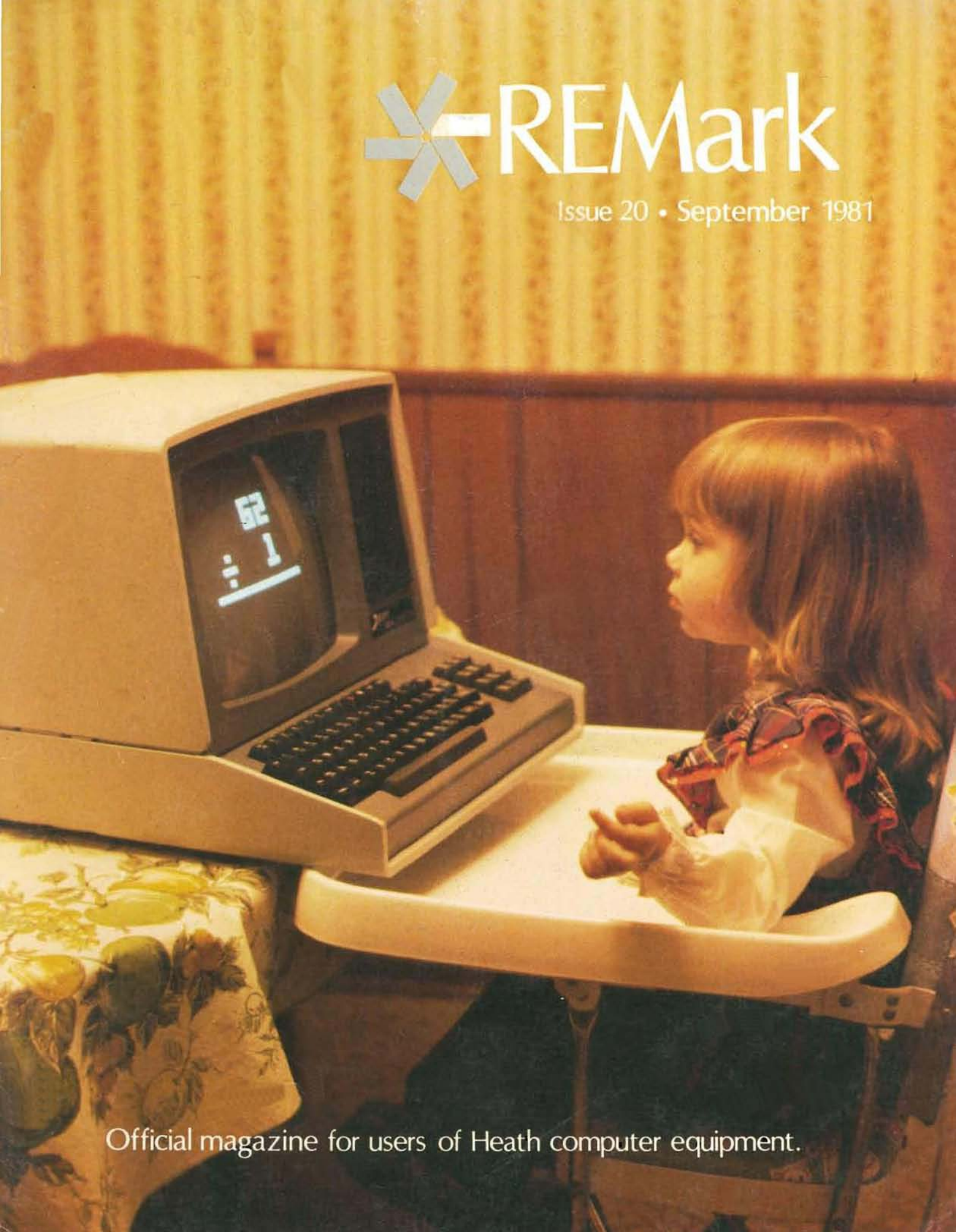
REMark

Issue 20 • September 1981



Official magazine for users of Heath computer equipment.

on the cover . . . .

""but Uncle Gerry . . . ?

BACK-TO-SCHOOL

NUMBERS — An example of COMPUTER AIDED IINSTRUCTION.

Photo by Gerry Kabelman

on the stack

>CAT

**Computer Aided Instruction*

"REMark" is a HUG membership magazine published ten times yearly. A subscription cannot be purchased separately without membership. the following rates apply.

	U.S. Domestic	Canada & Mexico	International
Initial	$18	$20 US FUNDS	$28
Renewal	$15	$17 US FUNDS	$22

Membership in England, France, Germany, Belgium, Holland, Sweden and Switzerland is acquired through the local distributor at the prevailing rate.

Back issues are available at $2.50 plus 10% handling and shipping. Requests for magazines mailed to foreign countries should specify mailing method and add the appropriate cost.

Send payment to:

Heath Users' Group
Hilltop Road
St. Joseph, MI 49085

HUG Manager and Editor .....Bob Ellerton
Assistant Editor and
Software Developer .....Patrick Swayne
HUG Secretary .....Nancy Strunk
Software Developer .....Gerry Kabelman
HUG BB .....Terry Jensen

REMark

NUMBERS

"But Uncle Gerry, I don't know how to do that!!", exclaimed Melissa when the division problem appeared on the screen as shown on the cover of this issue of REMark.

Melissa's complaint is about the program called NUMBERS.BAS and follows on the next couple of pages.

NUMBERS.BAS is an example of a COMPUTER AIDED INSTRUCTION (CAI) program. Many educators, parents and students have requested CAI programs for use with the Heath/Zenith computers. HUG is offering the NUMBERS.BAS program as an example of what a CAI program can do. HUG also has available a complete disk of CAI programs running under HDOS and MBASIC. This disk, HUG part 885-1097 ($20.00), is described in detail in issue #18 of REMark.

NUMBERS.BAS is a very good example of a CAI program for addition, subtraction, multiplication and division using one inch numbers for the early elementary students.

The program is broken down into four sections and may best be explained by reviewing the listing and trying it.

The sections are:

Description	Line Numbers
1. Startup	10 to 200
2. Letter & Number Setup	1000 to 1550
3. Main Program	2000 to 2560
4. Subroutines	10000 to 16040

The letter and number setup is the most important part of this program. Lines 1000 to 1040 create the parts of the characters, while lines 1100 to 1550 actually create the characters from the parts. Also note the subroutines for inputting and printing the large characters.

This program may not make you an Einstein, but it will help you feel like you could be one.

