MAKING SENSE OF MAKEBIOS

One of the nice things about the newest Heath version of CP/M is the ability to assemble the BIOS for your system, with what you need included and what you don't need left out. The MAKEBIOS program included with CP/M makes assembling the BIOS fairly easy, but the procedure is hard to understand for some, and it is particularly hard to do on 5.25 inch single sided single density drives. This article will present a simple procedure for assembling the BIOS that can be done on two drives of any type without disk swaping, and on one drive with less swaping than with the standard procedure. I assume that you have been reading the REMark series on CP/M and/or the CP/M documentation and have some idea of what the CP/M programs MOVCPM and SYSGEN do.

A New MAKEBIOS.SUB

One of the most confusing things about the BIOS assembly procedure is the SUBMIT file MAKEBIOS.SUB, because it is hard to decide which drives to specify as parameters on the command line, and difficult to figure out what goes on which disk. The following is a replacement SUBMIT file that requires no parameters. Use an editor to enter this file, and call it MAKE.SUB (to set it apart from the original).

```
MAKEBIOS B:1 B:
ASM BIOS.BBZ
REN B:BIOS.HX0=BIOS.HEX
MAKEBIOS B:2 B:
ASM BIOS.BBZ
REN B:BIOS.HX1=BIOS.HEX
STAT B:BIOS.ASM $R/W
ERA B:BIOS.ASM
PREL B:BIOS B:
MAKEBIOS B:3 B:
STAT B:BIOS.SYS $DIR
```

Assembling the BIOS

To assemble the BIOS, you will need to prepare two disks. The first is a bootable system disk containing the following files: ASM.COM, MAKEBIOS.COM, our MAKE.SUB, PIP.COM, PREL.COM, STAT.COM, and SUBMIT.COM. Any other files that you may need later, such as SYSGEN.COM, can also be put on the disk, as long as there is at least 1k free space on the disk. Enter STAT to see how much space there is. The easiest way to make this disk is to DUPlicate your regular system disk, delete any files not needed to make room, and PIP the files you do need to it. If you have only one drive, use FORMAT to make a new disk and SYSGEN to make it a system disk, then copy the needed files to it. The second disk you will need should have only BIOS.ASM on it. Do not use your original BIOS.ASM disk because BIOS.ASM will be deleted during the procedure. This disk does not need to be bootable.

After you have prepared the two disks, boot up on the first one and place the second one in drive B:. If you have only one drive, you will be instructed to insert the second disk (disk B) in drive A: when it is needed, and to replace the first disk when it is needed. To start the procedure, enter SUBMIT MAKE and hit RETURN. If you have two drives, you can go take a coffee break at this point because the procedure will take a while. Single drive users will have to stay by the computer to swap disks. When the procedure is finished, the new BIOS.SYS will be present on disk B, ready for installation.

Installing the New BIOS

The first disk you should install the new BIOS on is the system disk you used for assembling it, to test it. First, copy MOVCPMnn (use the appropriate MOVCPM program, such as MOVCPM17.COM for an H17 disk), SYSGEN.COM, CONFIGUR.COM, and FORMAT.COM to the disk. Delete any files used in the first step (except STAT or PIP) to make room. Boot up on the disk and delete the old BIOS.SYS with the following commands:

```
STAT BIOS.SYS $R/W
ERA BIOS.SYS
```

Then copy the new BIOS.SYS (from disk B) to the system disk and enter

```
MOVCPMnn * A:
```

Again, you would actually enter MOVCPM17 for an H17 disk, etc. When MOVCPM is finished, run SYSGEN. When it asks for a source drive, hit RETURN, and when it asks for a destination, type A and then hit RETURN. When it prompts for a destination again, reset the system and re-boot. CONFIGUR will run automatically as it always does when you make a new system with MOVCPM.

You can use this disk to re-SYSGEN your other system disks if you wish. Place the disk you wish to SYSGEN in drive B: (if you have two drives) and enter SYSGEN. When it asks for a source, enter A and hit RETURN. When it asks "COPY BIOS.SYS?", type Y. When it asks for a destination, enter B if you have two drives, and hit RETURN. If you have one drive, enter A, insert the disk to be sysgened in the drive, and hit RETURN. You can sysgen as many disks as you wish, and when you are

finished replace the system disk you started with (if you removed it) and hit RETURN in response to the destination prompt. Keep in mind that the new BIOS will be larger than the old one if you re-assembled it for two drive types, so you must have space for it on the disk you are sysgening. Use STAT BIOS.SYS to see how big it is, and STAT B:BIOS.SYS to see how big the old one on other disks is.

Since the purpose of re-assembling the BIOS is usually to allow a new type of drive to be used, you probably want to make a system disk for the new drive. For example, suppose you had an H89 with one H17 type (hard sector) drive and you added the H37 controller and one or more new drives. Let's say that up to this point you have been using CP/M on the hard sector drive and you want to make a soft sector system disk. Use the system disk that you used to make the new BIOS. Boot up on it, insert a new disk into the soft sector drive and FORMAT it. Remember that when you boot from a hard sector drive, the soft sector drives are labeled D:, E:, and F:, for units 0, 1, and 2. After the new disk is formatted, run MOVCPM. Use the appropriate MOVCPM for the new drive type (MOVCPM37 in our example). Then run SYSGEN and hit RETURN when it prompts for a source. When it asks for a destination, enter the appropriate drive letter and hit RETURN. Hit RETURN again when it asks for another destination. Then use PIP to copy all of the files (including the BIOS) to the new disk as follows:

```
PIP D:=*.*[R]
```

When this is done, you can reset the system and boot up on the new drive. It can be used to make more system disks of the same type.

PS:

SDUMP Patches

The following patchs are improvements for the SDUMP program on HUG disk 885-1213. The first patch is for those who do not have MAC.COM and cannot re-assemble SDUMP. This patch causes sector patches to be made as soon as you enter WRITE instead of when you access another sector or exit SDUMP. Make the patch with DDT as follows (your entries are in **bold print**).

```
DDT SDUMP.COM
NEXT  PC
I280 0100
-L554
 0554  LHLD 0001
 0557  LXI  D,0027
 055A  DAD  D
 055B  DCX  H
 055C  DCX  H
 055D  DCX  H
 055E  SHLD 0562
 0561  CALL 0000
 0564  JMP  071B
 0567  CALL 0D42
 056A  LDA  11BB
-A557
 0557  LXI D,27
 055A  DAD D
 055B  MVI C,1
 055D  NOP
-^C   (control-C struck)
>SAVE 18 SDUMP.COM
```

Make sure that the data shown by the L command matches that shown above. If it does not, do not make the patch. Our stock is being updated with later versions.

If you have MAC.COM you can make the above patch to the source code and re-assemble. At the label WRTDSK:, make the following change (shown in bold type):

```
WRTDSK: MVI     C,1      ;"DIRECTORY" WRITE
        CALLBIOS DWRITE  ;WRITE BUFFER BACK
        JMP     ENDFIL   ;EXIT
```

This next patch corrects a problem in the buffer input function in MACRO.LIB. In the current version, if you enter something, delete part of it, then hit RETURN, the character count returned reflects the number of characters after the deletions, but the deleted characters are still in the buffer. For example, if you enter

```
=>TRACK 0 SECTOR 11
```

then press DELETE and hit RETURN, SDUMP displays sector 11, not sector 1 as it should. To correct the problem, make the following changes to MACRO.LIB, starting at the label INPUT.

```
INPUT   MACRO   ADDR,BUFLEN
        MVI     C,10
        IF      NOT NUL ADDR
        LXI     D,ADDR  ;;SET BUFFER ADDRE
        ENDIF
        PUSH    D       ;;SAVE BUFFER ADDR
        IF      NOT NUL BUFLEN
        MVI     A,BUFLEN ;;SET BUFFER LENG
        STAX    D
        ELSE
        MVI     A,127
        STAX    D       ;;SET BUFFER DEFAU
        ENDIF
        CALL    BDOS    ;;BDOS ENTRY
        POP     D       ;;RESTORE ADDRESS
        INX     D
        LDAX    D       ;;GET NO. OF CHARS
        INX     D       ;;MOVE TO STRING
        ADD     E       ;;ADD CNT TO ADDR
        MOV     E,A
        JNC     $+4     ;;NO CARRY
        INR     D       ;;ADD CARRY
```

Vectored to page 20

Dear HUG,

We are starting to use MBASIC to organize some of our officer personnel records. Since we are still using a single sided 40 track single drive disk system, we have implemented a method to maximize free space on our program and data disks.

1. SET HDOS STAND-ALONE on all system disks (REMark 12, page 13).

2. SYSGEN a system disk and delete enough unnecessary files to free the 80 sectors required for MBASIC. Copy MBASIC onto this disk.

3. Initialize two disks, labeling one MBASIC PROGRAMS and the other MBASIC DATA FILES. Copy only the following files onto each disk.

   PIP.ABS
   SYSCMD.SYS

4. Using an editor (or HDOS) create the file MBASIC.DOC as follows:

** When instructed to "Please Replace Diskette in Drive SY0:"

Replace the disk in the drive with a disk containing MBASIC. The disk will be mounted, MBASIC loaded, and the drive reset to allow insertion of another disk. **

5. Using SUBMIT (HUG 885-1060) create the .ABS file below:

```
TYPE MBASIC.DOC
RESET SY0:
MBASIC/F:5    [# of files optional]
RESET "SY0:" [Note difference from
              previous RESET]
```

6. Copy the files from steps 4 and 5 to each of the disks created in step 3.

NOTE: In step 5 you must use the HDOS Reset command. The normally preferred "Safe Disk Reset -- R.ABS" (REMark 12) will dismount and remount the same disk, rather than allowing the disk to be changed (when used by SUBMIT).

To use the system:

1. Boot (or reset) to the MBASIC MASTER disk. Run MBASIC.

2. Reset "SY0:", insert MBASIC PROGRAMS disk. LOAD desired program.

3. Reset "SY0:" [our programs do this for us!] and insert MBASIC DATA disk. Run the program.

If for any reason you must return to HDOS while either the -PROGRAM or -DATA disks are mounted, the MBASIC program on each disk will reset the drive, load and reset MBASIC to allow another disk to be inserted in the drive.

Since we use many different programs with the same datasets, this method has been very effective.

Sincerely,

Charles F. Santose
HQ 3/107th ACR
4630 Allen Road
Stow, OH 44224

Dear HUG,

Thanks for the "MBASIC to Machine Code Link Revisited" in REMark 21. I had long been planning to load all my favorite programs from an elegant menu as a mixture of .ABS and .BAS files and this article combined with the "Menu-Driven Demo Program" article of REMark 19 and the "Screen Formatting in BASIC" article of REMark 12 certainly eased the task.

As fully explained in REMark 21, the MBASIC to Machine Code program actually begins the loading of the machine code program with the stack at 042176A. While this is "safe" and usually caused no problems I did find that "SPACE PIRATES" from the Software Toolworks would not execute from a "menu" because of this. The fix was simple...just change the last data entry in line 2170 from &0200 to &0202. This caused all machine code programs to begin at the "normal" USRFWA of 042200A, casued no problems with any other programs that I tried, and best of all made it possible to load "SPACE PIRATES" from my menu.

Truly yours,

Joseph E. McGlone
155 Lynn Drive
Mansfield, OH 44906

Vectored from page 15

```
XRA     A       ;;CLEAR A
STAX    D       ;;TERMINATE ENTRY
ENDM
```

This patch causes the entry to be terminated with a null, which SDUMP will recognize as the end of the entry.

PS:

larger than 32767. In line 2080, we want to multiply the high byte of the string address by 256 and add the low byte to produce the complete address. If the result of the multiplication is greater than 32767, something is flagged internally that causes MBASIC to refuse to assign the value to USR0 (line 2090). But if we AND off the high bit before the multiplication and OR it back afterwards, MBASIC thinks everything is OK.

