the user and for calling the machine language instructions. The machine language code will perform the actual multiplication. Here is the BASIC code for our little adventure:

```
100 "MULTIPLY"INPUT "MULTIPLICAND (0-255): ";N:POKE
    66031,N
110 INPUT "MULTIPLIER (0-255): ";M:POKE 66032,M
120 CALL 66100
130 PRINT "PRODUCT= ";PEEK 66033+256*PEEK 66034
140 GOTO 100
```

The structure of the BASIC routine shows that certain details of the upcoming machine language routine have already been determined. Thus, the starting address of the machine code must be at 66100. We have designated locations 66031 and 66032 as being the input (multiplicand and multiplier) values. Locations 66033 + 66034 will hold the result of the multiplication process as a 2-register, 16-bit value. It is important to remember when working with this CPU that it is generally more convenient to have the low order byte of a 16-bit value precede the high byte when stored in memory.

Using this information, here is a machine language routine that might seen to do the job:

Program *Machine Language Parts of the PC-1350 Monitor.*

```
640F:10 64 FA 84
6413:1A 00 00 04
6417:88 55 85 63
641B:40 2A 07 82
641F:13 04 0A 88
6423:35 88 63 00
6427:29 12 10 64
642B:FA 84 1B 37
642F:37 37 37 10
6433:64 2F 82 00
6437:02 18 10 00
643B:00 82 19 10
643F:64 3A 1A 85
6443:11 B5 1B 37
6447:10 64 FA 86
644B:1A 84 02 B7
644F:DB 02 64 DA
6453:02 79 82 00
6457:02 07 06 19
645B:10 64 3B 86
645F:53 11 3A 87
6463:53 10 64 2F
6467:82 19 37 02
646B:00 00 60 10
646F:64 FF 1F 37
6473:02 DC 03 67
6477:82 10 6F 03
647B:1B 10 65 D1
647F:D4 00 11 D0
6483:D5 FF 10 67
6487:DC D5 FF 10
648B:6F 01 02 D0
648F:03 65 82 1B
6493:37 37 37 37
6497:37 37 80 00
649B:5F 10 65 00
649F:19 11 02 52
64A3:10 64 FC 57
64A7:30 11 FD 57
64AB:31 11 FE 57
64AF:32 10 65 02
64B3:57 79 00 00
64B7:10 65 02 52
64BB:20 10 64 FC
64BF:52 21 11 FD
64C3:52 22 11 FE
64C7:52 80 10 65
64CB:00 53 11 02
64CF:57 11 01 00
64D3:5E 81 19 02
64D7:58 32 37 84
64DB:10 64 FA 1A
64DF:84 78 65 A1
64E3:82 10 6C F3
64E7:1B 85 78 65
64EB:A1 82 11 F1
64EF:1B 02 F5 11
64F3:F0 52 23 11
64F7:F5 52 37

6560:78 64 DA 88
6564:00 03 10 6C
6568:F1 18 88 10
656C:66 2F 19 85
6570:00 04 63 E0
6574:05 2A 09 82
6578:13 04 0A 88
657C:35 2C 03 88
6580:18 88 78 65
6584:A1 10 66 2C
6588:32 1B 02 66
658C:87 DB 02 33
6590:86 DB 89 00
6594:03 78 65 A1
6598:26 DA 26 06
659C:50 41 2B 0A
65A0:37 59 78 65
65A4:A8 DA 59 58
65A8:64 0F 74 30
65AC:67 3A 3A 03
65B0:74 07 37 00
65B4:04 84 02 30
65B8:1E 88 10 6C
65BC:F1 18 87 50
65C0:63 00 29 04
65C4:51 51 51 51
65C8:10 65 E1 00
65CC:03 19 37
```

```
Addr   Code      Labels   Mnemonics        Comments
6100   90        START    SLP 10           Zero out 10
6101   60 00              AND (P) 00       and 11 to
6103   50                 INC P            hold product.
6104   60 00              AND (P) 00
6106   10 60 31           LDW 6031
6109   57                 LDA (W)          Now load AB
610A   03 00              LDB 0            with multiplicand.
610C   88                 SLP 08           Load (08)
610D   11 32              LDWL 32          with
610F   55                 LD (P)(W)        multiplier.
6110   90                 SLP 10           Next, add
6111   49        LOOP     DE (08)          AB to
6112   3A 04              FCS BYE          (10)-(11)
6114   14                 ADD (P+) AB      a total
6115   20 05              REV LOOP         of (08) times.
6117   10 60 33  BYE      LDW 6033         Offload
611A   90                 SLP 10           product in (10)-(11)
611B   1B                 EX1 (P)(W)       to 6033-34.
611C   37                 RTS              Return to BASIC.
```

To enter the machine code into memory, simply position the monitor to address 6100 and key in each 2-digit hexadecimal code followed by ENTER.

To execute the program, key BRK to exit the monitor. Then issues the command RUN "MULTIPLY". Respond to the prompts to key: 2 ENTER 2 ENTER. (To solve for 2 times 2....) Oops! Did you get a response of 514? Obviously, something is wrong.

Since this is a tutorial discussion, the above program is deliberately incorrect for illustrative reasons. Now to explore further. What could be wrong?

The first thing that might come to mind is that maybe we goofed on an address in the BASIC program. Let's explore that possibility. Key: 6031 DEF M. What do you see there? How about 02 02 02 02? Well, 0202 is indeed the hexadecimal equivalent of 514 decimal. So, it doesn't appear that the problem lies with the BASIC portion of our program.

Onward we snoop. Lets try running the program again for more clues. Key: BRK, RUN "MULTIPLY" and press ENTER. Respond to the prompts with 3 ENTER, 4 ENTER. Check locations 6031 to 6034 using the monitor. What happened now? Does it look interesting?