:GK:

```
10 '    NUMBERS.BAS              Version 07.01.81        :GK: @

100 CLEAR 9000:WIDTH 255:DIM L$(130)
110 E$=CHR$(27):EP$=E$+"E":Y$=E$+"Y":F$=E$+"F":G$=E$+"G":P$=E$+"p":Q$=E$+"q"
120 Y5$=E$+"y5"+Q$:X5$=E$+"x5":Q1$=CHR$(34):A1$=CHR$(64):J$=E$+"J":@

200 PRINT EP$X5$:LS$="   " @

1000 B$=E$+"B":D$=E$+"D":D1$=D$+D$:D$=D1$+D1$+D1$+B$:D1$=D1$+D$:@
        N1$="iiiiii"+D$:N2$="    ii"+D$:N3$="ii    "+D$:@
        N4$="  ii  "+D$:N5$="ii  ii"+D$:N6$="iiii  "+D$:@
        N7$="iii ii"+D$:N8$="ii ii "+D$ @

1010    M1$="ii iii"+D$:M2$=" iiii "+D$:M3$="   ii "+D$:@
        M4$=" ii   "+D$:M5$="  iiii"+D$:M6$="iiiii "+D$:@
        M7$="  iii "+D$:M8$=" ii ii"+D$:M9$="      "+D$ @

1020 O1$="ii ii ii"+D1$:O2$="iii  iii"+D1$:O3$="iiiiiii"+D1$ @

1030 C$=E$+"C":C$=C$+C$+C$+C$:C$=C$+C$:A$=E$+"A":R$=A$+A$+A$+A$+A$+C$:UP$=A$
1040 B1$=D1$+UP$:B2$="        "+D1$:@
```

```
1100 '                          Setup of Characters
1110 L$(7)="":'                                      Bell
1120 L$(8)=B1$+B2$+B2$+B2$+B2$+B2$+R$:'              Backspace
1130 L$(32)="      ":'                               ( ) SPACE
1140 L$(33)=N1$+N1$+N1$+N1$+M9$+N1$+R$+UP$:'         !
1150 L$(37)=N5$+M3$+N4$+M4$+N5$+R$:'                 %
1160 L$(43)=M9$+N4$+N1$+N4$+M9$+R$:'                 +
1170 L$(45)=M9$+M9$+N1$+M9$+M9$+R$:'                 -
1180 L$(48)=N1$+N5$+N5$+N5$+N1$+R$:'                 0
1190 L$(49)=" iii  "+D$+N4$+N4$+N4$+N1$+R$:'         1
1200 L$(50)=N1$+N2$+N1$+N3$+N1$+R$:'                 2
1210 L$(51)=N1$+N2$+M5$+N2$+N1$+R$:'                 3
1220 L$(52)=N5$+N5$+N1$+N2$+N2$+R$:'                 4
1230 L$(53)=N1$+N3$+N1$+N2$+N1$+R$:'                 5
1240 L$(54)=N1$+N3$+N1$+N5$+N1$+R$:'                 6
1250 L$(55)=N1$+N2$+N2$+N2$+N2$+R$:'                 7
1260 L$(56)=N1$+N5$+N1$+N5$+N1$+R$:'                 8
1270 L$(57)=N1$+N5$+N1$+N2$+N1$+R$:'                 9
1280 L$(63)=N1$+N5$+N2$+M5$+N4$+B$+N4$+R$+UP$+UP$:'      ?
1290 L$(65)=M2$+N5$+N1$+N5$+N5$+R$:'                 A
1300 L$(66)=N1$+N5$+M6$+N5$+N1$+R$:'                 B
1310 L$(67)=N1$+N3$+N3$+N3$+N1$+R$:'                 C
1320 L$(68)=N1$+M8$+M8$+M8$+N1$+R$:'                 D
1330 L$(69)=N1$+N3$+N6$+N3$+N1$+R$:'                 E
1340 L$(70)=N1$+N3$+N6$+N3$+N3$+R$:'                 F
1350 L$(71)=N1$+N3$+M1$+N5$+N1$+R$:'                 G
1360 L$(72)=N5$+N5$+N1$+N5$+N5$+R$:'                 H
1370 L$(73)=M2$+N4$+N4$+N4$+M2$+R$:'                 I
1380 L$(74)=N2$+N2$+N2$+N5$+N1$+R$:'                 J
1390 L$(75)=N5$+N8$+N6$+N8$+N5$+R$:'                 K
1400 L$(76)=N3$+N3$+N3$+N3$+N1$+R$:'                 L
1410 L$(77)=O2$+O3$+O1$+O1$+O1$+"  "+R$:'            M
1420 L$(78)=N5$+N7$+N1$+M1$+N5$+R$:'                 N
1430 L$(79)=L$(48):'                                 O
1440 L$(80)=N1$+N5$+N1$+N3$+N3$+R$:'                 P
1450 L$(81)=N1$+N5$+N5$+M1$+N7$+R$:'                 Q
1460 L$(82)=N1$+N5$+N1$+N8$+N5$+R$:'                 R
1470 L$(83)=L$(53):'                                 S
1480 L$(84)=N1$+N4$+N4$+N4$+N4$+R$:'                 T
1490 L$(85)=N5$+N5$+N5$+N5$+N1$+R$:'                 U
1500 L$(86)=N5$+N5$+N5$+M2$+N4$+R$:'                 V
1510 L$(87)=O1$+O1$+O1$+O3$+O2$+"  "+R$:'            W
1520 L$(88)=N5$+M2$+N4$+M2$+N5$+R$:'                 X
1530 L$(89)=N5$+N5$+M2$+N4$+N4$+R$:'                 Y
1540 L$(90)=N1$+N2$+N4$+N3$+N1$+R$:'                 Z
1550 L$(127)=L$(8):'                                 Delete @

2000 '                          Print "HELLO MY NAME IS Z89"
2010 A1=1:PRINT Y$" 0";:S$="HELLO":GOSUB 16030:PRINT Y$"& ";:S$="MY NAME IS":@
       GOSUB 16030:PRINT Y$",6";:S$="Z89":GOSUB 16030:GOSUB 11000:@

2020 '                          Print "WHAT IS YOURS?"
2030 PRINT EP$;:S$="WHAT IS":GOSUB 16030:PRINT Y$"& ";:S$="YOURS":GOSUB 16030:@
       PRINT Y5$" "Y$". ";:S$="?":GOSUB 16030
2040 GOSUB 15010:@

2050 '                          Print "HOW MANY QUESTIONS?"
2060 PRINT EP$;:S$="HOW MANY":GOSUB 16030
2070 A=0:B=0:C=0:
2080 PRINT Y$"& ";:S$="QUESTIONS":GOSUB 16030
2090 PRINT Y$"00";:S$="? ":GOSUB 16030:N=1:GOSUB 15010
2100 A=VAL(A1$)
2110 IF A<1 THEN 2060 ELSE 2120 @

2120 A=INT(A):'                 Allow only integers
2130 PRINT EP$:@
```

```
2140 '                          Start selecting numbers to be used. @

2150 W=INT(RND(1)*PEEK(8219)):' Random Function @
                                1=Addition      @
                                2=Subtraction   @
                                3=Multiplication@
                                4=Division
2160 IF W>4 THEN 2150
2170 J=10:J1=10
2180 IF W=2 THEN J=20:GOTO 2200
2190 IF W=4 THEN J=100:GOTO 2200
2200 X=INT(RND(1)*PEEK(8219))+1:'@
                                1st Random Number
2210 IF X>J THEN 2190
2220 FOR I=1 TO W*(RND(1)):NEXT I
2230 Y=INT(RND(1)*PEEK(8219))+1:'@
                                2nd Random Number
2240 IF Y>J1 THEN 2200
2250 ON W GOTO 2260,2270,2280,2290
2260 Z=X+Y:GOTO 2320:'          Find Sum
2270 IF X>Y THEN Z=X-Y:GOTO 2320:@
       ELSE GOTO 2150:'         Find Subtrahend
2280 Z=X*Y:GOTO 2320:'          Find Multiplicand
2290 IF Y=0 THEN 2150
2300 IF INT(X/Y)=X/Y THEN Z=X/Y @
       ELSE GOTO 2150:'         Find Dividend @

2310 '                          Print Numbers
2320 GOSUB 13010
2330 PRINT P$Y$"0?";:N=1:GOSUB 15010:Z1=VAL(A1$)
2340 PRINT Y$"0;"E$"J";
2350 IF LEN(A1$)=3 THEN 2380 @

2360 Z2=Z1:GOSUB 14010:'        Reprint Answer if only one character @

2370 '                          If answer is CORRECT
2380 IF Z=Z1 THEN PRINT EP$;:S$="   RIGHT":GOSUB 16030:PRINT Y$"& ";:@
       GOSUB 12010:GOSUB 11000:B=B+1:C=C+1:IF A=B THEN 2420 ELSE 2130 @

2390 '                          If answer is INCORRECT
2400 Z2=Z:PRINT Y$"0 "J$;:S$="  WRONG!":GOSUB 16030:GOSUB 11000:PRINT Y$"0 "J$;:@
       GOSUB 14010:B=B+1:IF A=B THEN 2420 ELSE 2130 @

2410 '                           Print up tally display
2420 PRINT EP$Y$"  ";:S$="END  SCORE":GOSUB 16030 @

2430 '                          C=Score & A=Number of Problems
2440 PRINT Y$"& ";:S$=STR$(C):GOSUB 16000
2450 S$=" RIGHT":GOSUB 16030
2460 PRINT Y$", ";:S$="OUT OF ":GOSUB 16030
2470 S$=STR$(A):GOSUB 16030 @

2480 '                          Figure Percentage Correct * Display It
2490 IF C/A=1 THEN PRINT F$Y$"20";:S$="100 %":GOSUB 16030:GOTO 2510
2500 C1=INT(C/A*100):S$=STR$(C1)+" %":PRINT Y$"2"CHR$(64);:GOSUB 16030
2510 GOSUB 11000:@

2520 '                          Delay Visual "BYE"
2530 GOSUB 11000
2540 S$="AGAIN? ":PRINT EP$;:GOSUB 16030:PRINT Y5$;:A1$=INPUT$(1):PRINT X5$;
2550 IF A1$="Y" OR A1$="y" THEN 2060
2560 PRINT EP$;:S$="  BYE":GOSUB 16030:PRINT Y$"& "Y5$:END:@

10000 '                         *** SUBROUTINES ***     @

11000 FOR S=1 TO 100:NEXT S:'   Time Delay
11010 RETURN @

12000 '                         Select & Print Correct Message
12010 ON INT(RND(1)*5)+1 GOTO 12020,12030,12040,12050,12060
12020 S$="   BRAIN":GOTO 16030
```

```
12030 S$="    WOW":GOTO 16030
12040 S$="HAVE MERCY":GOTO 16030
12050 S$="FANTASTIC":GOTO 16030
12060 S$=" EINSTEIN":GOTO 16030 @

13000 '                          Print Numbers For Problem
13010 PRINT EP$Y$" ?";:S$=STR$(X):GOSUB 16000:PRINT Y$"'?";:S$=STR$(Y):GOSUB 16000
13020 ON W GOTO 13030,13040,13050,13060
13030 PRINT F$Y$"(5"L$(43);:GOTO 13070
13040 PRINT F$Y$"(5"L$(45);:GOTO 13070
13050 PRINT F$Y$"(5"L$(88);:GOTO 13070
13060 PRINT F$Y$"(5"N4$M9$N1$M9$N4$;
13070 PRINT Y$".0"STRING$(46,"i")G$:RETURN @

14000 '                          Print Answer In Proper Position
14010 S$=STR$(Z2):PRINT Y$"0?";:GOSUB 16000:GOSUB 11000:GOSUB 11000:RETURN @

15000 '
15010 A1$="":'                   Input Characters
15020 PRINT Y5$;:A$=INPUT$(1):PRINT X5$;:IF A$=CHR$(13) THEN 15090
15030 IF A$=" " THEN PRINT "          ";:A1$=A1$+" ":GOTO 15020
15040 IF (A$=CHR$(127) OR A$=CHR$(8)) AND LEN(A1$)>0 THEN 15100
15050 IF N=1 AND (A$>"/" AND A$<":") THEN @
        PRINT F$L$(ASC(A$))G$;:A1$=A1$+A$:GOTO 15020
15060 IF A$>"`" AND A$<"{" THEN A$=CHR$(ASC(A$)-32)
15070 IF N=1 OR (A$<"A" OR A$>"{") THEN PRINT "";:GOTO 15020
15080 PRINT F$L$(ASC(A$))G$;:A1$=A1$+A$:IF LEN(A1$)=9 THEN 15090 ELSE 15020
15090 N=0:GOTO 16040
15100 IF RIGHT$(A1$,1)="W" OR RIGHT$(A1$,1)="M" THEN PRINT E$"D"E$"D";
15110 PRINT L$(8);:A1$=LEFT$(A1$,LEN(A1$)-1):GOTO 15020 @

16000 S2$=S$:'                   Print LARGE Characters
16010 IF LEN(S$)>2 THEN 16030
16020 IF LEN(S2$)<3 THEN PRINT L$(32)"  ";:S2$=S2$+" ":GOTO 16020
16030 FOR I=1 TO LEN(S$):PRINT F$L$(ASC(MID$(S$,I,1)))G$;:NEXT I
16040 RETURN
```

REMark 20 — A Special Issue

This Issue of REMark is indeed special! Contained in the following pages is probably one of the most important single articles (or should I say manuals?) that has been featured in a long time. Al Dallas, the Editor of NIBBLE, in cooperation with both Dale Lamb and Tom Jorgenson have produced The HDOS Device Driver Programmers Guide. Similiar to The HDOS Programmers Guide which most of you are familiar with, this great piece of work answers some of those questions most often asked. How about this one? -- "What is PIC CODE and how is it implemented?" OR -- "How can I construct a special DVD for my XYZ printer?" Maybe this one -- "What can be accomplished by constructing my own DVD?"