In line 2110, the file name descriptor to be used by .LINK is a string argument of USR0. MBASIC enters the USR code with the DE registers pointing to three bytes containing the character count and address of the string. Below is an assembly listing of the USR code that shows how this information is used.

```
                        00001   *       THIS ROUTINE IS A USR PROGRAM THAT CAN BE
                        00002   *       INCORPORATED INTO AN HDOS MBASIC PROGRAM TO
                        00003   *       CAUSE A LINK TO A MACHINE LANGUAGE PROGRAM.
                        00004   *       BY G. CHANDLER, HEATH CO.
                        00005
000.040                 00006   .LINK   EQU     40Q
042.200  376 003       00007   MLINK   CPI     3               IS DATA TYPE STRING?
042.202  300           00008           RNZ                     RETURN IF NOT
042.203  353           00009           XCHG                    PUT STRING PARAM ADDR IN HL
042.204  043           00010           INX     H               SKIP OVER STRING LENGTH
042.205  176           00011           MOV     A,M             GET STRING ADDRESS LOW
042.206  043           00012           INX     H
042.207  146           00013           MOV     H,M             GET STRING ADDRESS HIGH
042.210  157           00014           MOV     L,A             HL = STRING ADDRESS
042.211  353           00015           XCHG                    SAVE ADDRESS IN DE
042.212  041 000 000    00016           LXI     H,0
042.215  071           00017           DAD     SP              LOCATE CURRENT STACK
042.216  061 200 042    00018           LXI     SP,42200A       SET STACK TO DEFAULT LOCATION
042.221  345           00019           PUSH    H               SAVE OLD STACK
042.222  353           00020           XCHG                    HL = STRING ADDRESS
042.223  377 040       00021           SCALL   .LINK           TRY TO LINK TO NEW PROGRAM
042.225  341           00022           POP     H               LINK FAILED, GET OLD STACK
042.226  371           00023           SPHL                    SET IT
042.227  311           00024           RET                     AND RETURN TO MBASIC
042.230  000           00025           END     MLINK
```

Note that the program first checks to see if the A register contains the number 3. This is to make sure that the argument to USR0 was a string. Then the address of the string is extracted and saved in DE. Next, the stack pointer is located and its value is saved, and the stack is reset to the default HDOS value, 42200A. The old stack value is pushed onto the new stack, which means that the new program is actually entered with the stack at 042176, but that is still a safe location. The address of the file name is returned to HL, and .LINK is called. If the link is successful, the rest of the code is not used, but if it is not successful the old stack is retrieved and reset and the program returns to MBASIC. NOTE: If you use B H BASIC and want to link to a machine code program, you don't have to bother with the above. To link to SEABATTL.ABS, as in the MBASIC example, use UNFREEZE "SY0:SEABATTL.ABS".

PART TWO -- CP/M

In CP/M there is no .LINK system call for loading and executing one program from another one, but there is a way to simulate it. You can insert a command line into the CCP's (Console Command Processor) command buffer, and then jump directly to the CCP. This method was shown to us by HUG member Marvin Fichter. In assembly, it looks like this:

```
                ;THIS ROUTINE CAN BE USED TO LOAD AND EXECUTE A
                ;MACHINE LANGUAGE PROGRAM FROM ANOTHER ONE.

                LINK:
0000 3A0200             LDA     2               ;GET BIOS PAGE
0003 D616               SUI     16H             ;FIND CCP PAGE
0005 67                 MOV     H,A
0006 2E00               MVI     L,0             ;HL = CCP START
0008 E5                 PUSH    H               ;SAVE IT
0009 EB                 XCHG
000A 1E07               MVI     E,7             ;DE = COMMAND BUFFER
```

```
000C 212500             LXI     H,COMMAND       ;POINT TO COMMAND
000F 0E13               MVI     C,(COMEND-COMMAND) AND 0FFH
0011 7E         LOOP:   MOV     A,M             ;GET A CHARACTER
0012 12                 STAX    D               ;STORE IT
0013 23                 INX     H
0014 13                 INX     D               ;INCREMENT POINTERS
0015 0D                 DCR     C
0016 C21100             JNZ     LOOP            ;LOOP UNTIL FINISHED
0019 E1                 POP     H               ;GET CCP ADDR
001A E5                 PUSH    H               ;SAVE AGAIN
001B 2E88               MVI     L,88H           ;HL = COMMAND POINTER
001D 3608               MVI     M,8             ;SET IT
001F E1                 POP     H               ;GET CCP ADDR
0020 3A0400             LDA     4               ;GET CURRENT DISK
0023 4F                 MOV     C,A             ;PUT IT IN C
0024 E9                 PCHL                    ;JUMP TO CCP

0025 11         COMMAND:DB      (COMEND-COMMAND-2) AND 0FFH
0026 4D42415349         DB      'MBASIC STARTREK',0
0036 =          COMEND: EQU     $

0036                    END
```

The first thing this program does is locate the CCP page address by subtracting 16H from the BIOS (Basic I/O System) page address. This makes this method possibly version dependent, because the size of the CCP may be different for different versions of CP/M. The value 16H may be valid only for Heath/Zenith CP/M version 2.2.02. If you are using Magnolia or DG CP/M, consult your documentation for the size of the CCP and its distance from the BIOS. After the program gets the address of the CCP, it adds 7 to it to get the address of the command buffer. Into this buffer it places the character count of the command, the command line itself, and a trailing zero. Then the program adds 88H to the CCP address to locate the command pointer. It inserts an 8 here indicating that the first command character starts 8 bytes after the start of the CCP. Then the program gets a number representing the current default system disk from location 4 in low memory and puts it in the C register. If we did not do this, CP/M would assume that the default system disk is the hardware system disk. Finally, control is transferred to the CCP, which processes the command line that was inserted and executes the new program.

A machine program can be run from MBASIC in CP/M using the same method. Below is a routine that will do it.

```
2000 '  THESE LINES CAN BE PLACED IN A CP/M MBASIC
2010 '  PROGRAM TO CAUSE IT TO LINK TO A MACHINE
2020 '  LANGUAGE PROGRAM
2030 '
2040 CCP=PEEK(2)-&H16:'                 GET CCP PAGE ADDRESS
2050 BUF=CCP*256+7:'                    GET ADDRESS OF COMMAND BUFFER
2060 COM$="STAT *.*"+CHR$(0):'          THIS IS WHAT WE'RE LINKING TO
2070 POKE BUF,LEN(COM$)-1:'             PUT COMMAND LENGTH INTO BUFFER
2080 FOR I=1 TO LEN(COM$)
2090 POKE BUF+I,ASC(MID$(COM$,I,1)):'   PUT COMMAND INTO BUFFER
2100 NEXT I
2110 PTR=CCP*256+&H88:'                 GET ADDRESS OF COMMAND POINTER
2120 POKE PTR,8:'                       SET IT
2130 IF CCP>127 THEN
     CCP=((127 AND CCP)*256) OR &H8000
     ELSE CCP=CCP*256:'                 GET CCP ADDRESS
2140 DEF USR0=CCP:'                     SET USR0 TO CCP ADDRESS
2150 CCP=USR0(0):'                      EXIT TO CCP
```

Note that the MBASIC routine does not set up the C register with the default system drive as with the assembly program. This means that you must leave the hardware system drive as the default system drive. You can still link to programs on other drives by specifying them in the command line. To run STAT from drive B: in our example, use B:STAT in line 2060. The MBASIC routine follows the assembly routine closely, and does not need more explanation than the comments included. We used the same trick in line 2130 that was used in the HDOS version, in case the CCP address is above 32767. If the attempt to link fails, control is returned to CP/M instead of to the MBASIC program, as with the HDOS version.

CP/M Part III

In last October and November issues of REMark, Issues 21 and 22, we started a series of articles on CP/M for the beginner. This issue will continue with the series and is intended to get you up and running with CP/M.

In the last issue, we discussed the four subsections of CP/M. The Console Command Processor (CCP), which is the interface between the computer and the user. The Basic Input/Output System (BIOS), which handles the input and output operations between the computer and any peripherals. The file management controller of CP/M is called the Basic Disk Operating System (BDOS). And lastly, the Transient Program Area (TPA), which is the area in memory that is reserved for user selected programs.

We briefly touched on the "cold" and "warm" boot procedures. Also, we defined ambiguous and unambiguous file names and file extensions. This brings us to the point where we can

Bootup CP/M.

Let's begin!

First, it must be assumed that you have made the necessary hardware changes, i.e. the Extended Configuration or "Org 0" Modification, as explained in issue 21. You must have purchased either the 5 1/4" or 8" CP/M operating system, as detailed in the same issue. For ease of teaching, we will assume the 5 1/4" system.

Having done these steps, we are ready to turn on the computer and "cold boot" the system.

Turn on your computer and any/all disk drives that may be part of your system. Insert your CP/M Distribution Disk I in your primary drive, which is called drive A:.

For the H-89 users, do a "SYSTEM RESET" by pressing the SHIFT and RESET keys on your keyboard. Type a "B" for "Bootstrap" at the "H:" prompt that appears on the screen. Your system will respond with the word "BOOT". Type a Carriage Return, <CR>, and you will be off and running.

The system will display a message on the screen that will say:

"32 K HEATH/ZENITH CP/M 2.2.XX"

where the "XX" is the current version of the CP/M operating system.

The "cold boot" procedure was pointed out in the diagram that was pictured in issue 22. The "Bootstrap" routine from Track 0, Sector 1 is being read into low RAM and will execute causing the CP/M Monitor to be loaded into High Memory. That will cause the screen to display the first series of messages of the CP/M "First Time" bootup routine. See "Display I". (We'll continue with that shortly.)

For the H-8 users, do a "System Reset" by simultaneously depressing the "0" and "RST/0" keys on the front panel of the H-8, and then press the the number "1" or "4" key, on the front panel, and you will be . . well, almost off and running. This will need some explanation.

Your screen will also display the "32K HEATH/ZENITH CP/M 2.2.XX" message as explained above. At that point, your system will appear to stop. You will need to type the space bar a few times and then the CP/M Monitor will be loaded into High Memory. The screen will display the first series of messages of the CP/M "First Time" bootup routine.

Why is typing the space bar necessary with the H-8? With the H-89, CP/M assumes a default BAUD rate value of 9600 BAUD. (BAUD is the rate at which the CRT (or your screen) can communicate with the microprocessor.) Therefore, the "Bootstrap" routine automatically loads the CP/M Monitor into High Memory. However, with the H-8, CP/M does not assume any value for the BAUD rate of the CRT. This is because there are a couple of possible CRT's available that can run with the H-8. (The H-89 has only one, the built in H-19 terminal.) Thus the need for typing the space bar to determine the BAUD rate of the CRT.

(Just a small note; CP/M, with the H-8, does not know whether the system has the H8-5 or the H8-4 SERIAL I/O card. By typing spaces, the system determines the baud rate and also which SERIAL I/O card is used.)

Let's look more closely at this

"First Time Bootup Routine".

CP/M does some hardware checking, of your entire system, each time it is called upon to do so. This "First Time Bootup Routine" does this checking automatically and displays the results on your screen. This gives you the opportunity to verify that you and CP/M agree on what your hardware system consists of.

The "First Time Bootup Routine" is actually a transient program of CP/M, called CONFIGUR. On your "Distribution Disk I", the first thing that appears on your screen is the results of entering this program, CONFIGUR. So, it is actually CONFIGUR that does the hardware checking. (We will talk about CONFIGUR later in the article.)

Looking at "Display I", we see that the "Bootup Routine" or CONFIGUR starts by indicating the basic but important verification data. It displays the version number and your serial number of the CP/M system that you have purchased. It then lets you know that its purpose is to configure "the CP/M operating system to a particular hardware environment".

CONFIGUR then displays your hardware environment as it determines it to be. If the information given is not what you feel your system consists of, check any connectors for proper connection. Once you agree with CP/M, you are ready to continue.