Since we now know that the problem lies within the machine language routine, we can try setting up a breakpoint to attempt to pinpoint the location and reason for the difficulty. The code from address 6100 to 610F is what might be called "set up" business. It is straightforward in nature. We are getting the stack ready for the real business of multiplication, which gets done by the instructions at addresses 6110 to 6116. Let's enter a breakpoint at address 6110 to see if the "set up" portion is performing as expected.

We set the breakpoint by positioning the monitor to 6110 then keying DEF B. Locations 6110 to 6112 with then be overwritten by a breakpoint instruction. This instruction directs the computer to jump to a monitor routine that will in turn allow us to examine the CPU and CPA registers.

Now key: BRK 6100 DEF J. (Remember, we *must* use DEF J when a breakpoint is in place.) Since 6031 and 6032 still contain 3 and 4 from our previous running of the program, these will remain the input values to the machine language routine. Once you have keyed the BRK 6100 DEF J sequence, the monitor should come back almost immediately, positioned at 64FC. The range 64FC through 64FE is the temporary CPU area. The locations are assigned as follows:

64FC = CPU Register P
64FD = CPU Register P'
64FE = CPU Register SP

In addition, the entire area from 6500 to 655F is dedicated to representing the CPA at breakpoint. Here is a technical explanation of how the temporary CPU area and the temporary CPA area are utilized with DEF J when a breakpoint is in place.

When DEF J is invoked, the temporary CPU register locations 64FC - FE are dumped to the real CPU registers and the temporary CPA is dumped to the real CPA. Control then passes to the specified jump address where instructions are executed until a breakpoint is reached. At this point the temporary CPA and CPU registers are made to reflect the actual CPU and CPA contents. The actual CPA and CPU registers are then stuffed with "safe" values so that the BASIC operating system will not crash. Control then passes to the monitor which points itself to the temporary CPU area.

It seems to take quite a few words to explain what is a relatively simple concept. Really, all we need to worry about now is that the code in our routine between address 6100 and 610F had the desired effect on the CPA registers involved in the operations. Now, let's make a list of what we should be expecting in the CPA registers when we reach the breakpoint we have set.

CPA Register A is at 6502, expected value: 03
CPA Register B is at 6503, expected value: 00
CPA register (08) is at 6508, expected value: 04
CPA register (10) is at 6510, expected value: 00
CPA register (11) is at 6511, expected value: 00

Now, you check those memory locations to see if they contain the expected values. Well, if you are doing the same things we have explained, then they do! This indicates that the first part of our routine seems to be working OK.

Let's delete the breakpoint at 6110. Key DEF D to remove it. Now position the monitor to 6117 and key DEF B to set a new breakpoint at that location. Key DEF J to execute the 6110 to 6117 portion of the machine language routine. (Note that this Monitor program is smart enough to know that since we recently deleted a breakpoint and have not specified a new jump address, it is to continue operations from the old breakpoint location!)

Once you have boldly executed the DEF J option without entering an argument to it, you can check the temporary registers area to see if they are as expected. The first thing you will see is: 64FC = 14. What CPU register does this represent? Is this what you expected it to be? Next check out the A, B, (08), (10) and (11) CPA locations as before.

At this point, a light may begin to dawn. If not, review the explanation under the ADD (P+) AB instruction on page 4 of *PCW* Issue 38. What have we neglected to do?

We have neglected to reposition P to (10) for each pass through the loop! The need for constantly updating the P pointer is a quirk of this particular CPU and a common source of frustration.

The fix is simple. Use the monitor to change location 6116 so that it contains 06 (instead of 05). Then, on each pass through the loop, the program will reverse branch back one more instruction and position P properly.

Before you try out the corrected program, be sure to remove the remaining breakpoint at 6117 by keying DEF D.

Now you can multiply!

If you have stayed this far, then you will find the rest of the monitor options easy. Here are a few more tips to ease your use of the program.

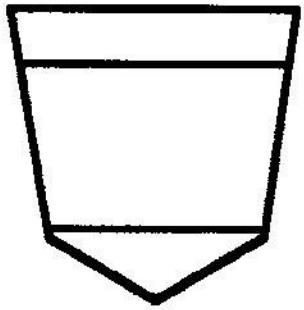
The DEF = option provides decimal to hexadecimal conversion. Key 4321 DEF = and observe 10E1. To see what nn value would be required for a reverse relative branch from 6116 to 6110, key G6116-G6110 DEF =.

DEF F is a convenient way to fill the CPA with any desired value prior to using DEF J.

DEF S uses a machine language routine to search hundreds of times faster than would be possible using BASIC. For example, try positioning the monitor to 6100 then key the sequence: BRK 37 DEF S to find the end point of the multiplication routine.

Enjoy your new programming tool!

POCKET COMPUTER NEWSLETTER



Issue 44 - December

FOR PC-1250/51/60/61 USERS

MACHINE CODE TIMING

While *Rick Wenger, 6221 - 18th Avenue, Kenosha, WI 53140,* was working on decoding the machine language instruction set of the SC61860 CPU, he gleaned the following information.

Every opcode tested out (on a Sharp PC-1250) at a multiple of 5.2 microseconds (1 microsecond = 1 millionth of a second). Thus, that is the time required for one machine cycle. Note that all instructions require more than one such cycle, as detailed in the listing that follows. (*Editor's note: On a PC-1350, the cycle time appears to be approximately 4 microseconds.*)

This timing information can be of value to machine language programmers who want to develop routines that operate in a precise amount of time or who want to work towards achieving minimum program execution times. The number of cycles required by an instruction is given. Multiply this by 5.2 microseconds to find the time required to execute an instruction in a PC-1250/1251.