All of these questions and a lot more will be answered by reading what AL, Dale, and Tom have put together in the following pages. Your Heath Users' Group feels that this information is SOOO complete and useful, that the entire text and related appendices are included in the following pages. The information even includes, as Appendix D, Dale Lambs CK.DVD. CK.DVD is a real-time clock that actually becomes a part of your operating system.

Be sure that you hold on to Issue 20 of REMark as it will probably be one that will be termed "collector's item" in the future. Further, the information in The HDOS Device Driver Programmers Guide will prove most valuable as we, together, strive to do more "work" with our computers!

The HDOS Device Driver Programmer's Guide

By Al Dallas (70250,637),
Dale Lamm (70555,302), and
Tom Jorgenson (70120,153)

Introduction

What is a device driver? Under HDOS, a device driver is a relocatable program (usually less than 3K) which the operating system loads in order to communicate with external devices. When programmers speak of HDOS as a "high-level" or sophisticated operating system, one of the things they have in mind is this device-independence, which makes HDOS adaptable to just about any peripheral equipment.

Originally, device drivers were a method of providing software support for the Heath line of printers. The console and disk device drivers were built-in to HDOS, because 1) they constitute a minimum system, 2) their functions are more sophisticated than those of printers or punches, and 3) it was assumed that end users would not need access to them. Users, it seems, have surprised quite a few with their knowledge, programming ability, and above all, their desire for access to everything about their computer. Heath's introduction of the H47 8" disk drives demonstrated a need for greater flexibility for mass-storage devices as well as printers. The result was version 2.0 of HDOS, which allows device drivers for "mass storage devices", such as disk drives.

Obviously, a device driver is necessary in order to use HDOS and system utilities with a peripheral device. A functioning device driver incorporates the entire system; i.e., SYSCMD's COPY and CAT commands will work with the new device, as will PIP and even MBASIC (provided the driver is loaded first). This is very powerful, as it amounts to "patching" the entire operating system and high-level languages to suit potentially unique external equipment. It also opens the door to psuedo-device driver development, such as a software clock.

We assume the reader has a good knowledge of assembly language programming techniques. The relocatable aspect of the driver code coupled with the numerous system communication parameters, flags and "magic" addresses make device drivers challenging to the uninitiated. Registered owners of HDOS 2.0 have several Software Tools at their disposal -- Heath supplies several device drivers in source code form along with a plethora of .ACM files describing system equates and other useful data.

Environment

The basic HDOS Memory Map includes HDOS resident in high memory, overlay space below it, and user code below that. When overlays are required, they are moved into position, relocated (more on this later) and entered (meaning the program counter jumps to the starting address and begins execution). Overlays are transient -- space is not permanently allocated to them -- unless HDOS must run with SY0: dismounted (i.e., Stand-Alone). Device drivers function the same way, and the driver must be LOADed at the command level (by SYSCMD) in order to allocate space permanently. Loading from within a program (by using the .LOADD system call) does not 'lock' the driver in memory. The routine to do that is explained later.

At boot-time, HDOS builds a Device Table in memory by scanning the Directory for two-letter files with the .DVD extension. It reads in each file's header and checks that it is flagged as a device driver, in an attempt to keep prying hands from creating illicit device drivers. HDOS must be re-booted to re-build this table, so remember to re-boot after assembling, debugging, or renaming a device driver. Curiously, the device table includes the physical sector of the device driver and therefore deleting a .DVD file on SY0: and then referencing it can confuse things badly. HDOS has verified that a valid .DVD file exists at a particular sector which the GRT now flags as available for new files -- prudent programmers will avoid this situation by simply re-booting after all creation, deletion or change of device drivers. Note also that device drivers on disks other than SY0: are not "known" to the system because they are not included in the table.

User programs access any new devices just the same as typical Heath devices. Performing an .OPENW with HL pointing to 'CK:' as the name is just as valid as 'AT:' as the name, though what the driver chooses to do with the data it is sent may be totally different. To better lace the ties with the operating system, HDOS informs the driver as it performs each step. For instance, a MOUNT CK: command causes HDOS to first load the device driver, then check device ready status, and finally mount the device. At each step, as HDOS finishes what it has to do, the driver code is entered and the appropriate function is requested, so that the driver code is informed of the action. The SY: driver, for example, doesn't care about the OPEN command -- it returns 'no error' to HDOS, where the real work is done (setting up tables, flags, etc.). However, an LP: device cannot be mounted, so if this function is requested, the driver must flag an Illegal Function error upon return. MBASIC is even simpler. Any device can be used as a data sink, provided its driver defines it as capable of WRITE, and the driver was first LOADed from the command mode. Just 'OPEN "O",1,"DV:", and start PRINTing data to #1.

PIC Header

PIC stands for Position Independent Code, probably the biggest hurdle to overcome for the potential device driver programmer. Device Drivers are, for all intents and purposes ORG'd at address 6. Why 6 and not 0 in a minute. When HDOS loads the driver into memory, its exact location will vary based on a number of factors including the amount of RAM in the computer. Therefore, HDOS must relocate (change all the addresses) in the driver before running (entering) it, so that the various JMPs and CALLs know where they're going. To keep track of which bytes need relocation, the assembler treats PIC code differently and generates a relocation table of bytes at the end of the actual program. Routines in HDOS process this table and change the necessary addresses.

Binary files in HDOS are stored with a header, or descriptor, to flag the file type and note the starting and entry addresses and the length of code. For this purpose, Heath has supplied the PICDEF.ACM file, which (starting at 0) generates a six-byte header, and creates the offset mentioned above. The header format is:

Byte	Value	Description
0	377Q	Binary File Flag
1	1	File Type = PIC
2,3		Length of Code + Rel Table
4,5		Address of Start of Rel Table

These are the actual bytes -- assembly language (as opposed to machine language) programmers do not refer to them directly. It is important to refer to actual values and absolute addresses by their symbolic names because the code is much more easily adapted to future releases (it also helps reduce a common source of errors). The only way to learn these techniques is by observing examples -- and the Heath device drivers and XTEXTs are especially good. Insert the PICDEF.ACM as an XTEXT last thing before the first actual program instructions, followed by CODE PIC in the opcode and operand fields. The CODE PIC pseudo must come before the start of the program (so that the entire program will be relocatable), but after all the external definitions (such as H17 ROM addresses) which are not to be relocated.

The INIT program supplied with HDOS is capable of initializing just about any mass-storage device when passed certain parameters. MAKMSD.ABS is a program to concatenate your xx.DVD file with a xxINIT.SYS file containing these parameters and device-specific initialization routines. One format for the xxINIT.SYS file is:

512-byte Read-Only Boot Driver (org'd at 42200A)

Sub-Functions
- Media Initialization

Volume Parameters
- Cluster Sizes
- Directory Offsets

Studying the SYINIT and DDINIT examples distributed with HDOS 2.0 will better explain how these files must be configured, but there is considerable latitude.

Device Header

The first 15 bytes of the file constitute the device header, defined as follows:

Byte	Value	Description
0	307Q	Device Driver Flag
1		Device Capabilities Byte
2		Mounted Units Mask
3		Maximum Number of Units
4-11		Unit Capabilities for 0 - 7
12	307Q	Device Driver Flag (if device will take Set options)
13		Pointer to INIT code (set by MAKMSD)

Bear in mind that these are the actual bytes, not the symbolic values. It is dangerous to ignore the Heath XTEXTs, because future compatibility requires their use. The actual bytes are shown here as an aid to understanding what the XTEXTs accomplish. The Device Driver flag is apparently just an arbitrary 11000111 pattern used to validate the driver code. The Device Capability Byte is defined:

Bit:	7 6 5 4 3 2 1 0
0	Directory-Type Device
1	Capable of Read
2	Capable of Write
3	Capable of Random-Access
4	Capable of Character Mode

These capability codes are defined in the DEVDEF.ACM file supplied by Heath. The Mounted Units Mask is typically defined as 0 for variable numbers of units, or 1 for devices with only one unit. The Maximum Number of Units is 8 (0..7) for any device, but your driver can set this to any value between 1 and 8. For each valid unit, the unit capability is flagged as explained above, and each invalid unit if flagged with a zero. These 13 bytes are used directly in building the Device Table entry in HDOS. Another 22 bytes are reserved by Heath prior to the Set Entry Point, by use of the symbolic SET Entry Point, DVD.STE.

Set Entry

The SET.ABS program is used to patch device driver files, among several other things. The address 53Q is considered the SET entry point, and the next 469 bytes are reserved for SET processing. To assure that these important addresses are maintained, Heath device drivers make use of the error-checking abilities of ASM to flag an error if the correct address is not produced. DVD.STE is defined as 53Q, and this code follows the header bytes:

```
.       SET     025Q      (SET is used as an ASM psuedo here)
        ERRNZ   *-.
        DS      DVD.STE-.

SETN    EQU     *
        ERRNZ   *-DVD.STE
```

A 'P' error is generated while assembling the program if this critical address (SETN) is not 53Q, but the technique allows the value of DVD.STE to be changed at any time, reassembling, and preserving the intent of the code. An undocumented feature of the 2.0 Assembler is the ability to turn relocation on and off, and therefore legally SET a value to the Origin pointer (*), thus:

```
        CODE    -REL    Turn Relocation Off
.       SET     *
        CODE    +REL    Turn Relocation On
        DS      DVD.ENT-.
```

These procedures are not mandatory, they simply represent good programming practice, make the code easier to update, and easier for others to work on.

The SET program loads the device driver into a convenient location in memory

and relocates the first 512 bytes (the Header and the SET processor). It then enters the driver's SET portion, passing a unit number parameter in A, and a pointer to the rest of the command line in DE. From here, your program can do anything you want, except that memory values thus updated must be in the part of the code that was not relocated. Only this portion is read back to the disk by SET when done, to save having to un-relocate the set code. Rather than free-wheeling, however, due to the limited space for set functions, most Heath drivers make use of routines in the SET.ABS program itself to process the various options. These useful routines are documented in Appendix C.