Your Distribution Disk I, disk "A:", is write protected, which will not allow you to make any changes to it. So, at the "STANDARD SYSTEM (Y OR N)?" question, a "Y" or <CR> is sufficient for a response. This will cause the "A>" prompt to appear on the screen, meaning, you are now at the CP/M command level.

Now that you have reached the command level of CP/M, you are ready to begin the first steps to creating your own SYSTEM disk. Let's look at a brief outline of the next steps that you will need to take.

I. FORMAT a disk
II. MOVCPM5 (create a system image)
III. SYSGEN (write the system image)
IV. PIP *.* (copy the transient files)
V. DUP (option of two disk system)
VI. REBOOT SYSTEM disk
VII. CONFIGUR the SYSTEM disk

After this point, you will be ready to go on and explore CP/M.

First, let's take a more detailed look at how to go about creating a SYSTEM disk.

FORMAT

When you receive a diskette (disk) from a manufacturer of said product, the disk will not be in the proper format for use with the CP/M operating system. That means the disk will not be recognizable by the system and will be rejected as a bad disk.

In order for CP/M to recognize any disk, it needs to be initialized or formatted. The transient program which accomplishes this procedure is called FORMAT. (For HDOS users, this is equivalent to INIT.) FORMAT is also used to reinitialize used CP/M disks.

In order to run this initialization, at the "A>" command prompt, enter "FORMAT" and answer the questions appropriately. The instructions of FORMAT are contained in your CP/M Manual and will not be detailed here. The important thing to remember for single drive users is that you will always be using drive A:. For the others, choose an appropriate drive.

FORMAT "blanks" the disk by placing the byte E5H in each byte location of a sector on a disk. The disk may then be used as a data disk under CP/M or it may be made into a system disk by running MOVCPM5 and SYSGEN.

MOVCPM5

Most HDOS users are probably asking "what is MOVCPM5?" This is the first transient command that does not relate to any commands of HDOS.

Do you recall the first line that CP/M places on the screen when the Distribution Disk I is booted up? "32K HEATH/ZENITH CP/M 2.2.XX". The 32K is not a value that is pulled from the air, nor is it the amount of memory in my computer. My H89 has 48K of memory. (Refer to "Display I".)

CP/M has the ability to be configured for any sized memory computer . . AND it can be reconfigured for any memory size on any particular machine, up to the maximum amount of memory of the system. CP/M is shipped with the system configured at 32K, because most systems have at least 32K of memory.

This means that even though my computer has 48K of memory, CP/M only recognizes 32K, until it is told that the system has more memory. MOVCPM5 (or MOVCPM8 for 8" disks) is the program that reconfigures the CP/M system to a particular memory size.

After you have FORMATted a disk for your new SYSTEM disk, place the Distribution Disk I into A: and type "MOVCPM5" (or MOVCPM8 for an 8" system). MOVCPM5 will determine the amount of memory and construct the new CP/M system "image" for the new memory size and return to the command level.

With MOVCPM5 you have created a new system image IN MEMORY, it has not been stored on the disk. You now need to store this to your SYSTEM disk. To copy the system image from memory to disk (or from disk to disk), CP/M has the transient program

SYSGEN.

SYSGEN copies the system image from a source, be it memory or another disk, to a destination disk.

To do this, enter "SYSGEN" at the A> prompt. The next question will ask for "SOURCE DRIVE NAME (OR RETURN TO SKIP):", enter a RETURN (at this time). For "DESTINATION DRIVE", choose whatever is appropriate for your system and place the FORMATted disk in same. Then enter a RETURN and the new system image will be copied to your formatted disk.

You have just created a SYSGENed disk under CP/M, which is configured for the memory size of your system. SYSGEN does not place or copy any of the transient programs from the Distribution Disk to the SYSTEM disk.

One important note: You will need to reboot the system after the SYSGEN procedure, if you have created a new memory size image. (You will really have no choice, as the system stops.)

There is only one more step to complete in order to finish your SYSTEM disk. Your first SYSTEM disk should contain all the transient programs or "files" of the Distribution Disk I. To copy the files from the Distribution Disk I to your SYSTEM disk, you will need to use the transient utility program called

PIP.

Peripheral Interchange Program (PIP) is the CP/M utility which will copy or move one/any/all files from one disk to another. If your system contains

a printer, PIP will be your media for making a hardcopy of your disk files. This will be one of your most widely used utilities.

PIP is not limited to unambiguous filenames. The wild cards "?" or "*" may be used to transfer any combination of ambiguous filenames. (To any HDOS users, PIP can be considered an "old friend".)

You are now ready to copy the Distribution Disk files to your SYSTEM disk. The following is the command line for invoking PIP for any number of drives, including a single drive:

A>PIP B:=*.*[R]

(The [R] is a "flag" that will allow PIP to copy Read Only (R/O) files.)

For a two or three drive system, you will insert the Distribution Disk I in A: and your SYSTEM disk in B:. Very straight forward. For a single drive system this will need an explanation.

The BIOS of CP/M creates three "logical" disk drives, even when you have only one "physical" drive. The "logical" drives are "mapped" or rerouted to your "physical" drive, drive A:. For example, if you call for "TYPE B:SAMPLE.DOC" at the command prompt A>, CP/M will prompt you to "Put disk B in 5.25 inch drive 0 and press RETURN". Then it will begin displaying the file SAMPLE.DOC on the screen.

When using PIP, this is exactly what happens with a single drive system. CP/M will prompt you to replace your SOURCE and DESTINATION disks at the appropriate times.

Two NOTES at this time:

1) For a single drive system this process is extremely slow and tedious. For each file on the disk, it will take at least two disk swaps (one for the file name and one or more for the file).

2) The DUP utility (which has not been mentioned up to this point) can only copy to "physically" separate disks. It does not "map" to drive A:.

At the conclusion of PIP, you will have completed your SYSTEM disk, which will now be indentical to your Distribution Disk I, with the exception that your SYSTEM disk will be configured for the amount of memory of your computer. The important point is that the SYSTEM disk will contain all the transient programs of the Distribution Disk.

NOTE: The order of execution of FORMAT, MOVCPM5, SYSGEN, and PIP may vary slightly. FORMAT must be run first, however PIP may precede MOVCPM5 and SYSGEN.

This completes the process of creating a SYSTEM disk. No matter how many drives your system has, you will need to know the above steps. However, for computer systems that have more than one drive, FORMAT and PIP can be "bypassed" by using the CP/M transient program

DUP.

DUP is the utility program, which creates a duplicate copy of one disk onto another. The only stipulation is that the disks must be the same size and same density.

DUP, as noted above, will only copy from "physical" drive to "physical" drive. It does not "map" to a "logical" drive. It is for this reason, that single drive systems cannot use DUP, and bypass the aforementioned steps.

To make a copy with DUP, reboot with the Distribution Disk I in A: and a blank disk in B:. (Please note: You do not need to FORMAT a disk, before using DUP.) Enter "DUP" at the command prompt A> and follow through the menu. (For detailed instructions, refer to your CP/M Users Manual.)

At the conclusion of DUP, your SYSTEM disk will be a duplicate of the Distribution Disk I, including the 32K memory image size. Enter "MOVCPM5" to construct a new memory image. Run "SYSGEN" to copy the system image from memory to your SYSTEM disk on B:.

You will now have a SYSTEM disk indentical to the SYSTEM disk described above after using FORMAT, MOVCPM5, SYSGEN, and PIP.

Be sure to make a copy of your CP/M Distribution Disk II. The Distribution Disk II does not need to be a SYSGENed disk, therefore, you need only FORMAT a data disk and PIP the files to the data disk. You can then put your CP/M Distribution Disks away in a safe, cool, dry storage area.

Now that you have completed your SYSTEM disk, let's take a look how to set up the disk for normal operations of CP/M by looking closer at the transient program

CONFIGUR.

Your CP/M Users Manual has the details on the complete list of available menus for CONFIGUR, so we will not go into great detail here. We will point out to you a few of the basic settings, that will help you get started.

As explained earlier, CONFIGUR does the hardware checking when you first bootup on your Distribution Disk I. CONFIGUR is much more than a "hardware checking" utility.

CONFIGUR is the utility that allows the user to customize his system in any number of ways and, after execution, the CP/M BIOS will recognize any hardware environment that is specified. (For HDOS users, this is similar to the device drivers and their subsequent SET commands.) CONFIGUR does the configuring of the BIOS for any/all peripherals.

CONFIGUR may be run from the command level at any time, and can be considered one of the most important transient programs of CP/M. Your system, even a very simply one, has many different settings that can be user customized by reCONFIGURing the CP/M BIOS.

```
32K HEATH/ZENITH CP/M 2.2.XX

HEATH/ZENITH CONFIGURATION PROGRAM
VERSION 2.2.XX
SERIAL NUMBER: YYY-YYYYY

THIS PROGRAM CONFIGURES THE CP/M OPERATING SYSTEM TO A
PARTICULAR HARDWARE ENVIRONMENT.

PLEASE WAIT DURING HARDWARE VERIFICATION...

H/Z89 WITH 48K OF RANDOM ACCESS MEMORY (RAM)
01 MINIFLOPPY DRIVE(S)
CRT BAUD RATE IS  9600
03 ADDITIONAL SERIAL PORTS FOUND


DRIVE A DISK IS WRITE PROTECTED.
MODIFICATIONS WILL NOT BE MADE TO THE DISK FOR THIS CONFIGUR RUN.

STANDARD SYSTEM (Y OR N)? <Y>:_
```

DISPLAY I

```
CP/M CONFIGURATION

A     SET TERMINAL AND PRINTER CHARACTERISTICS
B     SET DISK PARAMETERS
C     CHANGE THE DEFAULT I/O CONFIGURATION
D     AUTOMATIC PROGRAM CONTROL

X     CONFIGURE, MAKING CHANGES TO MEMORY ONLY
Y     CONFIGURE, MAKING CHANGES TO BOTH MEMORY AND DISK
Z     QUIT, MAKING NO CHANGES

SELECTION:
```

DISPLAY II

For example, in "Display I", the output is all upper case characters. CP/M is configured to "force" all lower case to upper case on output. This is just one simple, but important modification that we will change through CONFIGUR.

BAUD rates, PORT values, Nulls after <CR>, Disk Step rate, Error messages, Peripheral Defaults, and Automatic Execution of the Command Line are all options that can be modified through CONFIGUR. Making modifications will give you the opportunity to become more familiar with the menu levels of the CP/M CONFIGUR.

Notice that "Display II" is the main menu of CONFIGUR. From this menu, you can choose an option by selecting the appropriate letter.

NOTE: It is possible to leave CONFIGUR, without making any changes to the BIOS, by entering a "Z". "X" makes the changes in memory, but, upon reboot, no changes will have been made to the BIOS on the disk.

To get started, bootup on your SYSTEM disk, and this time, enter an "N" at the "STANDARD SYSTEM (Y OR N)?" question. That will bring you to the "CP/M CONFIGURATION" or main menu of CONFIGUR as shown in "Display II".

We have three changes that we want to make to the BIOS at this time. 1) We do not want to have lower case letters forced to upper case. 2) When pressing the DELETE key, we want it to "backspace" rather than echo the character. 3) We no longer want to "default" into executing CONFIGUR at bootup.

From the main menu, enter an "A", which will bring up the menu for setting the terminal parameters. At this point, by entering an "F", it will "FORCE OUTPUT TO UPPER CASE ON CRT: FALSE". Then enter an "L" and CONFIGUR will "ECHO ON DELETE: FALSE".

Enter a "Y" as your choice for "FINISHED, MAKE CHANGES AND RETURN TO MAIN MENU". At the main menu, enter a "D" to go to the menu that controls the automatic bootup execution. Enter an "A" and you will set "RUN AUTOMATIC COMMAND LINE ON COLD BOOT : FALSE".