Load Instructions:

LDA #nn	4
LDA P	2
LDA P'	2
LDA SP	2
LDA (P)	2
LDA (W)	3
LDB #nn	4
LDC0 #nn	4
LDC1 #nn	4
LDP #nn	4
LDP A	2
LDP' nn	4
LDP' A	2
LDSP A	2
LDW #nnnn	8
LDWL #nn	5
LD(P) (W)	3
LD(W) A	2
LD(W) (P)	2

Iterative Load Instructions:

LDDA (Y)	7
LDD(X) A	6
LDIA (Y)	7
LDI(X) A	6
LDO(P) (W)	5+4*C0
LDO(P) A	5+1*C0
LDO(P) HP	11+4*C0
LDO(W) A	4+3*C0
LDI(P) (P')	5+2*C1
LDI(P) (W)	5+4*C1

Short Form Loads:
All SLP instructions require 2 cycles.

Branch Instructions:
All absolute branches require 6 cycles.
All forward and reverse relative branches require 7 cycles if the branch actually takes place. Only 4 cycles are used if the test condition is not met.
The RDR instruction takes 10 cycles if the branch is taken, 7 cycles otherwise.

Exchange Instructions:

EXA B	3
EXA (P)	3
EXO(P) (P')	6+3*C0
EXO(P) (W)	7+6*C0
EX1(P) (P')	6+3*C1
EX1(P) (W)	7+6*C1

Increments And Decrements:

INCA/DEA	4
INCB/DEB	4
INCC0/DEC0	4
INCC1/DEC1	4
INCP/DEP	2
INCX/DEX	6
INCY/DEY	6
INC/DE(08)	4
INC/DE(09)	4
INC/DE(0A)	4
INC/DE(0B)	4

Addition Instructions:

ADDA #nn	4
ADD(P) A	3
ADD(P-) AB	5
ADC(P) A	3
ADD(P) #nn	4
ADEC0(P) A	7+3*C0
ADEC0(P) (P')	7+3*C0

Subtraction Instructions:
Same as corresponding addition instructions.

Rotate Instructions:

RDA	2
RLA	2
RRA	2
RLO (P)	5+1*C0

RRO (P)	5+1*C0
Boolean And Compare Operations:	
ANDA nn	4
ORA nn	4
BITA nn	4
CPA nn	4
AND(P) A	3
OR(P) A	3
BIT(P) A	3
CP(P) A	3
AND(P) nn	4
OR(P) nn	4
BIT(P) nn	4
CP(P) nn	4
AND(W) nn	6
OR(W) nn	6
BIT(W) nn	6
Subroutine Instructions:	
JSR nnnn	8
RTS	4
SS nn	7
I/O Instructions:	
OUTC	3
OUTD	2
OUTE	3
OUTF	2
REFR nn	6+nn
In nn	4
INA PORT	2
INA KBD	2
OSIGNAL	Until done
ONO Signal	Until done
Miscellaneous Directives:	
SET C	2
CLR C	2
PUSH A	3
POP A	2
PUSH-cc-nnnn	9
LNOP	3
SNOP	2
JCA	7 cycles for each condtion checked an not met. 9 when when a condition is met and a branch ensues. 7 for the final branch if none of the conditions are met.

Example JCA Timing:

PUSH-02-CONT	Number of cycles used by JCA:
JCA 01 XXXX	9 if A = 1
02 XXXX	7 + 9 = 16 if A = 2
XXXX	7 + 7 + 6 = 21 if A<>1 and A<>2

CORRECTION AND CLARIFICATION

Issue 42 of *PCW* carried an article that discussed display I/O operations. The top right-hand column on page 4 provided a short listing of a routine that the text said would cause the display to wink. The text also indicated that substituting SNOPs for the first IN 08 directive in that routine would make the winking disappear. These comments concerning the routine are in error.

Indeed, putting SNOPs in place of the first IN 08 instruction simply causes the routine to return immediately to BASIC.

The IN 01 instruction further on in the routine is what *prevents* the display from winking. If you replace that IN 01 directive with two SNOP instructions, then you can observe a pronounced winking of the display.

FOR HEWLETT-PACKARD HP-71B USERS

FINDING FACTORS

Engineers, scientists and mathematicians frequently find it necessary to factor equations. The old trial-and-error method just doesn't compare to having a computer find those prime factors! Here is a program tailored for the HP-71B that can find prime factors, frequently in the blink of an eye. It is an adaptation of a program originally contributed by mathematician *Norlin Rober*.

To put it to use, just load the program and start it up. Respond to the prompt for an integer number to be evaluated.

Program Finding Factors.

```
5 DESTROY ALL @ WIDTH 96 @ DELAY 0,.
5 @ A$=KEY$ @ FIX 0
10 INPUT "INTEGER? ";I @ J=I @ M=0 @
 T=2 @ GOSUB 50 @ T=3 @ GOSUB 50 @ T
=5 @ GOSUB 50 @ T=1
20 T=T+6 @ GOSUB 50 @ T=T+4 @ GOSUB
50 @ T=T+2 @ GOSUB 50 @ T=T+4 @ GOSU
B 50 @ T=T+2 @ GOSUB 50
30 T=T+4 @ GOSUB 50 @ T=T+6 @ GOSUB
50 @ T=T+2 @ GOSUB 50 @ IF T*T<=I TH
EN 20
40 GOTO 100
50 IF J/T-INT(J/T) THEN RETURN
60 IF T=I THEN 130
70 M=M+1 @ J=J/T @ IF J/T=INT(J/T) T
HEN 70
80 BEEP 1500,.15 @ U$=STR$(T) @ U$=U
$[1,LEN(U$)-1] @ DISP U$;" IS A FACT
OR..."
85 V$=STR$(M) @ V$=V$[1,LEN(V$)-1] @
 DISP "OCCURRING ";V$;" TIMES."
90 M=0 @ IF T*T<=J THEN RETURN
100 IF I=J THEN 130
110 IF J=1 THEN DISP "END OF LIST" @
 GOTO 5
120 T=J @ J=1 @ M=1 @ GOTO 80
130 BEEP @ W$=STR$(I) @ W$=W$[1,LEN(
W$)-1] @ DISP W$;" IS A PRIME" @ GOT
O 5
```