The primary routine, $SOP, is the Set Option Processor. $SOP is called with BC pointing to the command line, DE pointing to a processor table, and HL pointing to an option table. $SOP matches the command line to an option in the table, uses the index found in this table to fetch the processor (sub-routine) address, and then jumps to that processor with HL pointing to any additional data in the option table and BC pointing to the rest of the command line. The option table is defined as follows:

```
DW      End of Table Address
DB      Number of Data Bytes following option
DB      'SEARCH STRIN','G'+200Q
DB      Index into Processor Table (8 Bit)
DB      Additional Data Bytes (N - 1)
DB      'NEXT STRIN','G'+200Q

DB      0       (End of Table)
```

The processor table lists routines:

```
DW      HELP
DW      FLAG... etc.
```

Refer to the Heath Device Drivers for the assembly language methods used to implement these tables and provide for modification ease, documentation, and compatibility with future releases.

SET commands should include a HELP command which prints a list of valid commands. Typically, SET commands will either set a flag bit in a variable somewhere, or change a value. To assist these operations, the SET.ABS program includes the $PBF and $PBV routines. Both routines are compatible with $SOP, so that the FLAG or VAL processors listed in the processor table need only jump to $PBF or $PBV. The difference is evident in the option table data structure. $PBF expects at least 5 data bytes following the option string and $PBV expects 6.

```
$PBV Data Bytes:
   DB   Default Radix (2,8,10)
   DB   Minimum Value
   DB   Maximum Value
   DW   Address of Variable

$PBF Data Bytes:
   DB   Mask (Bits to Alter)
   DB   Bit Pattern to Set
   DW   Address of Variable
   DB   0 (if 6 data bytes are used)
```

$PBV is quite sophisicated. The Default Radix is used unless the user specifies B, D, Q, etc. after the value. $PBF uses a fail-safe mask just as the .CONSL SCALL does (see the System Programmer's Guide to HDOS). The remaining SET routines are described in the new SETCAL.ACM included as Appendix C.

Driver Entry

Starting at DVD.ENT (2000A, typically), the remainder of the device driver code is just that -- device driver. HDOS enters at this address with (A) equal to a Device Communication code as defined in DDDEF.ACM. If (A) exceeds DC.MAX, the driver must flag an Illegal Request error. Functions which are logical for the device in question (Write for a printer, for example) must be directed to appropriate processors. However, inappropriate functions must return errors to HDOS. In a gray area between are functions which are not erroneous, but at the

same time require no processing by the driver. These functions simply return no error to HDOS.

READ enters with a byte count in BC (typically a multiple of 256), a buffer address in DE (to which the data must be read), and a block number in HL. A block number is equal to a logical sector number (i.e., 320 as opposed to Track 32, Sector 0). A serial device simply ignores any value in HL.

WRITE is the same as READ, except that DE points to the data to be written out to the device.

READ REGARDLESS is anachronistic. It involves reading the label sector on the disk, disregarding volume number protection. Chances are good your driver can either map it to READ, return no error, or return a Device Not Suitable error without processing it at all.

OPENR opens a file for read. Disk drivers typically ignore all OPENs, but a tape device driver might use OPENR as a signal to rewind a data tape.

OPENW opens a file for write. The LP: device driver, for instance, uses this routine to initialize and prepare the device, a function that would probably be handled by READY if the driver had been first written under 2.0.

OPENU opens a file for update (random read/write). You most likely will not have to deal with this, in that the really tricky part is handled by HDOS.

CLOSE presents a good opportunity to dump a buffer out to a printer, but disk drivers typically ignore it.

ABORT cancels the current operation. The SY: driver resets the device, seeks track zero, and exits with no error flagged. LP: flags a Device Driver Abort error before leaving, however.

MOUNT is used by the SY: driver to set up volume protection and to seek track zero. (Register L = the volume number at entry). LP: ignores this routine.

LOAD is used by SY: to initialize constants in system RAM, re-vector obsolete ROM code, etc. This is a new function for 2.0.

READY is another function added for 2.0. Your code should perform some test to verify that your device is ready and return no error to HDOS. 'C' set indicates that the device is not ready, and HDOS will provide the loop -- this way, HDOS remains cognizant of interrupt requests as opposed to hanging up in your routine.

Often, a device driver may want, under certain circumstances, to load itself permanently in memory. The following code from SYSCMD.SYS explains the simple process.

```
LHLD    S.SYSM          Update System FWA
SHLD    S.RFWA
LHLD    AIO.DTA         Get Device Table Address
LXI     D,DEV.REX       Offset to Residency Flag
DAD     D
MOV     A,M
ORI     DR.PR           Set Flag = Perm. Resident
MOV     M,A
```

These symbols are defined in the DEVDEF.ACM, ESINT.ACM and ESVAL.ACM files. This works because HDOS has variable pointers which are addressing our device driver as it is entered. If the code is to be locked, this routine must be called before any other device I/O is attempted. In fact, it is usually not a good idea to perform system calls from within the device driver, because it was entered using the system call process which is only partially re-entrant.

You may want to include a LON G pseudo in your code just before the end to direct the assembler to list the relocation table. Heath drivers typically include a patch area here as well. With PIC code, entry begins at PIC.COD, so there is no need for an operand with the END statement.

Summary

Writing or modifying a device driver should not be beyond the capabilities of any assembly language programmer. Looking at naked Heath driver code can be confusing due to the large number of symbolic values assigned in .ACM files elsewhere, but studying these examples is the best way to learn about device drivers. Remember to define all external (non-relocating) addresses before using the CODE PIC psuedo, and to confine the variable accessed by SET to addresses higher than DVD.ENT.

This guide represents Al Dallas' study of device drivers and the HDOS Version 1.6 source code listings, along with many suggestions and helpful guidance of two HDOS wizards, Dale Lamm and Tom Jorgenson. There are no warranties, express or implied and Heath Company takes no responsibility for the data herein.

APPENDIX A
Minimum XTEXTs

```
DDDEF.ACM       Device Driver Communication Flags
DEVDEF.ACM      Capability Flags, etc.
DVDDEF.ACM      Driver Header Equates
ECDEF.ACM       Error Code Definitions
ESINT.ACM       For Direct HDOS operations only
ESVAL.ACM       Direct operations only
PICDEF.ACM      PIC Format
SETCAL.ACM      Routines in SET.ABS
```

APPENDIX B
Typical Driver Layout

```
0 - PIC.COD     PIC Header
PIC.COD - 20    Driver Header
21 - 42         Reserved
DVD.STE -       SET Code
 DVD.ENT-1              Entry Processor
                        Processor Routines
                        Processor Table
                        Option Table
DVD.ENT - ?     Driver Code
                        Entry Processor
                        Processor Routines
                        Sub-Routines
                        Data Area
```

APPENDIX C
SETCAL.ACM

```
SETCAL  SPACE   4,10
**      SETCAL - ROUTINES IN SET.ABS
*

$SNA    SPACE   3,10
**      SNA - SCAN TO NEXT ARGUMENT
*       SNA IS CALLED TO SKIP OVER BLANKS
*
*       ENTRY:  (BC) = LINE POINTER
*       EXIT:   (BC) UPDATED
*               'Z' SET IF AT END OF LINE
*       USES:   A,F,B,C
$SNA    EQU     42201A

$DCS    SPACE   3,10
**      DCS - DELIMIT CHARACTER STRING
*
*       ENTRY:  (BC) = LINE POINTER
*       EXIT:   (BC) UPDATED
*               (DE) = ADDR FIRST STRING CHAR
*               (HL) = ADDR LAST STRING CHAR
*               (A) = STRING LENGTH
```

```
*               'Z' SET IF STRING EMPTY
*       USES:   ALL
$DCS    EQU     42204A

$CNA    SPACE   3,10
*       CNA - CONVERT NUMERIC ARGUMENT
*       CNA CONVERTS ARGUMENT IN COMMAND LINE TO
*       A BINARY VALUE
*
*       ENTRY:  (BC) = LINE POINTER
*               (A) = DEFAULT RADIX
*       EXIT:   (BC) = UPDATED
*               (HL) = VALUE
*               'C' SET IF ERROR
*       USES:   ALL
$CNA    EQU     42207A

$FST    SPACE   3,10
**      FST - FIND IN SERIAL TABLE
*       FST SEARCHES A SERIAL TABLE FOR A
*       SPECIFIC KEY
*
*       ENTRY:  (HL) = ADDR OF TABLE
*               (DE) = ADDR OF SEARCH KEY
*       EXIT:   (DE) = UNCHANGED
*               'Z' SET IF MATCH FOUND
*       USES:   A,F,H,L
$FST    EQU     42212A

$TBLS   SPACE   3,10
**      TBLS - TABLE SEARCH
*       TABLE FORMAT:
*       DB      KEY1,VAL1
*       *       *
*       *       *
*       DB      KEYN,VALN
*       DB      0
*
*       ENTRY:  (A) = PATTERN
*               (HL) = ADDR OF TABLE
*       EXIT:   (A) = PATTERN IF FOUND
*               'Z' SET IF FOUND
*       USES:   A,F,H,L
$TBLS   EQU     42215A

$WTBLS  SPACE   3,10
**      WTBLS - WORD TABLE SEARCH
*       LOOK-UP WORD VALUE USING 1-BYTE KEY
*       TABLE FORMAT:
*       DB      KEY1
*       DW      VAL1
*       *       *
*       *       *
*       DB      KEYN
*       DW      VALN
*       DB      0
*
*       ENTRY:  (A) = PATTERN
*               (HL) = ADDR OF TABLE
*       EXIT:   (A) = PATTERN IF FOUND
*               'Z' SET IF FOUND
*       USES:   A,F,H,L
$WTBLS  EQU     42220A

$LBD    SPACE   3,10
**      LBD - LOOK UP BAUD RATE DIVISOR
*
*       ENTRY:  (DE) = BINARY BAUD RATE
*       EXIT:   'Z' SET IF VALID BAUD RATE
*               (HL) = DIVISOR
*       USES:   A,F,D,E,H,L
$LBD    EQU     42223A
```

```
$SOP    SPACE   3,10
**      SOP - SET OPTION PROCESSOR
*
*       ENTRY:  (BC) = LINE POINTER
*               (DE) = PROCESSOR TABLE ADDRESS
*               (HL) = OPTION TABLE ADDRESS
*       EXIT:   (RET) TO PROCESSOR IF NO ERROR
*               (BC) = UPDATED
*               (HL) = NEXT AVAILABLE DATA BYTE
*       USES:   ALL
$SOP    EQU     42226A

$PBF    SPACE   3,10
**      PBF - PROCESS BYTE FLAG
*
*       ENTRY:  (HL) = ADDR OF TABLE VECTOR
*       EXIT:   'C' SET IF ERROR
*       USES:   ALL
$PBF    EQU     42231A

$PBV    SPACE   3,10
**      PBV - PROCESS BYTE VALUE
*
*       ENTRY:  (BC) = NEXT CHAR ADDRESS
*               (HL) = TABLE VECTOR INDEX
*       EXIT:   (BC) UPDATED
*               'C' SET IF ERROR
*       USES:   ALL
$PBV    EQU     42234A
```