NOTE: When entering a selection, a single keystroke is all that is required. In our examples, the selected letters, "F", "L", and "D" from their respective menus, will change the value of "TRUE" to "FALSE". (The RETURN key will NOT be pressed to complete execution of a selection.)

Now, enter a "Y" to return to the main menu. Enter a "Y" and the changes will be stored in memory and on the disk. Reboot your SYSTEM disk and you are ready to start using your CP/M operating system.

We have come to a point where you will have to begin to use and play with CP/M on your own. This article has covered many points to get you started.

It should be noted again, that these series of articles are not intended to replace the CP/M Users' Manual and Users' Guide. The approach is one in which the steps of operations are explained in the order of proper execution. You will need to refer to your CP/M Users' Manual for details to any of the information that is supplied in this series.

It is not necessary for the next few series of articles to have a set pattern. Each issue will bring more helpful information for aiding you in your use of CP/M.

<TLJ>

```
0177 363A               MVI     M,':'           ;REPLACE WITH COLON
0179 23                 INX     H
017A 23                 INX     H
017B 23                 INX     H               ;MOVE TO SECOND DELIMITER
017C 363A               MVI     M,':'           ;REPLACE WITH COLON
017E E1                 POP     H               ;RESTORE TIME ADDRESS
017F E5                 PUSH    H               ;SAVE AGAIN

                ; CHECK IF ENTRY IS GOOD

0180 0E08               MVI     C,8             ;CHECK 8 CHARACTERS
0182 7E         CHLP    MOV     A,M
0183 FE30               CPI     '0'             ;LESS THAN ZERO?
0185 DAAB01             JC      GETNEW          ;BAD ENTRY
0188 FE3B               CPI     ':'+1           ;MORE THAN ":"?
018A D2AB01             JNC     GETNEW          ;BAD ENTRY
018D 23                 INX     H
018E 0D                 DCR     C               ;DONE?
018F C28201             JNZ     CHLP            ;IF NOT, CONTINUE

                ; SET TIME IN BIOS.SYS

0192 2A0100             LHLD    1               ;GET BIOS ADDRESS
0195 114D07             LXI     D,TIMADR        ;AND TIME ADDRESS
0198 19                 DAD     D               ;FIND TIME IN MEMORY
0199 D1                 POP     D               ;GET TIME ENTERED ADDRESS
019A 0E08               MVI     C,8             ;MOVE 8 CHARACTERS
019C F3                 DI                      ;DISABLE INTERRUPTS FOR MOVE
019D 1A         MOVLP   LDAX    D               ;GET A CHARACTER
019E 77                 MOV     M,A             ;PUT IT IN MEMORY
019F 13                 INX     D
01A0 23                 INX     H               ;INCREMENT POINTERS
01A1 0D                 DCR     C               ;DONE?
01A2 C29D01             JNZ     MOVLP           ;IF NOT, CONTINUE
01A5 FB                 EI                      ;RESTORE INTERRUPTS
01A6 2AD101     EXIT    LHLD    OLDSP           ;GET OLD STACK POINTER
01A9 F9                 SPHL                    ;SET IT
01AA C9                 RET                     ;RETURN TO CP/M
01AB E1         GETNEW  POP     H               ;FIX STACK
01AC C34C01             JMP     GETIME

01AF 0D0A456E74ENTER    DB      0DH,0AH,'Enter current time (HH-MM-SS): ','$'
01D1            OLDSP   DS      2
01D3 0A0030303ABUFFER   DB      10,0,'00:00:00'
02DD =          STACK   EQU     $+100H

01DD                    END
```

PS:

Continuing in CP/M

Last issue of REMark, we began the first in a series of articles on how to use the CP/M operating system. We briefly covered the history and basics of an operating system. It was pointed out that both of the Heath 8-bit computers, the H8 and H89, need the Extended Configuration Option in order to run CP/M. We advised purchasing additional material for learning CP/M because most beginners have found that the documentation supplied with CP/M is difficult to understand. That primarily brings us to where we can begin.

"Getting Started in CP/M".

The first step in starting to use CP/M will be for us to introduce the basics of the CP/M operating system. This must be done before we even turn on the computer. If you understand the basics, then when you sit down at the terminal, you will find your mind clear of uncertainties about the operating system and the functions available to you.

What can CP/M do? As explained in the last issue, CP/M is able to load and allow execution of user programs. It is able to handle all input and output (I/O) to any peripheral devices. Most important to a programmer, is the file management handling of CP/M. That is all CP/M can do, but we will see in a short while, how powerful these three functions actually are.

In order for CP/M to accomplish these three functions, it has been divided into four subsections. The segments consist of the CCP, the BIOS, the BDOS, and the TPA.

Console Command Processor

The Console Command Processor (CCP) is the portion of CP/M which is directly used by the operator of CP/M. This is what you will see on the terminal as you use CP/M. The CCP controls the interface between the operator and the CPU through the "command" mode of operation.

The commands of the CCP are of two levels: the built-in commands and the transient commands. Both levels are used to do the "housekeeping" functions of CP/M, which perform the creating, handling, listing, deleting, loading and running of any programs or files. The built-in or resident commands are permanently part of the CCP and may be accessed at any time from the command mode. The transient commands are those commands that are stored on disk and must be loaded from the disk and executed. The transient commands will be discussed at a later time.

The five built-in or resident commands are:

DIR	List the file names in a directory
TYPE	Type the contents of a file
ERA	Erase or delete a specified file
SAVE	Save what is in memory to a file
REN	Rename a specified file.

These resident commands are interpreted and executed immediately by the Console Command Processor when entered on the terminal.

Also available through the CCP, are the line editing functions it allows as input, while typing command lines. The editing options are executed by certain control-key sequences. These line editing functions are defined in the CP/M manual and will not be listed here.

Basic Input/Output System

The Basic Input/Output System (BIOS) handles all the input/output of the peripheral devices. Thus the BIOS is the only subsection of CP/M that is machine-dependent.

Accessing of the disk drives and other standard peripherals are the operations that the BIOS provides. The BIOS is the portion of CP/M that can be patched for any particular hardware environment. This is the advantage that was mentioned last issue; the ability for CP/M to be adapted to most any micro-system.

When you purchase the CP/M software package from Heath/Zenith, you will find that the BIOS has already been modified to operate on your Heath computer. No change is necessary. However, should you need to make any patches on your own, the CP/M package provides the standard BIOS and gives steps to modifying the BIOS.

In a future issue of REMark, maybe we will be able to examine the BIOS more closely and make a few modifications. As for now, it is good to know that all necessary changes have been made to the BIOS to operate on your Heath system.

Basic Disk Operating System

The Basic Disk Operating System (BDOS), sometimes referred to as the heart of CP/M, is the file management controller. The BDOS operations are completely independent and unseen by the user and/or programmer. Thus the programmer is free to deal with the pertinent matters of his duties.

The BDOS handles the basic disk file operations, such as the reading and writing of a record to/from a disk. It controls the "dynamic" allocation of the disk file construction, i.e. the BDOS will store a file in any available location(s) on the disk and remember where the entire file has been placed, for retrieval upon request.

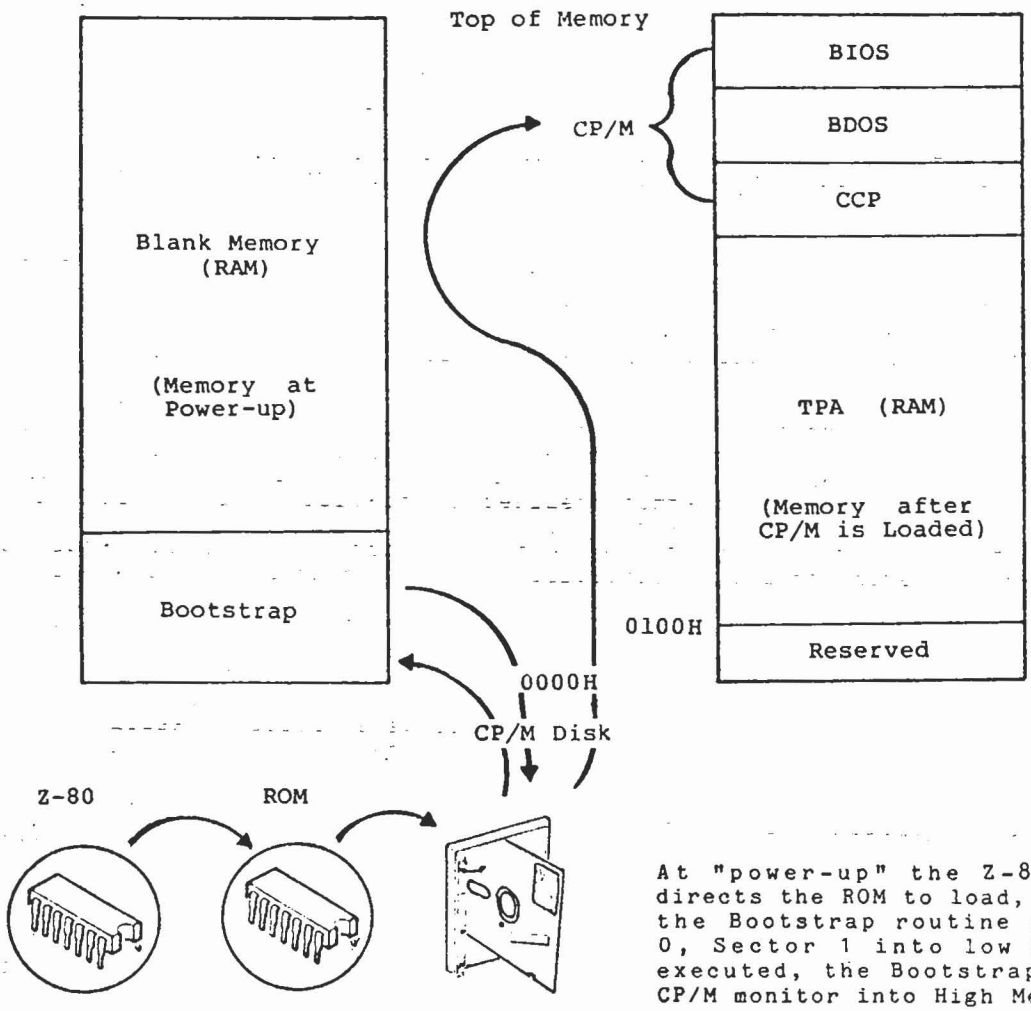

Diagram I

Transient Program Area

The Transient Program Area (TPA) is the segment that is not directly part of CP/M. The TPA is a term used to identify the area in memory, where user programs are stored and executed under CP/M. Any program loaded by CP/M is located at a starting address of 0100H and is stored upward in memory, filling bytes until the program is loaded or the system is out of memory.

With CP/M, when a program is loaded into the TPA, the CCP can be overwritten by the TPA, if the program is of sufficient size. A program will actually write right over the top of the CCP. It is even possible to overwrite the BDOS and BIOS, which means, a program can use virtually all memory for execution (with the exception of the addresses below 0100H). At the conclusion of the program, when control is to be returned to the CCP, it is necessary to reload the CCP by a "warm boot", if just the CCP has been overlapped. A "cold boot" must be done if the BIOS has been overwritten.

A general review, at this point, will show us that the CP/M monitor consists of the CCP, the BIOS, and the BDOS, which are located in high memory at bootup. The TPA is the memory area that user programs are loaded into and executed from. Please note the memory map layout for CP/M, see Diagram I.

Now it is time to move on with other basics of the CP/M operating system.

Bootup Procedure of CP/M

Most of you by now are familiar with the term "Boot". Just as HDOS has a "cold" and "warm" boot, so does CP/M. The bootup procedure is the same as for HDOS. The Monitor or ROM of the 8080 or Z80 (PAM-8 for the H8 and MTR-88 for the H89, respectively), directs the loading of the "bootstrap" program from off the disk into RAM. Then the "bootstrap" loader directs the loading, from disk, of the CP/M monitor into high memory, as indicated in Diagram I.