The HP-71B will then produce the prime factors along with the number of times each factor would be repeated. The program will handle any integer having up to 10 digits.

Here is an example of its operation so that you can verify proper loading of the program (which has a byte count of 586). Computer responses are in bold type:

INTEGER? 13510750
2 IS A FACTOR...
OCCURRING 1 TIMES.
5 IS A FACTOR...
OCCURRING 3 TIMES.
11 IS A FACTOR...
OCCURRING 1 TIMES.
17 IS A FACTOR...
OCCURRING 3 TIMES.
END OF LIST

The program is pretty fast when extracting relatively low value primes. However, if you test it with the number 9999999967 (the largest prime that the program is designed to handle) you can plan on waiting about 20 minutes. A beep sounds each time a factor is extracted so that you do not have to keep an eye on the display when dealing with large values.

CORRECTIONS TO TRIANGLE PROGRAM

Issue 42 of *PCW* presented a program for the HP-71B that solved triangulation problems. Unfortunately, the program contained several errors that may cause problems when attempting to solve certain cases. To correct the program edit line 410 so that it appears as follows and add line 415 as shown here:

```
410 IF K<>I+3 THEN G=A(K-3) @ H=A(9-I-K) @ A(K)=90
    @ GOTO 420
415 GOTO 430
```

Thanks to *Norman Ishler* for being the first to point out the need for these corrections.

FOR PC-1500 & PC-2 USERS

LEAST-SQUARES POLYNOMIALS

This program will determine the least-squares polynomial, of specified degree, fitting a set of given pairs (x,y).

To illustrate, suppose we wish to fit a third-degree polynomial to the following data:

X	Y
8.0	1.8
8.5	2.1
9.0	1.9
10.0	2.2
10.6	2.8
11.0	3.2
12.0	4.8

Begin program execution with RUN. Enter 7 in response to NO OF PAIRS? Enter the given values of X and Y as called for by prompts. For DEGREE OF POLY? enter 3. To examine all possibilities, respond to further prompts by entering Y for yes.

The resulting output will include the coefficients of the terms of the third-degree least-squares polynomial. For this example, the polynomial (after some rounding of coefficients) is: Y = -24.65 + 9.60X - 1.17X^2 + .05X^3.

A graph of this polynomial is plotted, with dashed lines marking off the area containing values of X from 8 to 12, and of Y from 1.8 to 4.8. A small circle locates each of the pairs (x,y) used as input data, providing a quick glimpse of how well the curve fits the given data. The coefficient of correlation is also given.

The display will then show a prompt requesting a value of X. As an example, enter 10 and press ENTER. The display will show 2.30810458. This is the value of Y obtained using the least-squares polynomial equation derived by the program. Key ENTER to continue with additional values.

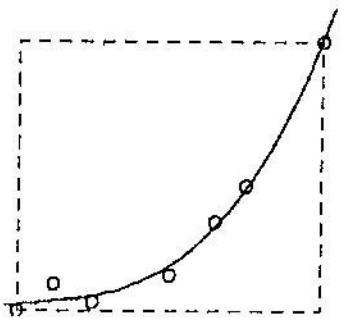
It is possible to find a least-squares polynomial of lower degree, without having to re-enter the initial data. To do this, key DEF N, and enter 2 as the degree. The same options as before will be available for the second-degree polynomial. (Note that it is not possible to specify a degree *higher* than that initially used. Attempting to do so will result in ERROR 9 being displayed.)

Note too, that the polynomial degree may not exceed the number of (x,y) pairs given.

Some Warnings

One should avoid using values of X that are beyond the range

Example *Operation of the Least-Squares Polynomial Program.*



```
INTERVAL FOR X:
         8.00000000
        12.00000000

INTERVAL FOR Y:
         1.80000000
         4.80000000

COEFFICIENTS:
0      -24.64780564
1        9.60166665
2       -1.16610716
3        0.04754995

CORRELATION:
         0.99529243
```

of the X-values included in the given data. Do not assume that a least-squares polynomial will accurately represent the relationship outside the interval of given values. To illustrate, in the example used above the least-squares polynomial should be applied using only values of X that are between 8 and 12.

The temptation to use high-degree polynomials (to obtain a higher correlation) should be resisted, especially when a relatively small number of given (x,y) pairs are used. Although a closer fit to the given data may result, a high-degree polynomial will give overdue attention to small "random" errors in the given data. This is particularly true when the given values of X and Y have been obtained by measurement.

In this program, coefficients are calculated by solving a system of linear equations. This system is often said to be "ill-conditioned," especially when a high degree is specified for the polynomial. (That is, slight errors in the elements of the matrix, caused by roundoff, will result in relatively large errors in the solution.) One partial cure for this problem is to "code" the input data, by subtracting a constant value from each X.