APPENDIX D
CK.DVD

```
        TITLE   'Super-Simple Super-Small CK.DVD'
        STL     'Version 2.2  18-Jun-81  D. Lamm'
*       This begins an elementary example of a "device driver", in this case
*       a real-time clock. For the sake of simplicity, no XTEXT's are used.
*       Instead, all required symbol definitions are included in the main
*       body of the source code (this file).
        SPACE   6
DVDFLV  EQU     0C7H            THIS FLAGS TO HDOS AS A DEVICE DRIVER
DVD.ENT EQU     200H            STANDARD DEVICE DRIVER ENTRY POINT
DC.MAX  EQU     11              ELEVEN DRIVER FUNCTIONS CURRENTLY SUPPORTED
DT.CR   EQU     00000010B       FLAG BIT; CAPABLE OF READS
DT.CW   EQU     00000100B       FLAG BIT; CAPABLE OF WRITES
EC.EOF  EQU     01H             ERROR CODE; END OF FILE
EC.FNO  EQU     09H             ERROR CODE; CHANNEL NOT OPEN
EC.ILR  EQU     0AH             ERROR CODE; ILLEGAL REQUEST
EC.FAO  EQU     19H             ERROR CODE; FILE ALREADY OPEN
UIVEC   EQU     201FH           HDOS UIVEC TABLE FROM MTR-88
$TBRA   EQU     193EH           ROM TABLE BRANCH ROUTINE
NL      EQU     0AH             HDOS NEWLINE CHARACTER
CAL     EQU     -1-500          CLOCK CALIBRATION; 500 TICKS=1 SECOND
        SPACE   3
        CODE    PIC
$       EQU     *+DVD.ENT-6     NEED TO DEFINE DVD.ENT AS A RELOCATABLE SYMBOL
        DB      DVDFLV          STICK IN THE DEVICE DRIVER FLAG
        DB      DT.CR+DT.CW     MAKE IT CAPABLE OF READS AND WRITES
        DB      1               MOUNTED UNIT MASK
        DB      1               MAXIMUM NUMBER OF UNITS
        DB      DT.CR+DT.CW     SUB-CAPABILITY IS SAME AS UNIT CAPABILITY
        DS      7               DON'T CARE ABOUT UNITS 2-8 SUB-CAPABILITY
        DB      0               NO SET OPTIONS AVAILABLE
        DS      $-*             RESERVE SPACE UP TO DRIVER'S ENTRY POINT
        STL     'DRIVER ENTRY POINT'
        EJECT
***     CK.DVD  PROCESS ENTRY POINT
*
*       ENTRY:  (A)     = PROCESS CODE
```

```
*               (BC)     = BYTE COUNT
*               (DE)     = DATA BUFFER ADDRESS
*
*       EXIT:   (PSW)    = CARRY CLEAR IF NO ERROR
*                        = CARRY SET IF ERROR; ERROR CODE IN (A)
*
*       USES:   DEPENDS ON FUNCTION CALLED
*
*
*
START   EQU     *               HDOS COMES HERE EVERY TIME DRIVER IS CALLED
        CPI     DC.MAX          SEE IF REQUESTED FUNCTION IS UNDEFINED
        JNC     ILLEGAL         IF SO, TREAT AS ILLEGAL REQUEST
        CALL    $TBRA           LET THE TABLE BRANCH ROUTINE FIND CORRECT JUMP
        DB      READ-*          FNCTN 0= READ FROM DEVICE
        DB      WRITE-*         FNCTN 1= WRITE TO DEVICE
        DB      IGNORE-*        FNCTN 2= READ REGARDLESS
        DB      OPREAD-*        FNCTN 3= OPEN FOR READS
        DB      OPWRITE-*       FNCTN 4= OPEN FOR WRITES
        DB      ILLEGAL-*       FNCTN 5= OPEN FOR UPDATES
        DB      CLOSE-*         FNCTN 6= CLOSE CHANNEL TO DEVICE
        DB      CLOSE-*         FNCTN 7= ABORT; TREAT AS CLOSE
        DB      ILLEGAL-*       FNCTN 8= MOUNT DEVICE
        DB      LOAD-*          FNCTN 9= LOAD DEVICE
        DB      IGNORE-*        FNCTN 10= EXAMINE DEVICE READY STATUS
        STL     'DOCUMENTATION'
        EJECT
*       When HDOS calls up a device driver, the accumulator contains the
*       function that HDOS wants the driver to perform. In version 2.0
*       of HDOS, there are eleven possible functions. The clock driver
*       ignores some, treats others as illegal, and processes the rest.
*       The routine at label "Start" uses the supplied function number to
*       branch into the proper handler, via the ROM routine called
*       "Table Branch", or $TBRA for short. It is up to the individual
*       function processors to handle errors that occur within a processor.
*
*       In general, if a device driver returns with a set carry bit, HDOS
*       assumes that an error occurred. The single case here where an error
*       is not fatal is when reading from the clock. An "End of File"
*       error must happen, or HDOS will continue to read until you abort
*       the driver with control-C.
*
*       The "Load" processor is special in that it can only happen once.
*       When you type "Load CK:" from the HDOS command mode, HDOS will
*       check that the drivers address is in the device table, which was
*       built upon boot-up. If so, it spools the driver code into memory
*       and enters at the label called "Load". The load function places
*       a detour in HDOS's normal TICCNT interrupt path which causes control
*       to branch to the label called "Clock" every TICCNT (2 ms intervals).
*       The routine called "Clock" will determine whether 500 of these
*       TICCNTs have occurred, and if so, will update the time-of-day held
*       in the storage area labeled "Timebuf". The "Read" and "Write"
*       functions are the only functions that pull data out of the buffer
*       or stick data into the buffer.
*
*       After the Load Processor is finished patching internal vectors,
*       it returns control to HDOS. The operating system then flags the
*       device driver as being permanent in memory. Further "Load CK:"
*       commands are ignored by HDOS. If they were not, the system would
*       crash as soon as the "Load" processor trys to patch vectors that
*       have already been patched. No provision is made to "Unload" the
*       driver after it has been made permanent. The CPU overhead associated
*       with maintaining the correct time-of-day is minimal, and you will
*       not notice any degradation in CPU execution speed in a practical sense.
*
*       The "Illegal" function is branched to when HDOS issues an impossible
*       or meaningless command to the driver. The "Ignore" function does
*       just that...ignore. It is called when the command is meaningless,
*       yet otherwise harmless.
```

-- CONTINUED ON PAGE 18 --

885-1103 SEA BATTLE Game for HDOS $20.00
885-1211 SEA BATTLE Game for CP/M $20.00

Move over, Space Invaders! Here comes SEA BATTLE, a fast action graphics game for HDOS or CP/M on an H89 or H8 with H19. Imagine that you are the captain of a single high speed destroyer with two guns, and you face an armada of a huge carrier with fighters, bombers, and escorting submarines. You maneuver your ship into position to fire. Watch out! Those fighters and bombers are attacking! Your ship can take a few fighter hits, but one bomb and you're sunk! If you are sunk, your radioman has time to get off a quick SOS and the Admiral gives you another ship, but he only had 5 to start with, so be careful. Finally you manage to cripple a few of the fighters and bombers and score a hit on the carrier, but what's that on the horizon? A periscope! Quick! Your only defense against submarines is evasion, and you have to score 14 more hits on the carrier to sink it. And if you do sink it, his radioman sends an SOS as the ship sinks into the waves, and soon another carrier, armed more powerfully than before is sending waves of fighters and bombers after your ship!

This game features scoring, bonus points, and records your name and score if you score the highest. A freeze mode lets you answer the phone (or whatever) right in the middle of a game. SEA BATTLE was written by Victor A. Abell, author of Pinball and Reversi on HUG disk 885-1067, and requires a 32k system. The complete source code is included!

885-1022 HUG EDitor V 2.0 $15.00

The HUG Program Development Editor (ED), a fast character editor, has been improved. Version 2.0 offers all of the previous version's features plus the following enhancements. It now can delete (backspace) correctly through tabs and even through a carriage return or new-line (does an automatic control-R on the previous line if you delete a new-line). You can insert escape characters into the text and view them in two ways: as true escapes for graphic effects, etc., and as "^[" so that you can see where they are. A command has been added to put a line gauge on the 25th line of H19/H89's and to remove it. Version 2.0 is compatible with all mass storage devices supported by HDOS (SY:, DK:, or custom). It has a CP/M style printer toggle that lets you send any part of a file to a printer while you are editing. It will run on any version of HDOS since 1.5 and requires only a minimum system. Source code is included.

885-1210 HUG CP/M Editor $20.00

Now you can have the popular HUG Editor with all of the enhancements described above for CP/M. EDIT.COM is a fast character editor with single letter commands and automatic backup file creation. It can edit files of any size up to a disk full, and you can specify different input and output drives. It comes with source code and complete documentation. This version requires CP/M 2.0 or higher (ORG-0) and a minimum system.