The process of loading the operating system into memory from powerup is referred to as, doing a "cold boot". When a program has finished and the command is to be transfered back to the CCP, a "warm boot" may be necessary if the program has over-written the CCP. (A "warm boot" may be executed at anytime by entering the control sequence, CTRL-C.)

As stated in our last issue, Heath/Zenith supports either the 5 1/4" or 8" disk versions of CP/M. Drives of either system are identified by the letters of the alphabet. For the CP/M system, the drives are identified by A:, B:, C:, D: and E:, depending on which drive, the 5" or 8", is the bootup drive. In order to bootup on your computer, you will need at least one disk drive. No matter which disk version of CP/M you have, the bootup drive will always be drive A:.

CP/M File Handling

When dealing with the CP/M file handling abilities of the CCP, we must become familiar with some of the definitions. Most of you are already familiar with the file handling of HDOS, CP/M is virtually identical. However, some of us have never been confronted by the terms before.

Filenames of CP/M are referenced by an eight character "filename" and its three character extension or "file type". The filename is generally a brief description of the file it is naming, making it easily recogizable. The file extension in CP/M is used to aid the user and/or programmer in avoiding confusion when identifying the file type. Some of the common CP/M file extensions are:

.ASM	Assembly source code	.BAS	BASIC source code
.COB	COBOL source code	.DAT	ASCII Data file
.FOR	FORTRAN source code	.SUB	SUBMIT command file
.$$$	ED or PIP temporary file		

These file types or extensions are suggested as a standard guide for naming all files. By maintaining the "filename" as a general description and by staying with the standard "file type" format, distinguishing files will be an easy task for yourself and other users.

Unambiguous filenames: Filenames that reference one and only one CP/M file.

Examples: STAT.COM, TEST.ASM, PIP.ASM, ANYTHING.BAS

Wild cards: The characters "*" and "?", which are used to match any character in a particular location of a filename.

Ambiguous filenames: Filenames that reference, by use of wild cards, more than one CP/M file.

Examples: T??T.ASM, ANY?????.BAS, *.BAS, TEST.*

The use of ambiguous files are for directory search and pattern matching, which allows you to reference similiar type files or programs in one command.

Rapping this Up

In the last issue of REMark, I stated, that in this issue, we would explain the set up procedure for CONFIGURing CP/M to your computer. In my continued study of CP/M, I felt that the basic facts and functions shown in this issue needed to be dealt with first. It appears from my vantage point, that we should be able to begin using CP/M in the next issue.

Vectored to Page 29

Vectored from 15

of the .BIN extension file and executes it into machine code. It must be present during run time, and has an overhead of 3 to 8 K bytes, depending on the length and complexity of the program. I do not want to go to deep into this and confuse you, I just wanted you to know basically how it worked.

Any program logic errors must then be corrected by going back to the text editor and again compiling the program before it can be run. Like any compiler, it's a bit more of a hassle than an interpreter such as BASIC, but once the debugged program is finished, the results are well worth it.

Any serious individual wanting to learn Pascal will not learn it by reading this column. I recommend several learning aids. I have the Heath Pascal programming course, which is a good start. Two books I highly recommend are PASCAL PRIMER by David Fox and Mitchell Waite, published by Howard Sams & Co. Inc., and PASCAL, by Paul M. Chirlian, published by Matrix publishers, Portland Oregon. I obtained both of these from my local Heath store. Keep in mind you cannot effectively learn any language by reading. You must set up your system and try the examples and read the books and try some more. Until you begin to write and debug, you will know very little. This column is just to introduce you to the language and give you a push.

Now to get into the meat of things. The steps to produce a good Pascal program are outlined below. Define exactly what you want your program to do. Get straight what will be input and output, and the format. Write out the program flow in English, by breaking your program into sections, each which performs an individual task, each of which can be individually designed and debugged. These will be your modules. Continue to use plain English and break your parts down into sub parts, until each section will be easy to write and debug on its own. This is the basis of structured programming. Now translate the small English modules into Pascal code, debugging each subsection before continuing on to the next. Go through each section mentally and ask yourself--is this what I want it to do? Will this give me the desired result? If not, redo it, or you'll end up doing it over and having to re-compile it later.

Now begin to put all the small pieces together into a main program, calling each module (think of this as a subroutine in BASIC) as you need it. Remember, each variable, constant, or procedure must be defined BEFORE you use it. This keeps things straight and makes you think more about what you are attempting to do.

The general Pascal program has the following format:

```
Program title and declaration of files
used.
Declaration of constants.
Declaration of special variable types.
Declaration of the variables themselves.

Procedure heading I
  Declaration of any local constants or
  variables
  Procedure body
Procedure heading II
   .
   .
   .
Function Heading
  Declaration and function body similar
  to that of procedures

MAIN program body
End
```

Just like BASIC, there are reserved KEYWORDS that cannot be used for anything like variables or constants, but I'll cover these as we use them.

I'll end here for the month. That will give you a chance to get your reading materials, Pascal if you don't have it, and start studying. We'll meet next month and start with some simple programs. See you then.

EOF

Vectored from 14

We will show examples of some of the basic commands of the CCP to get you started in using CP/M for yourself. We will briefly study the system initialization programs, FORMAT, SYSGEN, and MOVCPM, which are part of the transient programs that come in the CP/M software package.

As for this issue, I feel that you should be beginning to feel more comfortable with CP/M, after studying this article. For those of you who are beginning in computers, this article should be of help in understanding the CP/M operating system. For any HDOS users, much of this is "old hat" and needs very little study. If anything, it should be a plain good review of how an operating system works, be it HDOS or CP/M.

<TLJ>

Getting Started with CP/M Part 4

William N. Campbell, M.D.
855 Smithbridge Road
Glen Mills, PA 19342

(Note: Any members of HUG and/or users of Heath/Zenith equipment may copy this material, if desired, for their own use. The reason for the "Copyright" is that I may use this material along with my previous articles in REMark (after converting all to CP/M) in a CP/M and MBASIC vers 5.x "course" in the future. Thank you.)

Terry Jensen has eloquently covered the basics of getting Heath's implementation of CP/M booted and configured and has correctly pointed out that it is extremely important that the user should read the documentation. I should like to point out that (in my opinion) the new user should ONLY read Heath's documentation (which is in the front of the supplied manual) and should totally disregard any and all currently supplied documentation written by Digital Research which makes up the bulk of the manual. The foregoing statement applies only to the newcomer to computers and to the "first time user" of CP/M. It has been pointed out that Digital Research's documentation is intended only for the programmer, and not the end user. It is written in a fashion such that the end user may only become horribly confused. IF you understand and are conversant with CP/M, THEN you may read their documentation. So you ask, "How am I going to learn CP/M?". Heath offers a "CP/M Course" (EC-1120 $99.95). I have no personal experience with this course, although Heath's educational courses are usually excellent. On the other hand, there are currently available four reasonably priced "paperback" guides to CP/M. I have read them all. I strongly suggest that you purchase one of them, and I will indicate my personal preferences.

Osborne CP/M User Guide by Thom Hogan
Publisher: OSBORNE/McGraw-Hill
630 Bancroft Way
Berkeley, CA 94710
Cost: $12.99 plus postage

This book is the most comprehensive "guide" available at this time; my number one recommendation, excellent in all respects. Chapter 8 should be required reading for every actual or potential computer user. This guide is worth twice the cost! Summing it up, Hogan's User Guide is #1!

CP/M Primer by Murtha and Waite
Publisher: Howard W. Sams & Co., Inc.
4300 W. 62nd Street
Indianapolis, IN 46268
Cost: $14.95 plus postage

The above Primer is well written, a good text, but lacks discussion of some important details and has no discussion at all of the "SUBMIT" program usually distributed with CP/M.

Using CP/M by Fernandez and Ashley
Publisher: John Wiley & Sons, Inc.
605 Third Avenue
New York, NY 10158
Cost: $8.95 plus postage

This is a "self-teaching" guide and is very well done. Some may prefer this self-teaching method of presentation.

CP/M Handbook with MP/M by Rodnay Zaks
Publisher: Sybex
2344 Sixth Street
Berkeley, CA 94710
Cost: $13.95 plus postage

I do not recommend Zaks' text as it contains too many errors.

After purchasing one or more of the above, you can sit down at your computer and explore the use of CP/M. There is still one remaining problem, however. CP/M has certain poorly documented flaws. Once you are aware of them, you can usually work around them without much difficulty. In many instances, you won't even notice them because you will give up trying to make a certain CP/M supplied program work as you think it should, and will erroneously have concluded that YOU were doing something incorrectly.

With the above in mind, I will try to help you get started with CP/M and give some illustrations that hopefully will aid you in exploring CP/M. I will go into some detail of certain aspects that are not fully explained elsewhere. Also, I will try to explain things that most confused me when I began using CP/M.

THE MOST IMPORTANT THING I CAN TELL YOU IS TO LEARN THE CP/M OPERATING SYSTEM BEFORE YOU INVESTIGATE MBASIC, ASSEMBLY LANGUAGE PROGRAMING, OR SIMPLY "PLAY GAMES". OTHERWISE YOU **WILL** WASTE MUCH TIME!!

ANOTHER VERY IMPORTANT ITEM IS TO ALWAYS **USE** A DUPLICATE DISK WHEN YOU ARE LEARNING **ANY** OPERATING SYSTEM. IF SOMETHING UN-

FORTUNATE HAPPENS, THEN SIMPLY COPY A NEW DISK FROM THE ORIGINAL AND START OVER.

NOTE THAT THE BEGINNER CAN ONLY LEARN THE CP/M SYSTEM (or any other operating system) BY ACTUALLY SITTING DOWN AT A COMPUTER AND TYPING IN THE VARIOUS COMMANDS, AND TRYING OUT THE VARIOUS PROGRAMS!

SINCE YOU COMMUNICATE WITH THE COMPUTER THROUGH A TYPEWRITER-LIKE KEYBOARD, YOU SHOULD BE A FAIRLY GOOD TOUCH TYPIST. THIS SAVES YOU MUCH TIME. If you need to brush up on your "typing", I recommend "Touch Typist", for CP/M systems, from Newline Software, P.O. Box 402, Littleton MA 01460. Cost $29.95. This is a program that uses your computer to teach you how to "touch type" (or sharpen your skills).

PRELIMINARY CONSIDERATIONS

The features discussed in this article apply to CP/M version 2.0 and above.

You will communicate with your computer by entering certain information and/or commands at the keyboard of your terminal, AND many times you will execute the commands by hitting the Return key on your keyboard. Throughout this article <cr> means hit the Return key. I will repeat this for emphasis. IF YOU SEE "<cr>" IN THIS ARTICLE, IT MEANS YOU HIT THE RETURN KEY ON YOUR KEYBOARD!

CP/M uses A: as name of 1st drive, B: as name of 2nd drive, C: as name of 3rd drive, etc..

Throughout, I assume you are "logged on" drive A: (the drive you booted from), and have the A> prompt showing. If you have more than one drive it is simple to log on to another drive. For example if the A> prompt is displayed, and you wish to move to a second drive (B:) simply type B:<cr>, and there you are with the B> prompt showing. To get back to A:, just type A:<cr> and you are back with the A> showing. (Note that drive B: must contain a disk, and drive A: must always contain a SYSGENed disk.)

If you try any of the examples in this article, you must have on your disk the files required for it. For example, if you are using the PIP command, you MUST have PIP.COM on your disk; if you are using SUBMIT and XSUB programs, then you MUST have SUBMIT.COM and XSUB.COM on your disk.

Many operations require the use of Control characters. They are all performed by holding down the CTRL key, and then ALSO depressing the appropriate desired "character". To do a CTRL C, for example, hold down the CTRL key, and while it is held down, also depress the C key. You may use upper or lower case "characters", incidentally.