A slight modification to the program will take care of coding. Change line 22 to:

```
22 X=X(I)-H,T=Y(I)
```

and change line 60 to:

```
60 S=X-H
```

To use the coding, a value must first be assigned to the variable H. This should be approximately the mid-value of the given values of X. (In the example used in this article, a suitable choice for H would be a value of 10.)

It is important to note that the resulting polynomial will give the coefficients of powers of (X-H), rather than of X.

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

Program *Least-Squares Polynomials.*

```
 10 INPUT "NO OF PAIRS? ";M:DIM X(M-1),Y(M-1):GOTO 12
 11 GOTO 10
 12 WAIT 0:FOR I=0TO M-1
 13 CLS :PRINT "X(";STR$ (I+1);")";:INPUT "? ";X(I):GOTO 15
 14 GOTO 13
 15 CLS :PRINT "Y(";STR$ (I+1);")";:INPUT "? ";Y(I):NEXT I:WAIT :CLS :GOTO 17
 16 GOTO 15
 17 INPUT "DEGREE OF POLY? ";N:IF M>NTHEN 20
 18 GOTO 17
 20 CLS :DIM A(N),B(2*N),C(N),M(N,N+1)
 21 FOR I=0TO M-1:A=1
 22 S=X(I),T=Y(I)
 23 FOR J=0TO N:B(J)=B(J)+A,C(J)=C(J)+A*T,A=A*S:NEXT J
 24 FOR J=N+1TO N+N:B(J)=B(J)+A:A=A*S:NEXT J:NEXT I:GOTO 26
 25 "N"INPUT "N? ";N
 26 FOR I=0TO N:FOR J=0TO N:M(I,J)=B(I+J):NEXT J:M(I,N+1)=C(I):NEXT I
 27 FOR I=0TO N-1:FOR J=I+1TO N:A=M(J,I)/M(I,I)
 28 FOR K=I+1TO N+1:M(J,K)=M(J,K)-M(I,K)*A:NEXT K:NEXT J:NEXT I
 29 FOR I=NTO 1STEP -1:A(I)=M(I,N+1)/M(I,I)
 30 FOR J=0TO I-1:M(J,N+1)=M(J,N+1)-M(J,I)*A(I):NEXT J:NEXT I:A(0)=M(0,N+1)/M
 31 A=C(0)/M,D=0,E=0:FOR I=0TO M-1
 32 X=X(I):GOSUB 60:B=Y-A,C=Y(I)-A,D=D+B*B,E=E+C*C:NEXT I:R=SQR (D/E)
 40 USING "######.########"
 42 A$="":INPUT "DISPLAY COEFF (Y/N)? ";A$:IF A$="Y"GOSUB 70
 44 A$="":INPUT "PLOT (Y/N)? ";A$:IF A$="Y"CLS :GOSUB 90
 46 A$="":INPUT "PRINT COEFF (Y/N)? ";A$:IF A$="Y"GOSUB 80
 50 INPUT "X? ";X
 52 GOSUB 60:PRINT Y:GOTO 50
 60 S=X
 62 Y=A(N):FOR J=N-1TO 0STEP -1:Y=S*Y+A(J):NEXT J:RETURN
 70 FOR I=0TO N:PRINT "A(";STR$ I;")=";A(I):NEXT I
 72 PRINT "CORRELATION:",R:RETURN
 80 LPRINT "COEFFICIENTS:"
 82 FOR I=0TO N:LPRINT STR$ I;TAB 3;A(I):NEXT I:LPRINT
 84 LPRINT "CORRELATION:",R:RETURN
 90 A=X(0),B=A,C=Y(0),D=C:FOR I=1TO M-1:IF X(I)<ALET A=X(I):GOTO 94
 92 IF X(I)>BLET B=X(I)
 94 IF Y(I)<CLET C=Y(I):GOTO 98
 96 IF Y(I)>DLET D=Y(I)
 98 NEXT I:E=(B-A)/200,F=180/(D-C)
100 GRAPH :GLCURSOR (8,-240):SORGN :RLINE -(200,180),6,,B:POKE &79EA,0
102 P=0:FOR I=-8TO 208STEP 3:X=A+E*I:GOSUB 60:V=(Y-C)*F
104 IF V>240OR V<-60LET P=0:GOTO 110
106 IF PLINE -(I,V):GOTO 110
108 GLCURSOR (I,V):P=1
110 NEXT I
112 GLCURSOR (0,0):A$="":INPUT "PLOT INPUT DATA (Y/N)? ";A$
114 IF A$<>"Y"THEN 118
116 FOR I=0TO M-1:GLCURSOR ((X(I)-A)/E-4,(Y(I)-C)*F-4):LPRINT "o":NEXT I
118 GLCURSOR (0,-100):TEXT :LPRINT "INTERVAL FOR X:":LPRINT A:LPRINT B
120 LPRINT :LPRINT "INTERVAL FOR Y:":LPRINT C:LPRINT D:LPRINT :RETURN
```

FOR PC-1350 USERS

PC-1350 ROM ROUTINES

OK, all you PC-1350 machine language programmers, here is the dope for which you have been waiting. Provided -- once again -- by that super-PC-sleuth: *Norlin Rober, 407 North 1st Avenue, Marshalltown, IA 50158.* Norlin once again has demonstrated his mastery of the realm-of-ROM by locating those key routines that can be so useful to programmers. The article that follows speaks for itself and Norlin's outstanding ability and dedication to sleuthing-out the internal secrets of pocket computers. Thanks, Norlin, for all you have provided over the years to *PCW* readers!

Getting Started

Here are some routines in the ROM of the Sharp PC-1350 that are useful in machine language programming.

The ROM of the PC-1350 exists in two parts. The first part, which is located from 0000 to 1FFF, is the internal or "hidden" ROM, which is not accessible using PEEK. The second part is located between addresses 8000 to FFFF. It is referred to as the external ROM.