885-1089 MACRO, CTOH, and Utilities $20.00

This disk is a new collection of HDOS utilities, and contains the following:

MACRO -- This is a macro pre-processor for the standard HDOS assembler. It provides full macro capabilities to the ASM user, including nested macros and nested definitions. It can link to ASM and pass a command line to it so the two appear to the user as one macro assembler.

CTOH -- This is the complement to H8COPY on disk 885-1207. It allows you to copy CP/M programs to HDOS (5-inch CP/M only). It runs under HDOS and can display the CP/M directory.

HTERM -- This program turns an H89 or H8 with H8-4 and H19 into a terminal for use with another computer. It allows all escape codes and normal control characters to be transmitted and received, and can send breaks. It can transmit files from disk to the external device and can store and save or print incoming data. Although it was designed as a terminal for other computers, it also can be used as a modem program for MicroNET, etc.

IHEX and IABS -- These programs convert files from .ABS format to Intel HEX format and vice versa. The Intel HEX format is ideal for sending machine code programs over a modem, because it provides load address and entry point information and a checksum for every 16 bytes of data. The IABS program reports any checksum errors by address when loading a HEX file so you can quickly locate errors and make a good file from two bad ones. IHEX lets you develop programs for CP/M with the HDOS assembler, convert them to HEX, then copy them over and LOAD them.

TAB2SPC -- This program was written in answer to those &%$#@!!! editors that replace spaces with tabs in text files. It replaces those tabs with the correct number of spaces so that the appearance of the file is maintained.

These programs require HDOS and at least 32k of memory.

HUG Products List

Part Number	Description		Selling Price
	CASSETTE SOFTWARE (H8 and H88)		
885-1008	Volume I Documentation and Program Listings (some for H11)		$ 9.00
885-1009	Tape I Cassette		$ 7.00
885-1012	Tape II BASIC Cassette		$ 9.00
885-1013	Volume II Documentation and Program Listings		$ 12.00
885-1014	Tape II ASM Cassette H8 Only		$ 9.00
885-1015	Volume III Documentation and Program Listings		$ 12.00
885-1026	Tape III Cassette		$ 9.00
885-1036	Tape IV Cassette		$ 9.00
885-1037	Volume IV Documentation and Program Listings		$ 12.00
885-1039	WISE on Cassette H8 Only		$ 9.00
885-1057	Tape V Cassette		$ 9.00
885-1058	Volume V Documentation and Program Listings		$ 12.00
	HDOS SOFTWARE (H8/H17 or H89 -- 5-inch only)		
	MISCELLANEOUS COLLECTIONS		
885-1024	Disk I H8/H89		$ 18.00
885-1032	Disk V H8/H89		$ 18.00
885-1044	Disk VI H8/H89		$ 18.00
885-1064	Disk IX H8/H89		$ 18.00
885-1066	Disk X H8/H89		$ 18.00
885-1069	Disk XIII Misc H8/H89		$ 18.00
	GAMES		
885-1010	Adventure Disk H8/H89		$ 10.00
885-1029	Disk II Games 1 H8/H89		$ 18.00
885-1030	Disk III Games 2 H8/H89		$ 18.00
885-1031	Disk IV Music H8 Only		$ 23.00
885-1067	Disk XI Graphic Games .ABS and B H BASIC (H19/H89)		$ 18.00
885-1068	Graphic Games (H19/H89)	*	$ 18.00
885-1088	Graphic Games (H19/H89)	*	$ 20.00
885-1093	Dungeons and Dragons Game Requires H89 or H8/H19	*	$ 20.00
885-1096	Action Games (H19/H89)	*	$ 20.00
885-1103	Sea Battle Game (H19/H89)		$ 20.00
	UTILITIES		
885-1019	Device Drivers (HDOS 1.6)		$ 10.00
885-1022	HUG Editor (ED) Disk H8/H89		$ 15.00
885-1025	Runoff Disk H8/H89		$ 35.00
885-1043	MODEM Heath to Heath H8/H89		$ 21.00
885-1050	M.C.S. Modem for H8/H89		$ 18.00
885-1060	Disk VII H8/H89 SUBMIT, CLIST, FDUMP, ABSDUMP, etc.		$ 18.00
885-1061	TMI Cassette to Disk H8 only		$ 18.00
885-1062	Disk VIII H8/H89 (2 disks) MEMTEST, DUP, DUMP, DSM		$ 25.00
885-1063	Floating Point Disk H8/H89		$ 18.00
885-1065	Fixed Point Package H8/H89		$ 18.00
885-1075	HDOS Support Package H8/H89		$ 60.00
885-1077	TXTCON/BASCON H8/H89		$ 18.00
885-1079	HDOS Page Editor		$ 25.00
885-1080	EDITX H8/H19/H89		$ 20.00
885-1082	Programs for Printers H8/H89		$ 20.00
885-1083	Disk XVI RECOVER, etc.		$ 20.00
885-1089	MACRO, CTOH, and misc Utilities		$ 20.00
885-1092	RDT Debugging Tool H8/H89		$ 30.00
885-1095	HUG SY: Device Driver HDOS 2.0		$ 30.00
885-1098	H8/HA-8-3 Color .ABS/.ASM		$ 20.00
885-1099	H8/HA-8-3 Color in Tiny Pascal		$ 20.00
	PROGRAMMING LANGUAGES		
885-1038	WISE on Disk H8/H89		$ 18.00
885-1042	PILOT H8/H89		$ 19.00
885-1059	FOCAL-8 H8/H89		$ 25.00
885-1078	HDOS Z80 Assembler		$ 25.00
885-1085	PILOT Documentation		$ 9.00
885-1086	Tiny Pascal H8/H89		$ 20.00
885-1094	HUG Fig-Forth H8/H89 2 Disks		$ 40.00
	BUSINESS, FINANCE AND EDUCATION		
885-1047	Stocks H8/H89		$ 18.00
885-1048	Personal Account H8/H89		$ 18.00
885-1049	Income Tax Records H8/H89		$ 18.00
885-1051	Payroll H8/H89		$ 50.00
885-1055	Inventory H8/H89	*	$ 30.00
885-1056	Mail List H8/H89	*	$ 30.00
885-1070	Disk XIV Home Finance H8/H89		$ 18.00
885-1071	SmBusPkg III 3 Disks H8/H19 or H89	*	$ 75.00
885-1091	Grade and Score Keeping	*	$ 30.00
885-1097	Educational Quiz Disk H89 or H8/H19	*	$ 20.00
	AMATEUR RADIO		
885-1023	RTTY Disk H8 Only		$ 22.00
885-1052	Morse8 Disk H8 Only		$ 18.00

* Means MBASIC is required

H11 SOFTWARE

885-1008	Volume I Documentation and Program Listings (some for H11)		$ 9.00
885-1033	HT-11 Disk I		$ 19.00

CP/M SOFTWARE (5-inch only)

885-1201	CP/M (TM) Volumes H1 and H2	%	$ 21.00
885-1202	CP/M Volumes 4 and 21-C	%%	$ 21.00
885-1203	CP/M Volumes 21-A and B	%%	$ 21.00
885-1204	CP/M Volumes 26/27-A and B	%%	$ 21.00
885-1205	CP/M Volumes 26/27-C and D	%%	$ 21.00
885-1206	CP/M Games Disk	%%	$ 21.00
885-1207	TERM and H8COPY		$ 20.00
885-1208	HUG Fig-Forth H8/H89 2 Disks		$ 40.00
885-1209	Dungeons and Dragons Game MBASIC and H89 or H8/H19		$ 20.00
885-1210	HUG Editor		$ 20.00
885-1211	Sea Battle Game for CP/M		$ 20.00

% Means CP/M 1.43 only (ORG-4200)
%% Means CP/M 1.43 or 2.2 (Heath)
Other CP/M disks are for 2.2