WHENEVER YOU ARE RUNNING CP/M, AND FOR SOME REASON REMOVE A DISK FROM A DRIVE AND INSERT ANOTHER DISK, **ALWAYS** DO A "CTRL-C". THIS IS KNOWN AS A WARM BOOT, AND UNLESS YOU DO THIS AFTER SWITCHING A DISK, CP/M WILL NOT KNOW YOU HAVE CHANGED DISKS! Always do this! (Note that if you are in MBASIC always do (or have your program do) a RESET - this will accomplish the same thing.)

FILE NAME AND EXTENSION CONVENTIONS

A complete file name including extension consists of one to eight alphanumeric characters, followed by a period (.), followed by a 1 to 3 character alphanumeric extension (Example: FILENO1.TXT). No spaces are allowed. You do NOT have to include the period (.) or extension (ext.) UNLESS the file is of a special type, or you wish to distinguish it from a file of the same name. Here are some of the special file types used:

.COM	Any machine language program.
.ASM	Any assembly language program you may create using an editor.
.LIB	Any text file you create with an editor for use with ED's "R" command.
.PRN	A "listing" which the assembler creates for you if you want it.
.SUB	A file which you create with an editor for use with SUBMIT (see SUBMIT).
.BAS	This is an extension which the MBASIC interpreter puts on for you if you "SAVE" a program from MBASIC.)

Other than the "special types", most of the files that you create yourself may have, or may not have a period (.) and a 1 to 3 character extension AS YOU DESIRE. For example, here are some valid file names that you might create: THISFILE.TXT, THISFILE.DAT, THISFILE.PGM, THISFILE, FILE1, FILE2, FILE.123, FILE24. All of these files could reside on one disk at the same time, since they are ALL different, one from the other! You will usually create text files using an editor, and programs and/or files using a BASIC language interpreter.

COMMANDS AND COMMONLY USED CP/M UTILITY PROGRAMS

All of these commands and programs are executed from the monitor prompt (A>). Note that our default drive is the A: drive, and you do NOT have to explicitly type A: if the drive involved is the A: drive. If you are logged on to drive B:, then B: is the default that you need not explicitly type, but you must specify all others, including A:. There are times, however, when you do need to type the default drive name. (See below under PIP examples.)

Note also that when we refer to a "file" we usually are referring to "contents of the file", not just the name we (or someone else) has named any given file.

The CTRL-P Toggle

Although this is not a "command" it is a most useful feature of CP/M, if you have a printer! If you do a CTRL-P (hold down the Control key and then, while the Control key is held down, depress the P key) then everything carried out on the screen of your terminal is simultaneously (practically) printed on your printer (echoed)! Doing a second CTRL-P turns off the feature. This is an especially nice feature when you are learning the various CP/M commands since it gives you hard copy of what you did, especially if you made an error. You can see precisely what you typed, and have a permanent record of it. This feature also works nicely in CP/M's editor, ED (although it will not work with many editors). Just be sure and toggle CTRL P on BEFORE you invoke ED (see ED). (This "toggle" does NOT work with SUBMIT or MBASIC, but it does work with BASIC-E (type CTRL-P before you RUN a program). In HUG's editor (885-1210), it works while you are IN the editor.)

DIR

Entering DIR<cr> at the monitor prompt displays the directory of files currently on the diskette, UNLESS a particular file has been set to "SYS" using the STAT command (see STAT), in which case that file will NOT be displayed. Examples:

DIR<cr> displays files on disk in drive A: (default).

DIR B:<cr> displays files on disk in drive B:. If you have only one drive and your Heath/Zenith CP/M is configured for one drive, you will be asked to place disk B: in drive A: and hit RETURN. If you access the system again (type another command), you will be prompted to replace disk A:.

The display invoked with the DIR command simply lists the files and does NOT show the period (.) that actually exists between the file name and its extension (if there is an extension). For example, the file ED.COM is displayed as ED COM.

Note that SOME files may NOT be listed with the DIR command. (ALL files WILL be listed if you execute the STAT *.*<cr> command - see STAT below.)

STAT

This is used in several different ways and displays certain information about your disk files. STAT also allows you to change certain "file attributes". Examples:

STAT<cr> shows the amount of unused disk space on your disk. STAT B:<cr> shows the amount of unused disk space on disk in drive B:.

STAT *.*<cr> shows an alphabetical listing of ALL files on disk in drive A: (if the default is A:). Any file set to "SYS" is enclosed in parentheses. Such a file (if set to SYS) would look like (ED.COM) in the listing, assuming we had set ED.COM to SYS. Any R/O (read only) files are so noted, as are the R/W (read/write) files. STAT B:*.*<cr> does the same thing for drive B:.

STAT filename.ext $R/O<cr> will make filename.ext a "read only" file. This means that you will not be able to "write" that file. For example, if you were editing that file, you would NOT be able to write the edited version to disk using the same file name. In my illustration here "filename.ext" means the exact file name that you are setting to read only. For example, STAT B:THISFILE.TXT $R/O<cr> would set file by the name of THISFILE.TXT which is on disk in drive B: to a read only status.

STAT filename.ext $R/W<cr> is exactly the same as above, except it sets the file to a "read-write" status. Most of your files usually are of this type to begin with. So, the main use of this form of STAT command is to change a preexisting "read-only" file to a "read-write" file.

STAT filename.ext $SYS<cr> will fix the file name so that it does NOT show up when you request a listing of all files with the DIR command. Note that a file set this way can NOT be copied by PIP unless you use PIP's "R" switch which you tack on the end of PIP's command line. The R is enclosed in square brackets, like this: [R].

STAT filename.ext $DIR<cr> will restore the file so that it DOES show up in the listing of the the DIR command.

(Editor's Note: This discussion does not cover all of the uses of STAT, such as displaying disk parameters and displaying and setting I/O device assignments.)

PIP

This is the copy command. PIP means Peripheral Interchange Program.

Note that, in all uses of the PIP command, the original file (the file to be copied) is left unchanged, unless you are using PIP to concatenate several files, and the new "combined" file has same name as one of the "subfiles".

NOTE THAT THE DIRECTION OF THE COPYING IS ALWAYS FROM THE FILE INDICATED TO THE RIGHT OF THE "=", TO THE FILE NAMED ON THE LEFT OF THE "=". In other words, the com-

mand always contains an "=". The file to the left of the "=" is the file that is being CREATED, while the file to the right of the "=" is the file that is being copied. Another way of remembering this: TO = FROM! Of course, you can copy from disk in drive A: to disk in drive B: OR you can copy from disk in drive B: to disk in drive A:. Here are some examples that may clarify this for you.

PIP ABC.TXT=B:XYZ<cr> will copy the contents of file named XYZ FROM disk in drive B: TO the disk in default drive (A:) and give it the name ABC.TXT. Note again that the file on the right side of the "=" in this illustration is copied to the file named on the left side of the "=", e.g. from RIGHT TO LEFT (TO = FROM)! If you have only one drive, you will be prompted to swap disks (if you have Heath/Zenith CP/M) at the appropriate times (see DIR).

PIP B:ABC.TXT=XYZ<cr> copies the contents of a file named XYZ FROM disk in drive A: TO disk in drive B: and gives file the name ABC.TXT.

(You don't have to say "A:" in the above 2 examples, as PIP interprets the ABSENCE of any drive specification as a default, and assumes you mean drive A: which is normally the default drive.)

PIP C:DRIVE3.ABC=B:HOHOHO.XYZ<cr> will copy file HOHOHO.XYZ from disk in drive B: and place it on disk in drive C: and give the copied file the name of DRIVE3.ABC.

PIP NUNAME=OLDFILE<cr> copies file OLDFILE on disk A: (default) to a file called NUNAME on disk A:. The contents of the files are identical. Both files are on the disk after the copy command is carried out.

PIP B:=THISFILE<cr> copies a file called THISFILE from disk in drive A: (or the default drive) to disk in drive B: AND gives it same name as the original file on disk in A:, in this case THISFILE.

PIP A:=B:*.*<cr> copies ALL files (file names and the contents of the named files) from disk in drive B: to disk in drive A:.

PIP BIGFILE=FILE1,FILE2,FILE3<cr> combines (concatenates) FILE1 and FILE2 and FILE3 into ONE file named BIGFILE. In this example, all files are on disk in A:.

PIP LST:=THISFILE<cr> will print out on your printer the contents of a file named THISFILE (FROM disk in drive A: TO your "LST:" device - your printer.)

PIP also has some very useful "switches". "Switches" means that in addition to the fundamental copying that is done, certain other things can also be simultaneously carried out. They are all implemented by typing the "switch", enclosed in square brackets ([]), immediately after the desired PIP command (NO space typed), just before the <cr> which executes the PIP command. Examples:

PIP B:=THISFILE.TXT[V]<cr> will copy THISFILE.TXT from disk in drive A: (default) to disk in drive B: and give it same name, and THEN will compare the two files (V=Verify) and make sure they are identical. The switch used here is the "V" switch.

PIP THISFILE.TXT=B:THISFILE.TXT[L]<cr> copies the file named THISFILE.TXT which resides on disk in B: to a file named THISFILE.TXT on disk in A: (default) AND simultaneously changes any upper case letters in original file to lower case (The new file on drive A: will be entirely lower case). Note that this has nothing to do with the NAMES of the file being in upper or lower case. You can always enter the file names in EITHER upper or lower case when you are typing from the keyboard. Note that nothing is changed in the original file, just in the new file created.

PIP B:UPPER.TXT=LOWER.TXT[U]<cr> copies contents of file named LOWER.TXT on disk in A: to disk in B: and names it UPPER.TXT, AND also changes any lower case letters to upper case in the contents of the new file, while doing so. The contents of the new file UPPER.TXT will be entirely upper case!

PIP LST:=THISFILE[N]<cr> prints the contents of file named THISFILE on your printer and puts a line number in front of each line. Also, try the "N2" switch.

PIP LST:=THISFILE[T8]<cr> prints the contents of the file THISFILE on your printer and expands tabs to spaces. The 8 means that tabs are assumed to be at every 8 columns, which is normal. This switch MUST be used if you PIP files to a printer that cannot handle tabs.

PIP LST:=THISFILE[NT8]<cr> prints THISFILE with both the N and T switches set. Any number of switches can be combined in this way.

There are many more nice switches, or "parameters" that you may wish to investigate. (Editor's Note: I suggest that you read pages 18 through 25 in An Introduction to CP/M Features and Facilities, one of the manuals provided with CP/M, then read pages 8 and 9 in CP/M 2.0 User's Guide for CP/M 1.4 Owners. This is the only way to fully learn the power of PIP, and will introduce you to the "dreaded" Digital Research manuals. -- PS:)

ERA

This command deletes files. Note that you CAN delete a file with the SYS attribute set (see STAT), but that you can NOT erase a file with the R/O (read only) attribute set (see STAT). Examples:

ERA FINISHED.TXT<cr> will erase from disk in drive A:, a file named "FINISHED.TXT".

ERA B:*.*<cr> erases all files on disk in drive B:. This is a MOST DANGEROUS command! Because of this, you are asked if you really want to do it, and must answer Y (for Yes) to complete the command.

The "ERA" command actually removes the file name from the directory, rather than erasing the contents of the file. However, the net result is the same! The contents of the file with its file name removed from the directory is no longer accessible to the user. Note that there are ways to recover from this if you have the necessary knowledge and special programs (such as SDUMP on HUG disk 885-1213).

TYPE

Allows you to display on your terminal screen, the contents of files which contain ASCII characters (alphanumeric characters such as are on your keyboard -- files created with ED, for example). Note that files with the extension .COM are in machine language and only display "gibberish" if you try to TYPE them. (Try it and see.) Example:

TYPE EDITED.TXT<cr> will display on your screen the contents of a file named EDITED.TXT.