It appears (to date) that any revisions in ROM made by Sharp are in the "external" portion. To use the routines beginning at addresses above 8500, you will need to make sure that you have a computer with the same version of ROM as the one to which these addresses apply. Fortunately, it is easy to check this, since there are some checksum routines built into the PC-1350.

To determine the checksum for the external ROM of your PC-1350, execute CALL 6802A. After four or five seconds, the sum will be displayed. If you get anything other than 39380, your version is different from the one to which the routines in this article apply. Thus, any of the routines described in this article that are located above 8500 may not work on your PC.

Floating-Point Arithmetic

Arithmetic operations involving floating-point numbers are carried out in four floating-point registers. These will be referred to as R1 to R4. These registers occupy a portion of the "Central Processing Array" (CPA). Each register consists of 8 bytes. Register R1 uses CPA registers 10 to 17. R2 uses 18 to 1F, R3 uses 20 to 27, and R4 uses 28 through 2F.

The easiest way to get started with floating-point arithmetic is to obtain input data from a "fixed" variable. That is, from a variable named A through Z. The memory map shown in *PCW* Issue 39 (on page 6) shows the locations of these variables. A calculated result may be transferred back to a fixed variable for printing, display or further use. An example later in this article will illustrate such usage.

Here is a list of the beginning addresses of routines that will manipulate data and perform calculations within the CPA floating-point registers:

```
0142  Clear R1
0136  Clear R2
013C  Clear R3
01B6  Copy R2 into R1
01DE  Copy R3 into R1
01C4  Copy R4 into R1
01D2  Copy R1 into R2
01EF  Copy R1 into R3
01FB  Copy R1 into R4
01E3  Copy R2 into R3
019C  Copy R2 into R4
01A8  Copy R4 into R2
020F  Exchange R1 with R2
```

In the routines involving arithmetic and functions, twelve-digit mantissas are calculated and normalized by each routine. The result, however, is not rounded to 10 digits. The carry flag will be set if an overflow or an illegal input argument is encountered. The trigonometric functions and their inverses are calculated according to the current mode setting (DEGREE, RADIAN or GRAD).

```
8AB8  R2 + R1 into R1   (R3, R4 unaffected)
8ACF  R2 - R1 into R1   (R3, R4 unaffected)
8AD9  R2 * R1 into R1   (R4 unaffected)
8AE3  R2 / R1 into R1   (R4 unaffected)
8AEC  R2 ^ R1 into R1
8084  INT(R1) into R1    (R3, R4 unaffected)
8FF4  ABS(R1) into R1    (R2, R3, R4 unaffected)
8856  SGN(R1) into R1    (R2, R3, R4 unaffected)
8809  SQR(R1) into R1    (R4 unaffected)
8AF3  LN(R1)  into R1
8AFB  LOG(R1) into R1
8802  EXP(R1) into R1
89F1  Change sign of R1
8B10  SIN(R1) into R1
8B17  COS(R1) into R1
8B1E  TAN(R1) into R1
8B25  ASN(R1) into R1
8B2C  ACS(R1) into R1
8B33  ATN(R1) into R1
C554  Set DEGREE Mode
C55C  Set RADIAN Mode
C564  Set GRAD Mode
```

An 8-register floating-point stack is also available, for temporary storage of intermediate results. This stack uses the RAM area 7680-768F, although the programmer does not need to be concerned with the location. The routines that utilize this stack are as follows:

```
0FE9  Push R1 onto FP stack. Note that the contents
      of R1 will not be retained.
10C1  Pop R1
10AA  Pop R2
```

Certain constants may be put into R1 or R2, in floating-point format, using the following routines:

```
8C41  Load R1 with 1
8B63  Load R1 with -1
08F0  Load R1 with 12-digit value of Pi
0156  Load R2 with 1
0812  Load R2 with LOG e
081E  Load R2 with 180/Pi
```

There are also routines available for conversion between hexadecimal and decimal. The 2-byte hexadecimal number involved will occupy a portion of the CPA and is designated here as Hexadecimal Register H1. The high-order byte of H1 is located in CPA register 19 and the low-order byte is in CPA register 18. (Note that H1 overlaps a portion of R2. Hence H1 and R2 may not be used simultaneously.)

```
11B0  H1, treated as unsigned binary, converted
      to BCD in R1
11B7  H1, treated as signed binary, converted
      to BCD in R1
163A  R1 converted to unsigned binary, placed
      into H1, carry flag set if out of range
1633  R1 converted to signed binary, placed
      into H1, carry flag set if out of range
```

After a sequence of operations has been performed, the final result should be rounded to 10 digits by calling upon this routine:

```
037A  Round R1 to 10-digit mantissa
```

An Example

This example illustrates how a simple machine language program can be used to perform the equivalent of the BASIC

statement: A = SQR(B*B+C*C).

First it will be necessary to set aside a portion of memory for the machine language program. The beginning of the BASIC area, normally 66030, will be set to 66100. To do this, execute POKE 66F01,0,661. Then, in the PRO mode, execute the NEW command. The area 6030 through 60FF will ther be available as machine language program space.

The accompanying listing shows the instructions needed to perform the calculation mentioned above. The mnemonics are those given in Issue 38 of *PCN*.

The program may be entered into memory using POKEs. You may begin with POKE 66030,0,7,610,66C,6F0,690,618,6E1,...., and so on.

To use the program, first store the values to be used into the variables B and C. Next, execute CALL 66030. The variable A will then contain the result of the calculation. As a test, execute the program with B=6 and C=7. When you display A you should see 9.219544457.