MISCELLANEOUS

885-0017	H8 Poster	$ 2.95
885-0018	H89 Poster	$ 2.95

(Vectored to page 31)

```
*       If either the "Read" or "Write" function is executed, they first
*       determine whether there is already file I/O in progress for the
*       respective function. If that is the case, an error is returned and
*       the second invocation of "Read" or "Write" is ignored. The "Close"
*       function resets both the "Read" and "Write" functions status flags.
*       Channels open on CK.DVD must be closed before new channels can be
*       opened on the clock driver.
        EJECT
*       Another point that bears explanation is the "Xfercnt" storage cell,
*       used during writes to the clocks buffer. Since we have no way of
*       knowing for sure how many bytes the caller wants to stick in the
*        clock buffer, we have to have a means of limiting insertions to a
*       specific number of bytes, in this case, eight. When Basic writes to
*       a file, it usually sends along enough null characters to make the
*       file write  multiples of 256 bytes. If we simply moved the supplied
*       bytes into the time buffer and beyond, we would likely crash whatever
*       was loaded in memory above the clock driver (usually another driver).
*       During writes to the time buffer, a running total of how many bytes
*       have already been sent to the buffer is kept in "Xfercnt". As soon
*       as eight bytes have been written, the remainder are simply eaten up
*       and not used. The "Xfercnt" is reset to eight when the clock driver
*       is once again opened for writes.
*
*       Another case for possible trouble is when the clock is open for
*       reads and the caller requests but one byte. If we let the caller
*       get the time out of the buffer in this fashion, chances are that
*       the time will change in between single byte transfers. That is why
*       the "Read" function processor checks to make certain that more
*       than eight bytes have been requested. For the sake of Basics "line
*       input" function, we always send a "newline" immediately after the
*       current time-of-day. If the caller requests more than nine bytes,
*       it is treated to a barrage of nulls until its appetite has been
*       satisfied, or until a full 256 bytes have been sent, whichever comes
*       first. Basic will ask for 256 bytes at a time, but user programs
*       or PIP will ask for varied amounts of bytes. If more than 256 bytes
*       is requested, CK.DVD will only return 256 bytes to the caller, and
*       will return an end-of-file error. If less than 257 bytes are requested,
*       the CK.DVD will return no error code.
*
*       At the end of this listing are two Benton Harbor Basic routines
*       that allow you to read or write to the clock. It is possible,
*       from the HDOS command mode, to do the same. If you want to see
*       what time it is, type "COPY TT:=CK:", or more simply, "PIP CK:".
*
*       If you want to set the clock from the command mode, type this:
*       COPY CK:=TT:      [carriage return]
*       12:34:56          [or whatever]
*       CTL-D
*
*       Note that anything beyond eight characters is ignored by the "Write"
*       processor inside the clock driver.
*
*       Experienced assembly language programmers will have no problem in
*       getting the current time from the driver, or updating the time from
*       their special programs. The driver is small as far as drivers go,
*       only about 300 bytes of memory are given up to it.
*
*       Note finally that ANY interrupt driven clock will lose time if the
*       interrupts are turned off for some reason, such as SY: accesses.
*       The value of the CAL factor has been chosen to yield accurate
*       timekeeping only if the SY:'s are not heavily used and the CPU's
*       clock frequency is exact.
        STL     'CK.DVD FUNCTION PROCESSORS'
        EJECT
ILLEGAL EQU     *               COME HERE IF ILLEGAL DRIVER FUNCTION REQUESTED
        MVI     A,EC.ILR        PUT THE "ILLEGAL REQUEST" ERROR CODE IN (A)
        STC                     SO HDOS KNOWS AN ERROR OCCURRED
        RET                     TO WHOMEVER CALLED THE DRIVER
        SPACE   3,9
IGNORE  EQU     *               COME HERE TO IGNORE A REQUESTED FUNCTION
        XRA     A               CLEAR THE CARRY BIT
```

```
        RET                     TO WHOMEVER CALLED THE DRIVER
        SPACE   3,9
LOAD    EQU     *               COME HERE WHEN USER TYPES "LOAD CK:"
        LHLD    UIVEC+1         GET THE HDOS "TICCNT" VECTOR
        SHLD    CLKRET+1        INSTALL AT END OF OUR CLOCK ROUTINE
        LXI     H,CLOCK         GET OUR CLOCK'S START ADDRESS
        SHLD    UIVEC+1         REPLACE THE HDOS "TICCNT" VECTOR
        XRA     A               CLEAR THE CARRY BIT
        RET                     RETURN, SHOWING NO ERROR
        SPACE   3,9
OPREAD  EQU     *               FLAG TO DEVICE THAT A CHANNEL IS OPEN ON IT
        LDA     RSTAT           FETCH OUR READ STATUS FLAG
        ORA     A               SEE IF ALREADY OPEN FOR READS
        JZ      OPREAD1         NOT OPEN, SO SKIP NEXT
        MVI     A,EC.FAO        PUT "FILE ALREADY OPEN" ERROR CODE IN (A)
        STC                     TELL HDOS WE HAVE AN ERROR
        RET
OPREAD1 MVI     A,-1            PUT A 0FFH IN READ STATUS FLAG CELL
        STA     RSTAT           THIS FLAGS THE DEVICE NOW OPEN
        XRA     A               CLEAR CARRY BIT
        RET
        SPACE   3,9
OPWRITE EQU     *               FLAG TO DEVICE THAT A CHANNEL IS OPEN ON IT
        LDA     WSTAT           FETCH OUR WRITE STATUS FLAG
        ORA     A               SEE IF ALREADY OPEN FOR WRITES
        JZ      OPWRIT1         NOT OPEN, SO SKIP NEXT
        MVI     A,EC.FAO        PUT "FILE ALREADY OPEN" ERROR CODE IN (A)
        STC                     TELL HDOS WE HAVE AN ERROR
        RET
OPWRIT1 MVI     A,-1            PUT A 0FFH IN WRITE STATUS FLAG CELL
        STA     WSTAT           THIS FLAGS THE DEVICE NOW OPEN
        MVI     A,8             PUT MAXIMUM BYTE XFER COUNT IN HOLD PLACE
        STA     XFERCNT
        LXI     H,TEMPBUF       POINT AT FIRST ADDRESS IN TEMPORARY BUFFER
        SHLD    BUFPTR          SAVE IT IN POINTER HOLDING PLACE
        XRA     A               CLEAR CARRY BIT
        RET
        SPACE   3,9
CLOSE   EQU     *               FLAG DEVICE DISCONNECTED FROM ACTIVE CHANNELS
        XRA     A               ZERO (A)
        STA     RSTAT           RESET "OPEN FOR READ" FLAG, IF SET
        STA     WSTAT           RESET "OPEN FOR WRITE" FLAG, IF SET
        RET                     WITH A CLEAR CARRY BIT
        SPACE   3,9
MOVE    EQU     *               MOVE (BC) BYTES FROM ((HL)) TO ((DE))
        MOV     A,B             SEE IF BYTE COUNTER AT ZERO
        ORA     C
        RZ                      ALL FINISHED; (DE)=NEXT *TO* ADDRESS
        MOV     A,M             FETCH BYTE TO BE MOVED
        STAX    D               WRITE BYTE VIA (DE)
        INX     H               BUMP *FROM* LOCATION
        INX     D               BUMP *TO* POINTER
        DCX     B               DECREMENT BYTE COUNTER
        JMP     MOVE            LOOP UNTIL (BC) DECREMENTED TO ZERO
        STL     'CK.DVD WRITE PROCESSOR'
        EJECT
***     CK.DVD  WRITE PROCESSOR
*
*       ENTRY:  (BC)    = BYTE COUNT; MUST BE EQUAL TO OR GREATER THAN EIGHT
*               (DE)    = BUFFER ADDRESS; WHERE NEW TIME STRING IS COMING FROM
*
*       EXIT:   (PSW)   = CARRY CLEAR IF NO ERROR
*                       = CARRY SET IF ERROR; CODE IN (A)
*
*       USES:   ALL
*
*
*
WRITE   EQU     *               COME HERE WHEN CALLER WANTS TO WRITE TO DVD
        LDA     WSTAT           FIRST, SEE IF WE'RE OPEN FOR WRITES
        ORA     A
```

```
        MVI     A,EC.FNO        PREPARE OURSELVES FOR "CHANNEL NOT OPEN" ERROR
        STC
        RZ                      RETURN IF CHANNEL WAS NOT PREVIOUSLY OPENED
WRITE1  MOV     A,B             SEE IF BYTE COUNT AT ZERO
        ORA     C
        RZ                      RETURN WITH CLEAR CARRY; TIME IN "TEMPBUF"
        LDA     XFERCNT         SEE HOW MANY BYTES WE'VE WRITTEN SO FAR
        ORA     A               SEE IF MAXIMUM NUMBER WRITTEN INTO "TEMPBUF"
        JZ      WRITE2          EIGHT BYTES WRITTEN; NOW MOVE TO REAL BUFFER
        DCR     A               ADJUST TRANSFER COUNT
        STA     XFERCNT         UPDATE TRANSFER COUNT
        LHLD    BUFPTR          GET POINTER INTO "TEMPBUF"
        LDAX    D               FETCH CHARACTER TO BE WRITTEN INTO "TEMPBUF"
        MOV     M,A             PLACE CHARACTER IN "TEMPBUF"
        INX     H               POINT TO NEXT POSITION IN BUFFER
        SHLD    BUFPTR          UPDATE POINTER INTO "TEMPBUF"
        INX     D               BUMP POINTER INTO SOURCE FIELD
        DCX     B               DECREMENT BYTE COUNTER
        JMP     WRITE1          TRY AND TRANSFER SOME MORE CHARACTERS
WRITE2  LXI     H,TEMPBUF       SOURCE FOR MOVE
        LXI     D,TIMEBUF       DESTINATION OF MOVE
        LXI     B,8             HOW MANY TO MOVE
        DI                      DON'T WANT "TICCNTS" TO MESS US UP !
        CALL    MOVE            PUT TIME IN THE REAL "TIMEBUF"
        EI                      ENABLE ALL INTERRUPTS
        RET                     CARRY CLEARED; (BC)=ZERO
        SPACE   3
*       This looks kludgy, first writing the time into a temporary buffer
*       then putting it into the actual time buffer, but if a user program
*       for some reason only transfers one byte at a time, we risk the
*       chance of a TICCNT updating the actual buffer while we are writing
*       into it. The time is moved into the actual buffer only after eight
*       bytes have been placed into the temporary buffer.
        STL     'CK.DVD READ PROCESSOR'
        EJECT
***     CK.DVD  READ PROCESSOR
*
*       ENTRY:  (BC)    = BYTES REQUESTED; MUST BE AT LEAST NINE
*               (DE)    = BUFFER ADDRESS; WHERE TIME STRING GETS PLACED
*
*       EXIT:   (PSW)   = CARRY CLEAR IF 256 OR LESS BYTES REQUESTED;
*                         ELSE, RETURN WITH END-OF-FILE ERROR. IF LESS
*                         THAN NINE BYTES REQUESTED, EXIT THROUGH "ILLEGAL".
*               (BC)    = UNUSED BYTE COUNT (NORMALLY ZERO)
*               (DE)    = ADDRESS OF NEXT BYTE TO BE READ (IF ANY)
*
*       USES:   ALL
*
*
*
READ    EQU     *               COME HERE WHEN CALLER WANTS TO READ TIME OF DAY
        LDA     RSTAT           FIRST, SEE IF WE'RE OPEN FOR READS
        ORA     A
        MVI     A,EC.FNO        PREPARE OURSELVES FOR "CHANNEL NOT OPEN" ERROR
        STC
        RZ                      RETURN IF CHANNEL NOT PREVIOUSLY OPENED
        MOV     A,C             CHECK FOR REQUEST OF AT LEAST NINE BYTES
        CPI     9
        JNC     READ1           IF (C) GREATER THAN OR EQUAL TO NINE
        MOV     A,B             SEE IF A MULTIPLE OF 256 BYTES REQUESTED
        ORA     A
        JZ      ILLEGAL         CAN'T SUPPLY LESS THAN NINE BYTES !
READ1   PUSH    B               SAVE BYTE COUNTER
        LXI     B,9             FORCE DEFAULT MOVE OF NINE BYTES
        LXI     H,TIMEBUF       WHERE TIME STRING IS COMING FROM
        DI                      IN CASE A "TICCNT" BOTCHES THINGS UP !
        CALL    MOVE            GIVE THE STRING TO THE CALLER OF THE DVD
        EI                      TURN ALL INTERRUPTS BACK ON
        POP     B               RESTORE BYTE COUNTER
        LXI     H,-9            ACCOUNT FOR BYTES ALREADY SENT TO CALLER
        DAD     B               (HL) EQUAL TO NUMBER OF BYTES TO PAD OUT
```

```
        MOV     B,H             PUT UNUSED BYTE COUNT IN (BC)
        MOV     C,L
        MVI     L,9             ACCOUNT FOR BYTES ALREADY SENT TO CALLER
READ2   MOV     A,B             SEE IF DONE PADDING OUT THE BUFFER
        ORA     C               (BC)=ZERO IF DONE
        RZ                      FINISHED, SO EXIT WITH NO ERROR
        XRA     A               ZERO (A)
        STAX    D               WRITE A NULL INTO THE BUFFER
        DCX     B               DECREMENT BYTE COUNTER
        INX     D               BUMP BUFFER POINTER
        INR     L               BUMP THE MODULO 256 BYTE COUNTER
        JNZ     READ2           CONTINUE READING UNTIL (L)=0 OR (BC)=0
        MOV     A,B             SEE IF MORE THAN 256 BYTES REQUESTED
        ORA     C
        RZ                      WAS NOT; RETURN WITH NO ERROR
        MVI     A,EC.EOF        ELSE; RETURN WITH EOF ERROR
        STC
        RET
        STL     'CK.DVD TICCNT ENTRY POINT'
        EJECT
***     CK.DVD  TICCNT ENTRY POINT
*
*       ENTRY:  EVERY TICCNT (2 MILLISECOND INTERVALS)
*
*       EXIT:   TO NORMAL HDOS TICCNT PROCESSOR
*               TIME IN "TIMEBUF" UPDATED IF A NEW SECOND
*
*       USES:   ALL; HDOS SAVES IT'S OWN REGISTERS
*
*
*
CLOCK   EQU     *               COME HERE EVERY "TICCNT" AND UPDATE THE TIME
        LHLD    TICKS           RETRIEVE OUR OWN PRIVATE "TICKER"
        INX     H               ADD ONE MORE TICK
        SHLD    TICKS           UPDATE OUR PRIVATE "TICKER"
        LXI     B,CAL           GET CALIBRATION FACTOR
        DAD     B               WILL CREATE A (CY) IF "TICKS"=500 DECIMAL
        JNC     CLKRET          NOT TIME FOR A NEW SECOND, SO RETURN
        SHLD    TICKS           (HL) WAS ZERO, RESET PRIVATE "TICKER"
        MVI     C,'0'           PUT AN ASCII ZERO IN REGISTER (C)
INRS    LXI     H,TIMEBUF+7     POINT AT UNITS SECONDS
        INR     M               BUMP IT
        MOV     A,M
        CPI     '9'+1           OVERFLOW ?
        JM      CLKRET          NO, SO RETURN
        MOV     M,C             RESET UNITS SECONDS
INRTS   DCX     H               POINT AT TENS SECONDS
        INR     M               BUMP IT
        MOV     A,M
        CPI     '6'             OVERFLOW ?
        JM      CLKRET          NO, SO RETURN
        MOV     M,C             RESET TENS SECONDS
INRM    DCX     H               POINT AT THE COLON
        DCX     H               POINT AT UNITS MINUTES
        INR     M               BUMP IT
        MOV     A,M
        CPI     '9'+1           OVERFLOW ?
        JM      CLKRET          NO, SO RETURN
        MOV     M,C             RESET UNITS MINUTES
INRTM   DCX     H               POINT AT TENS MINUTES
        INR     M               BUMP IT
        MOV     A,M
        CPI     '6'             OVERFLOW ?
        JM      CLKRET          NO, SO RETURN
        MOV     M,C             RESET TENS MINUTES
INRH    DCX     H               POINT AT THE COLON
        DCX     H               POINT AT UNITS HOURS
        INR     M               BUMP IT
        MOV     A,M
        CPI     '4'             OVERFLOW ?
        JM      CLKRET          NO, SO RETURN
```

```
INRH1   DCX     H               POINT AT TENS HOURS
        MOV     A,M
        CPI     '2'             IS IT 24 AND NOT 14 OR 04 ?
        JM      INRH2           STILL MORE TO CHECK
        MOV     M,C             RESET TENS HOURS
        INX     H               POINT AT UNITS HOURS
        MOV     M,C             RESET UNITS HOURS
        JMP     CLKRET          (IT WAS MIDNIGHT)
INRH2   INX     H               POINT AT UNITS HOURS
        MOV     A,M
        CPI     '9'+1           OVERFLOW ?
        JM      CLKRET          NO, SO RETURN
        MOV     M,C             RESET UNITS HOURS
INRTH   DCX     H               POINT AT TENS HOURS
        INR     M               BUMP IT
        SPACE   3,9
CLKRET  EQU     *               NOW CONTINUE ON WITH HDOS CLOCK INT. ROUTINE
        JMP     0               NEW ADDRESS INSTALLED AT "LOAD" TIME
        SPACE   3,9
***     CK.DVD  STORAGE AREAS
*
*
TICKS   DW      0               THIS IS OUR PRIVATE "TICK COUNTER"
TIMEBUF DB      '00:00:00'      CORRECT TIME OF DAY MAINTAINED HERE, IN ASCII
BUFEND  DB      NL              TIME STRING ALWAYS TERMINATED WITH A "NEWLINE"
RSTAT   DB      0               HOLD PLACE FOR "OPEN FOR READ" STATUS FLAG
WSTAT   DB      0               HOLD PLACE FOR "OPEN FOR WRITE" STATUS FLAG
XFERCNT  DB      0               HOLD PLACE FOR COUNT OF BYTES TRANSFERED
BUFPTR  DW      0               HOLD PLACE FOR POINTER INTO "TEMPBUF"
TEMPBUF DB      '00:00:00'      TEMPORARY BUFFER DURING "WRITE" PROCESSING
        SPACE   3,9
*       Another note for the astute programmer:
*
*       This driver, as do   all Heath drivers except the disc drivers, will
*       allow you to read or write any number of bytes you want. Contrary to
*       what the System Programmer's Guide says, I/O to non-storage devices
*       need not be in multiples of 256 bytes. Realizing this fact makes it
*       easier to get data one character at a time from a device or into a
*       device. Programs will be more efficient, memory wise, if they do not
*       have to maintain a 256 byte buffer just to handle small I/O tasks.
*       Be aware that this may not be the case with drivers not coming from
*       Heath Company.
        STL     'BENTON HARBOR BASIC EXAMPLES'
        EJECT
*       00010 REM       READ CLOCK FROM BENTON HARBOR BASIC
*       00020 REM       11-JUN-81   DALE LAMM
*       00030 OPEN "CK:" FOR READ AS FILE #1
*       00040 INPUT #1,;T$
*       00050 PRINT T$
*       00060 CLOSE #1
*       00070 GOTO 10
*
*       The above merely opens a channel on the clock driver for reads,
*       then reads the current time-of-day, and then closes the channel.
*       The program repeats until the control-C key is struck.
        SPACE   6
*       00010 REM       WRITE NEW TIME TO CLOCK FROM BENTON HARBOR BASIC
*       00020 REM       11-JUN-81   DALE LAMM
*       00030 OPEN "CK:" FOR WRITE AS FILE #1
*       00040 INPUT "WHAT TIME IS IT NOW ?  ";T$
*       00050 PRINT #1,T$
*       00060 CLOSE #1
*       00070 OPEN "CK:" FOR READ AS FILE #1
*       00080 INPUT #1,;T$
*       00090 PRINT "VERIFYING... CLOCK NOW READS "T$
*       00100 CLOSE #1
*       00110 END
*
*       The example above demonstrates how a new time-of-day may be put
*       into the clocks buffer. The clock remains running, and keeps
*       time using the just installed time string as the base.
```

```
*       Note that no error checking is done by the actual clock driver.
*       Whatever characters are inserted into the clocks buffer will be
*       the new base for timekeeping. Likewise, you may use whatever
*       character suits your fancy to delimit the hours, minutes, and
*       seconds.
*
*       The newly entered time-of-day is read back to the user for
*       verification. Only the first eight characters in T$ are loaded
*       into the drivers buffer. There must be at least eight new
*       characters written into the buffer, else, the new time-of-day
*       will be meaningless. Since Basic pads out writes to a file with
*       nulls, it is not possible to use Basic to update only the first
*       two digits in the clock drivers buffer.
        STL     'PIC TABLE'
        EJECT
        LON     G               TURN ON THE PIC TABLE LISTER
        END
```