Note that here, as elsewhere in CP/M, you may temporarily halt the scrolling of the file by typing a CTRL S (hold down the control key and simultaneously hit the "S" key). This is actually a "toggle". Do it once and scrolling is stopped. Do it again, and scrolling starts again, etc.

REN

This command will rename a file on disk. Note that the file on the right side of the = is renamed to that on the left side of the =. Examples:

REN B:THATFILE=B:THISFILE<cr> The file named THISFILE on disk in drive B: is renamed to THATFILE. The second B: is not required (you could say REN B:THATFILE=THISFILE).

REN NEWNAME=OLDNAME<cr>. File named OLDNAME on disk in drive A: is renamed to NEWNAME.

SAVE

Most of the time the average user will not use this command. He WILL use it if he intends to write and use assembly language code. The use of this command therefore will not be covered here. It is nicely covered in Hogan's book and pages 9 and 10 of An Introduction to CP/M Features and Facilities.

ED

ED is the text editor supplied with CP/M. It is very similar to two editors offered by HUG, one for use with HDOS, and one for use with CP/M. ED is a "character" editor and uses an invisible pointer. "Screen" oriented editors are easier for beginners to use. Nevertheless, I advise getting acquainted with this CP/M supplied editor as there are some things you just can not do with many other editors. If you are familiar with HUG's ED.ABS, then you will have no trouble learning how to use this ED.COM, supplied with CP/M. If you are not familiar with HUG's ED, then I refer you to an article "Getting Started with a Text Editor" which I wrote and which is in REMark issue 11. I used HUG's ED as an example. There are some differences, however, and the following lists what you should do using CP/M's ED.COM. (I assume you have "Getting Started with a Text Editor" handy.) You invoke the editor in exactly the same way. An example:

ED THISFILE.TXT B:<cr>. This command loads ED.COM into memory, and tells ED that we are going to be editing THISFILE.TXT, and that we wish the edited file (when we are through editing) to be placed under same name on drive B:. (ED will do all this for us and when we terminate our editing session, ED will rename the original file to THISFILE.BAK and leave it for us unchanged on disk in drive A:.) Now, the B: in this example is entirely optional and if B: is left off the command line that we used to invoke ED, then ED puts the edited file back on disk in drive A:, but still renames the original file to THISFILE.BAK. This is one difference between ED and the HUG editors, which always put the output file on the same disk as the input file, unless you specify differently.

After you invoke the editor, ED responds with its prompt (*). (ONCE YOU ARE IN ED, YOU WILL SOON LEARN THAT THERE ARE 2 POSSIBLE "MODES" THAT YOU MAY BE IN -- IF YOUR CURSOR IS PRESENT JUST AFTER AN ASTERISK PROMPT (*), THIS MEANS YOU ARE IN ED'S COMMAND MODE. YOU ENTER ALL OF ED'S SINGLE LETTER COMMANDS IN THIS COMMAND MODE. THE OTHER MODE IS THE "INSERT" MODE, AND IT IS THIS MODE -- WITHOUT ED'S PROMPT (*) -- IN WHICH YOU ENTER YOUR TEXT. THIS WILL BECOME SELF EVIDENT AS YOU PROCEED!)

Our file contents (contents of "THISFILE.TXT") are NOT yet in ED's buffer memory. If there was no file named THISFILE.TXT, then ED informs us of this fact -- NEW FILE is displayed -- and we are

ready to start inserting text with the Insert command.

All ED commands should be entered in lower case. (I will explain why later.) Thus, the "insert" command should be "i" rather than "I".

To get preexisting contents of file into buffer memory you type #a<cr>. This appends (puts all of preexisting file into buffer memory) ALL into memory, if it will fit. Otherwise, it NEARLY fills available memory, then stops. (See below.)

The invisible pointer is at the beginning of the file. (Note that if you are working with an extremely LARGE file, and buffer memory is not large enough to hold the entire file, then the invisible pointer is at the END of the partially appended file, and you must put it at the beginning using the command b<cr>.) When you are finished editing this first "append", then type #w<cr> (writes buffer memory to disk), then type #a<cr> to append the next portion of the large file so you can edit it.)

The Insert command (i) in ED.COM is used by simply typing i<cr> and then you start typing the text you want to insert. You finish the insertion by typing a CTRL-Z (hold down the CTRL key and depress the z key).

\ CTRL-Z is also used as a delimiter in the "search and replace" command feature.

Now, lets summarize the differences between HUG's ED (one referred to in Getting Started with a Text Editor) and CP/M's ED:

With HUG's ED you hit the ESC key one time as a delimiter, and you hit the ESC key 2 times to terminate an insertion, and hit the ESC key 2 times to execute a command (or series of commands). On the other hand, with CP/M's ED, you type a CTRL z as a delimiter, and a CTRL z to terminate an insertion, and hit the RETURN key to execute a command or series of commands. And, when you use CP/M's ED, always use lower case when you type the commands.

Other than the above differences the two Editors are used in much the same way.

Here is an example of how to create a short file using CP/M's ED - (Note that the text within square brackets [text] tells you what YOU do.)

From the monitor prompt (A>) you type:

ED newfile.txt<cr> [this is how you invoke ED; material to right of the space after ED is called a "command line"]
[ED will be loaded into memory and will respond NEW FILE and then its prompt (*)]
i<cr> [you type i and hit RETURN]
This is first line of file.<cr>
[you type this line and hit RETURN]
This is last line of newfile.txt.<cr>
[you type RETURN and a CTRL z. This terminates the insertion. Now type following:]
b#t<cr> [you type b#t and hit Return. b=go to beginning, #t=type all. ED responds by printing out the 2 lines we inserted. The pointer is still at the beginning.]

Now, type t<cr> and ED will type the first line for you. Now, just hit the RETURN key and ED will type the second line. If you had more lines in your file, just hitting the RETURN key would display each succeeding line. (The invisible POINTER also moves down one line each time you hit Return key.)

e<cr> [Now, assume we are finished with file and desire to exit. We type e for End.]

Congratulations, you have just written a file using ED.COM. Use DIR<cr> to note that it is indeed on disk. Use TYPE newfile.txt<cr> to see the disk contents of that file.

With the above in mind, and referring to "Getting Started with a Text Editor", I hope you are able to master ED.COM. Just learn the essential basic commands that I covered in that article, and this should give you a good start in using CP/M's ED.COM.

Why we use lower case for ED's commands

CP/M's ED as supplied has the "u" switch set to minus (-u). This means that you can insert material into a file using lower case, and also, using the search and replace command, you can insert lower case material. What is not yet well documented is the fact that if the respective commands are entered in upper case, then (even though the printed material is displayed on your terminal in lower case) the inserted material that goes into ED's memory buffer is translated into UPPER CASE...even though the "u" switch is set for lower case! So, ALWAYS USE LOWER CASE WHEN ENTERING ED'S COMMANDS!

ED has many powerful features. Here is one. You will have noticed by now that ED has an automatic line numbering feature.

(If you do NOT want this feature simply type, at ED's asterisk prompt, -v<cr>. This turns this feature off.) The automatic line numbering is a very nice feature. If you have some text in memory that you are editing, and you desire to move the invisible pointer to any given line, simply type the line number, a colon (:), and <cr>. Done!

(Note: these line numbers are NOT part of YOUR text. They are simply a feature provided by ED. If you want hard copy (copy provided by your printer) of any given text file, just do a CTRL-P BEFORE you invoke ED! Then, get your text into memory with #a<cr>, then type it all out to your screen using the #t<cr> command. You will get hard copy containing line numbers at the same time!)

Here is an example of how to move the pointer to a line number. Let's pick line number 123. From ED's prompt (*) you simply type

123:<cr>

and there you are. Just type t<cr> to see line 123. (You can, of course, combine these 2 commands as 123:t<cr> and accomplish the same thing.)

Another remarkable feature of ED, and one that is often overlooked, is its ability to include pre-existing files (pre-existing on disk) into a file that you are presently editing. The file on disk MUST have an extension of ".LIB", and it MUST reside on the same disk as the file to be edited (can NOT be on disk B: if file to be edited is on A:). The ED command that carries this out is:

rfilename<cr>

What happens is that whatever text "filename.lib" contains is inserted into whatever text you are editing, just before the current location of the invisible pointer. Here is how to try this out:

Using ED, create a file called XYZ.LIB. Just insert (i<cr>) 2 or 3 lines into the file, then CTRL Z to stop the insertion, type e<cr> to exit ED. You now have on disk a file called XYZ.LIB.

Now, go into ED with another filename given when you invoke ED, insert several lines, terminate insertion with CTRL z, go to beginning of file with b<cr>. Next, hit return key 2 or 3 times (assuming you put a half dozen lines in with the insertion), type rxyz<cr>. You have just inserted the contents of file xyz, just before the invisible pointer (which was at the beginning of the last line displayed before you typed the rxyz<cr>.) To prove this, simply type b#t<cr> and the present contents of your current file are displayed. Note that it includes the contents of xyz.lib file! Type e<cr> to exit ED.

The "h" command. This is a very nice command that can save the user many headaches! If you type h<cr> (from ED's asterisk prompt), ED writes your total file to disk, and THEN automatically opens same file and returns you to ED. When you see the asterisk prompt, just #a<cr> and your edited file is back in memory. You should make frequent use of this command when you are doing extensive editing. This way IF something untoward happens, you have the partially edited file on disk, rather than just in memory.

The "n" command. Here is another frequently overlooked command. It is somewhat similar to the "f" (find) command. You may think of "n" as meaning "next". Suppose you have a VERY long file, one that is too large to fit in memory with one append. Suppose that you desire to find some "unique word(s)" somewhere towards the middle or end of the file. You invoke ED in the usual way, but you do NOT have to use the "a" (append) command. From the asterisk prompt, type:

nunique words^z01t<cr>

Let's examine that command since there are several different commands that are all lumped together. We typed n (which is the "next" command), then immediately typed our unique word or words, then terminated the "words we are trying to find" with a CTRL-Z (hold down CTRL key and also depress Z key), then we entered 01t which is a zero and the letter 1 (which means "go to the beginning of the line"), and t for type (see below for more on "01t"). Then we hit Return key. ED will automatically append (and also write if necessary) until it finds a "match" for the unique word(s) entered. It stops, and then 01t moves the pointer to the beginning of that line, and ED types out the line for you. Now, you can do whatever editing you want in this area of the file, then simply e<cr>. There are other ways you can use this command, so experiment with it.

The "x" command is a very useful command which allows you to move "blocks" of text around. It is nicely covered in Hogan's Guide, and a good explanation is also found in Appendix A of ED: A Context Editor for the CP/M Disk System (one of the Digital Research manuals).

Last, before leaving this section on the use of ED.COM, I repeat something from the article in REM issue 11 -

If you are ever in doubt as to the location of the "invisible pointer" while you are editing a file with ED, simply type 01t<cr> (zeroLT<cr>) and the pointer will be just BEFORE the first character of the

line just displayed for you by ED.