This example is not intended as a particularly practical use of machine language, but rather as an illustration of how the floating-point routines may be put to use.

The BASIC area may be restored to its normal range by execution of CALL 6800F. Note that doing this will also clear all variables and the RESERVE memory.

Program *Machine Language Calculation of A=SQR(B*B+C*C).*

Address	Codes	Mnemonics	
6030	00 07	LDCO 07	Set counter for 8-byte block
6032	10 6C F0	LDW 6CF0	Point W to fixed variable B
6035	90	SLP 10	Point P to R1 (in CPA)
6036	18	LDO (P)(W)	Copy variable B into R1
6037	E1 D2	SS 01D2	Copy R1 into R2
6039	78 8A D9	JSR 8AD9	Multiply
603C	E1 FB	SS 01FB	Save result in R4
603E	00 07	LDCO 07	Set counter for 8-byte block
6040	10 6C E8	LDW 6CE8	Point W to fixed variable C
6043	90	SLP 10	Point P to R1
6044	18	LDO (P)(W)	Copy variable C into R1
6045	E1 D2	SS 01D2	Copy R1 into R2
6047	78 8A D9	JSR 8AD9	Multiply
604A	E1 A8	SS 01A8	Copy R4 into R2
604C	78 8A B8	JSR 8AB8	Add
604F	78 8B 09	JSR 8B09	Calculate square root
6052	E3 7A	SS 037A	Round result to ten digits
6054	10 6C F8	LDW 6CF8	Point W to fixed variable A
6057	90	SLP 10	Point P to R1
6058	00 07	LDCO 07	Set counter for 8-byte block
605A	19	EXO (P)(W)	Exchange R1 with variable A
605B	37	RTS	Return from ML program

More ROM Info

Some other useful operations that can be performed using ROM routines are shown in the accompanying table. Hope it helps you to further enjoy the use of your PC-1350!

Table *More PC-1350 ROM Routines.*

Manipulation of bytes

027D Contents of B and A copied into XH and XL
0282 Contents of B and A copied into YH and YL

115C Contents of Y pushed onto Central Processing Array stack; A and B are lost.
1167 Y is popped from CPA stack; A and B lost.
1175 Block transfer of bytes, in RAM. B = number of bytes moved. Original block starts at address Y+1; bytes are moved into area starting at address X+1. The two areas may overlap only if Y > X.

Calculations (in hexadecimal)

Here U will represent the two-byte hexadecimal number whose high-order byte is in B, low-order byte in A. Also, V represents the two-byte number whose high- and low-order bytes are located, respectively, in CPA addresses ØB and ØA.

1EBE V is replaced by its complement. (This is the equivalent of subtraction from &10000.)
183D The product U * V is calculated and put into V. If an overflow is produced, flag C is set.

Power off

04D8 Computer is turned off. Power comes back on when the ON/BRK key is pressed.

BEEP

C318 The number of BEEPS produced will be whatever number is contained in floating-point register R1.

Display

04AD Display off
04B1 Display on

DC3C Numeric data in R1 is displayed, in default format, provided that Ø1 is stored into address 6F15 immediately preceding execution of this routine.

1E0C Clear Display Output Buffer. (This is the area located 6D00-6D5F.)

1DCF Display contents of Display Output Buffer. The display will show the characters whose ASCII codes are in appropriate locations in the Display Output Buffer, as follows:

Top line of display:	6D00 to 6D17
Second line	6D18 to 6D2F
Third line	6D30 to 6D47
Bottom line	6D48 to 6D5F

Delay

BF3C The length of the delay is the complement of the two-byte number contained in CPA registers 09 and 08, in units of 1/64 second. Use of the BRK key will discontinue the delay.

Keyboard Input

ROM subroutine 120A will wait until a key is pressed. Then A will contain the ASCII code for the key used, including codes that require the SHIFT or SML prefix.

The DEF prefix, however, must be disabled prior to calling subroutine 120A. This is done by storing ØØ into 783C. Immediately following subroutine 120A, 1Ø should be stored into 783C.

The codes obtained when the control keys are used are as follows:

CLS	02	OFF Switch	0A
SHIFT CLS	03	INS	0B
Scroll up	04	DEL	0C
Scroll down	05	ENTER	0D
SHIFT ENTER	06	Cursor right	0E
ON/BRK	07	Cursor left	0F
MODE	08	Shifted cursor left	13
		Shifted cursor right	14

Codes obtained for SHIFTed character keys are as follows:

A	81	H	88	S	F3
B	82	J	8A	V	F6
C	83	K	8B	X	F8
D	84	L	8C	Z	FA
F	86	M	8D	SPC	F1
G	87	N	8E	=	F4

This routine may also be used to accept repeated input of a key when the key is held down. This key-repeat feature is in effect whenever 02 is stored into 6F11 prior to execution of subroutine 120A. (Key repeat is turned off automatically when the key is released.)

CE-126P Printer

DC3C Numeric data in R1 is printed and displayed, in default format, if preceded by storing 01 into 6F15 and 14 into 783C. Follow by storing 10 into 783C.

1C16 Printout pointer (6FAE) is reset to zero. This will assure a fresh start for printing, canceling any leftover data resulting from having interrupted a previous printout with BREAK.

1C59 Clear Printout Buffer. This buffer is located 7480-7497.

1C05 Print contents of Printout Buffer (7480-7497) as one line. Non-character codes will be printed as spaces.

A372 The byte contained in A is transmitted to a buffer located in the CE-126P. The capacity of this buffer is 24 bytes. The contents of this buffer are printed when subroutine A51F is executed.

A51F Print the codes that have been transmitted to the CE-126P by subroutine A372. These codes will be printed right-justified. The entire character set of the CE-126P is usable, including Katakana characters (Codes A1 to DF), exponents (codes 80-8C), and various other symbols.

Index 1985 Articles.

Article	Author	Issue	Page
Announcement: Access Pocketable Portable	PCN, Staff	41	1
Announcement: Motorola Hand-Held Communicates Via Radio Waves	PCN, Staff	39	1
Announcement: Pocket Computer Sports Touch Pad Screen	PCN, Staff	40	1
Announcement: Sharp PC-2500 Portable Notebook Computer	PCN, Staff	42	1
Commentary: Watch Pocket	PCN, Staff	37	8
Commentary: Watch Pocket	PCN, Staff	39	8
Commentary: Watch Pocket	PCN, Staff	40	8
Corrections: See Issue Number 44	Various	44	8,2,3,8
Program: Appointments/Alarms (HP-71B)	PCN, Staff	39	2
Program: Bowling Score Keeper (PC-1500/PC-2)	Sinclck, Robert	42	6
Program: Calendar (HP-71B)	PCN, Staff (conversion)	41	5
Program: Chess Clock (PC-1500/PC-2)	Gibson, John	41	6
Program: CLOAD M? (Verify Machine Language Saves) (PC-1500/PC-2)	Bowman, Eric	40	7
Program: Curve-Fitting (HP-71B)	PCN, Staff (conversion)	40	5
Program: Disassembler (PC-1250/51,PC-3/3A)	Wenger, Rick	41	3
Program: Displaying Graphs (PC-1350)	Rober, Norlin	42	8
Program: Finding Prime Factors (HP-71B)	PCN, Staff (conversion)	44	2
Program: Histogram (HP-71B)	PCN, Staff (conversion)	43	2
Program: Least-Squares Polynomials (PC-1500/PC-2)	Rober, Norlin	44	3
Program: Memory Dump (PC-1250/51/60/61,PC-3/3A)	PCN, Staff (conversion)	37	3
Program: Monitor (PC-1250/51,PC-3/3A)	Wenger, Rick	40	2
Program: Monitor (PC-1350)	Wenger, Rick	43	5
Program: Morse Code (HP-71B)	PCN, Staff (conversion)	38	6
Program: Password Protection (PC1500/PC-2)	Bowman, Eric	38	7
Program: Payroll Deductions (HP-71B)	PCN, Staff (conversion)	37	4
Program: Payroll Deductions (PC-1250/51/60/61,PC-3/3A)	PCN, Staff (conversion)	37	2
Program: PC! Solve (PC-1500/PC-2)	Bowman, Eric	39	7
Program: Root Finder (PC-1500/PC-2,PC-1250-61,PC-3/3A,PC-1350)	Wenger, Rick	43	4
Program: Simultaneous Equations (PC-1250-61,PC-3/3A)	PCN, Staff (conversion)	40	4
Program: Triangles (HP-71B)	PCN, Staff (conversion)	42	5
Review: Sharp PC-1350	PCN, Staff	37	1
Review: Transoft Toolkit	PCN, Staff	38	1
Technical Info: Converting PC-1500 Programs to PC-1250/PC-3	PCN, Staff	40	4
Technical Info: Inside the PC-1350	Rober, Norlin	41	8
Technical Info: Keyboard & Display I/O (PC-1250/PC-3)	Wenger, Rick	42	2
Technical Info: Machine Code Exposed (PC-1250-61,PC-3/3A,1350)	Wenger, Rick	38	2
Technical Info: Machine Code Timing (PC-1250-61,PC-3/3A,PC-1350)	Wenger, Rick	44	1
Technical Info: Manual Addendum (PC-1250/51,PC-3/3A)	Wenger, Rick	43	1
Technical Info: Memory Map (PC-1261)	Rober, Norlin	39	4
Technical Info: Memory Map (PC-1350)	Rober, Norlin	39	5
Technical Info: Miscellaneous Tips (PC-1350)	Rober, Norlin	40	7
Technical Info: More M.L. Instructions (PC-1250-61,PC-3/3A,1350)	Wenger, Rick	41	2
Technical Info: PC-1350 Welcome Aboard	PCN, Staff	37	7
Technical Info: PC-1500/PC-2 Potpourri	PCN, Staff	37	5
Technical Info: Pocket Computer Speed Comparisons	Rober, Norlin	40	6
Technical Info: ROM Routines (PC-1350)	Rober, Norlin	44	5

A FEW MORE CORRECTIONS

The PC-1250 disassembler listing on page 4, Issue 41, of *PCN* is missing a line of code. Add this line to the listing:

C455: 14 1A 19 8B

Also, be advised that the byte of code at address C44A contains 0A.

PC-1350 Monitor Corrections

There are several items in the PC-1350 Monitor article that was presented in Issue 43 of *PCN* that need changing:

First, on page 6, step number 3 in the procedure for keying in the program should read as follows:

"Key in the machine language portion using POKE for the addresses 664DA through 664F9 *and* 66560 through 665CE."

Also, on page 6, keying BRK followed by 4 digits provides a *starting* address argument to DEF M, not an ending address.

In the listing for the Monitor on page 7, change location 6407 to contain 56 (not 58) and change location 6573 to be 20 (instead of E0).

HP-71B Curve-Fitting Corrections

The ends of several lines in the listing of the Curve-Fitting program in Issue 40 of *PCN* are missing. The last statement in each of the lines listed below should read as indicated:

```
50 ... a GOTO 20
65 ... ELSE DISP "3-LOG      4-POWER"
105 .. a IF L<=0 THEN J=1
130 .. a IF J*T=1 THEN RETURN
390 .. a C=R*R
```