DDT

DDT is the "debugger" which is primarily used in checking any assembly language programs you may write. I will not go into detail about this use. DDT can also be used to patch a machine language file. I will give below a patch which you SHOULD make to the CP/M file SUBMIT.COM, IF you have CP/M vers 2.202 from Heath. Please note that SUBMIT.COM in version 2.203 from Heath has ALREADY BEEN PATCHED for you and you do NOT have to make the following patch. It is not difficult to make the patch to vers 2.202. The material in square brackets to the right of the actual "patch" are not a part of the "patch" but merely some of my notes to help you understand what is going on. This "patch" fixes SUBMIT.COM so that it will understand and act upon a CTRL-Z. The CTRL-Z would be entered as an up arrow followed by a Z (^Z). You may have use for this if you are using SUBMIT when doing some editing. Do NOT patch your distribution disk. Copy SUBMIT.COM and DDT.COM to another "bootup" disk and carry out the following procedure. This patch was obtained from Digital Research. What you type is shown in bold print. From the A> prompt:

```
DDT SUBMIT.COM<cr>      [you type DDT SUB-
                        MIT.COM and hit the
                        RETURN key]
DDT VERS 2.2            [DDT displays these
NEXT  PC                three lines and its
0600 0100               prompt (-).]
-L441<cr>               [You type L441 and hit
 0441  SUI  61          RETURN.  DDT will dis-
-0443  STA  0E7D        play these 10 lines.
 0446  MOV  C,A         If the number to the
 0447  MVI  A,19        right of SUI is 41,
 0449  CMP  C           then the patch has
 044A  JNC  0456        already been made.  Do
-044D  LXI  B,019D      a CTRL-C to exit.  If
 0450  CALL 02A7        your screen does not
 0453  JMP  045E        match these lines, do
 0456  LDA  0E7D        not continue.]
-S442<cr>
 0442  61 41<cr>
 0443  32 .<cr>         [Type a period, <cr>.]
-G0                     [Letter G and zero.]

A>SAVE 5 SUBMIT.COM<cr>
```

Congratulations! You have just patched a machine language file. What went on in the above was that you used DDT's L (LIST) command to type out 11 lines beginning with memory location 441. After checking listing you used DDT's S (SHOW) command to display contents of memory at HEX address 442. It showed a 61. Then you changed it to 41, ended the S command with a period (.) and used the "G0" command to exit DDT. You simply changed one byte in SUBMIT.COM in memory. Next, you used the SAVE command to write the program from memory to disk (the "5" in the SAVE command is a decimal number indicating that there were 5 256-byte "pages" saved. Done!

SUBMIT

Note: When you use the SUBMIT and the XSUB programs, be sure you are logged in on the A: drive, and ensure that SUBMIT.COM, XSUB.COM, (ED.COM if your procedure involves ED editor and any .LIB files), and any .SUB files ALL are present on the disk in drive A:. Any other utility programs such as PIP.COM should also be on this disk if your .SUB file references them!

SUBMIT is CP/M's version of a "batch processor". What this means is that you will be able to put together (using an editor) a list of commands that you might wish to carry out on more than one occasion. For example, suppose that one daily routine at present consists of booting up CP/M, loading MBASIC and running a program called THIS.BAS. After you are finished with this program you desire to copy a file called IMP.DAT from your "boot disk" in drive A: to a backup disk in B:, then you wish to have the DIR of B: displayed. You simply create (using an editor) the list of commands that you would ordinarily carry out. And, the file that you create MUST have an extension called ".SUB". Let us say that you desire to call this ".SUB" file ABCD.SUB. Using ED, you would (from the monitor prompt A>) type ED ABCD.SUB. Then, when ED gave you its asterisk (*) prompt, you would type i<cr> and then type the following commands:

```
MBASIC THIS<cr>
PIP B:=IMP.DAT<cr>
DIR B:
```

Now you type a CTRL-Z, type, e<cr> and ED puts on disk a file called ABCD.SUB.

Here is how you use SUBMIT with this illustration. From the monitor prompt (A>) type:

```
SUBMIT ABCD<cr>
```

SUBMIT will execute your first command which loads MBASIC and runs your program called THIS.BAS. If the last logical statement in your program is SYSTEM (or if program ends, and you type SYSTEM<cr> to exit MBASIC), you can just sit back and watch SUBMIT automatically copy your file and then display the DIR of disk in drive B:. Now, this was an extremely simple illustration. SUBMIT is capable of handling many more commands. All you have to do is have them in a file created by an editor, and remember the file name must have the extension ".SUB". Also note that ALL COMMANDS GIVEN IN THE FILE CREATED BY ED WERE IN UPPER CASE. This is really NOT necessary as SUBMIT converts all lower case commands to upper case, but serves as a reminder to user of the following peculiarity of SUBMIT. Not only are your com-

mands converted to upper case, but also, if you are doing some editing with ED using SUBMIT, and are using the "F" (find) command, or the "S" (search and replace) command, SUBMIT also translates the "string to be searched for" to upper case, even though you entered it as lower case. Hence, the "search" will fail! You can NOT use some features of ED, with SUBMIT, unless you are aware of this "flaw". REMEMBER, SUBMIT will translate all entries into UPPER CASE with ED. (SUBMIT will copy files using PIP, however, WITHOUT any lower case conversion.) If you wished to have ED edit a file and wish to insert one or more short lines, here is how you do it (I am indebted to Curt Geske of Digital Research for this information). You do NOT use a <cr> after the Insert command (i), and you use a CTRL-L (entered as ^L) INSTEAD OF a <cr> after each short line, and you use a CTRL-Z to stop the insertion. All of this is entered on one line. Suppose I wished to add 3 short lines to a preexisting text file. Here is what I might have in my .SUB file:

```
XSUB
ED THISFILE
#A
2L
ITHIS IS ADDED^LTHIS TOO^LAND THIS^L^Z
E
```

Each CTRL-L (entered as ^L) in the above file, is interpreted by ED as a carriage return/line feed. The CTRL-Z (entered as ^Z) ends the insertion.

If the above was in a file named XYZ.SUB, and there was a preexisting file named THISFILE, and from the monitor prompt we entered

```
SUBMIT XYZ<cr>
```

then SUBMIT would insert 3 lines into THISFILE, just above the third line of the original contents of THISFILE. Try it!

Please note, however, that the insertion (if entered as lower case in the above .SUB file) would still be translated into upper case by SUBMIT. There just is NO way at present of inserting lower case material using ED with SUBMIT! Sorry!

You can also insert material as follows. You can insert a ".LIB" file into a program being edited by SUBMIT using the "R" command, BUT if the .LIB file contained lower case material, SUBMIT would translate it to UPPER CASE. Sorry again, but that is just the way SUBMIT works.

Here are 2 items that can NOT be in the list of commands/programs that you put into a ".SUB" file. You can NOT put in a CTRL C (even though entered as ^C). You can NOT put a <cr> on a line by itself (to use just a <cr> as a "keyboard response" when using XSUB -- see below)! You can, however, put comments in a SUBMIT file. Just introduce the comment line with a semicolon (;) in the first column, as in this example:

```
XSUB
;THIS LINE IS A COMMENT
ED TESTFILE
ITHIS LINE IS INSERTED^Z
E
```

Comments should not be used within ED or other programs, but can be used any time the command is for CP/M itself, because it is the CCP (Console Command Processor) in CP/M that allows this, and it has nothing to do with SUBMIT. You can even put escape sequences to control your terminal (such as ESC-E to clear the screen) in comment lines as long as the argument is upper case.

Note that if you desire to use SUBMIT with any program that ordinarily requires you to do something from the keyboard, and you wished to put the entries into your .SUB file, then the first entry in your ".SUB" file MUST be XSUB. (XSUB is another CP/M program that allows you to put the desired "keyboard responses", in advance, into your .SUB file.) SUBMIT works with DDT, and with your assembler (ASM.COM). It does NOT work with some advanced word processors; and it does NOT allow you to put "keyboard actions" into your .SUB file when you are using MBASIC. (Example: Your MBASIC program stops and asks you for the date. You could NOT put the date in the .SUB file. MBASIC requires your personal attention to such matters.) Be sure to remember to "patch" SUBMIT to allow CTRL Z action (see above under DDT) if you have CP/M vers 2.202 as mentioned above. Also remember that if you are using ED with SUBMIT, it really doesn't help you UNLESS your file to be edited is all UPPER CASE! Incidentally, you enter a CTRL-Z as ^Z (up arrow z) in your .SUB file if you need to enter it for LATER execution.

Finally, SUBMIT has an extremely handy feature. Suppose that you did not know in advance, certain items such as "filename", which drive you would be using, etc. SUBMIT allows you to use variables in your .SUB file - $1, $2, $3, $4, etc., if needed. You put these into your .SUB file. In our example given above, your file ABCD.SUB could look like this:

```
MBASIC THIS
PIP $1=$2
DIR $3
```

With this example you might enter the following command line to start SUBMIT:

```
SUBMIT ABCD B: IMP.DAT A:<cr>
```

Note the spaces separating the 3 items

AFTER the name of the .SUB file. What happens is that SUBMIT plugs in the first item (B:) where $1 was in the ABCD.SUB file, substitutes the second item for $2 and the third item for $3. So, when SUBMIT plays out this scenario, what happens is:

```
MBASIC THIS
PIP B:=IMP.DAT
DIR A:
```

Note also that you could have $1, and/or $2, and/or $3 in more than one of the commands in any given .SUB file.

MBASIC

I will cover MBASIC vers 5.2 in another article. I do wish to mention here (and will repeat this in that article) the only "flaw" in MBASIC vers 5.2 that I know of. When you "SAVE" a program from MBASIC to disk, using CP/M, always use UPPER CASE for the file name. For example -

SAVE "MYPGM<cr> and not SAVE "mypgm<cr>.

What happens is that the program IS saved and the program name IS put into the directory, if the program name was entered in lower case, BUT the program name is also put into the directory IN LOWER CASE. Microsoft apparently forgot to "mask" the lower case to UPPER CASE before sending it to the DIRectory. If your program name was saved in lower case, a DIR<cr> listing would show it as mypgm.BAS.

A listing such as mypgm.BAS can NOT be erased from the directory using ERA mypgm.BAS<cr>. (You CAN delete it from the directory by going into MBASIC and entering: KILL "mypgm.BAS"<cr>.)

Also, you would only be able to load this program into memory by entering - LOAD "mypgm<cr>. The MBASIC command LOAD "MYPGM<cr> would fail.

Of course, you can easily work around this potential problem. All you have to do is to ALWAYS "save" your programs using UPPER CASE!

Also, ALWAYS enter filename in UPPER CASE when you use the LOAD command! If the file exists in the DIRectory in UPPER CASE, and you enter filename in lower case in LOAD command, then you get error (file not found.)

The same problem is present when you "Open" files. ALWAYS use UPPER CASE when entering file names!

Now, the good points of MBASIC vers 5.2 are too numerous to mention here. Suffice it to say that "random file" handling is much easier with this version of MBASIC than in HDOS's version of MBASIC. I will cover all this in the next article.

Last, I would like to thank all the folks at HUG, and Barry Watzman, who have "proofed" this article and corrected errors in context. Any remaining errors are entirely mine.

CP/M is a registered trademark of Digital Research. MBASIC refers to "Microsoft BASIC-80", a product of Microsoft, Inc.

EOF

Local HUG News

Ted Benglen, II announces the formation of a Northern Colorado Heath Users' Group, FT.HUG (Fort HUG). They will meet at least once a month in the Ft. Collins area, and serve users in the Loveland, Greeley, Longmont and Boulder areas. Anyone wishing additional information may contact Ted Benglen, II at 822 E. County Road 30, Ft. Collins, CO 80525. (303) 669-4116

•

The Naval Postgraduate School Hobby Computer Club would like to invite all Heath users in the Monterey area to attend one of their meetings. They have regular meetings on the first Thursday of each month at 7:00 p.m. in Room 100, Spanagel Hall, Naval Postgraduate School, Monterey CA. There is an informal Heath Users' Group there with members of varied backgrounds. For addition information Heath users may contact: Tom McNair, NPS Hobby Computer Club, Rec. Services, NPS, Monterey, CA 93940.

The PNHUG (Pacific-Northwest HUG) was inadvertently left off the list of local HUG groups in Issue 25 of REMark. They meet the second Tuesday of odd months from 7:00-9:00 p.m. at the Tukwila, WA Heathkit Center and even months they meet the first Monday from 7:00-9:00 p.m. at the Seattle, WA Heathkit Center. Their contact person is Nathan Hall at 10553 41st Pl. NE, Seattle WA 98125 phone: (206) 363-3927. Mailing address for the club is: PNHUG (Pacific-Northwest HUG), c/o Jan Johnson, PO Box 993, Bellevue WA 98009. They also have a 24 hour Bulletin Board at the Tukwila Heathkit Center (206) 246-4468. Presently there are 150 in their group.