A Review of Small Business Package III

As a general overview for the SBPIII, the most important single factor is to know and understand what this package is NOT capable of, as well as, what it IS capable of doing. Do not expect more than what it can do or you will just get yourself into a position where you will be disappointed in the package as a whole.

First, realize this package is not a "General Journal". You are responsible for keeping your own Journal entries. Second, please understand that the Cost of Goods and Expense Ledger is just that. It is for posting to cost of goods accounts and expense accounts, as payments are made. It is not a General Ledger.

Second, it is not an inventory package. The SBPIII does not keep a running total of your inventory through the accounts receivable and accounts payable. The accounts receivable and accounts payable programs will maintain running totals of cash, A/R, and A/P as affected by these entries.

Thirdly, this package does include the necessary reports, that most small business utilize. The SBPIII prints several reports that relate to A/R and A/P, of which includes a Sales Baragraph. This package uses the running totals from A/R and A/P to help create and print a Profit and Loss Statement and a Balance Sheet.

Finally, the example accounts, payments, sales, posting and reports are all fictitious by creating a "make-believe" business. The accounts and related data have ALL been entered to show you specific examples of output, adjustments, payments, and any errors you are likely to run into. Even though the accounts are made up, they were written to show you what a typical business may encounter. We have found already that because we created a "make-believe" business, we were not able to cover all areas of the package and thus have come across some minor problems, for which we have included corrections.

This SBPIII is based on a "textbook" form of accounting i.e. Double-entry. Please note the word based. There is no place for actually showing double-entries. The double-entry is implied, e.g. when a Payment is issued for a particular A/R, CASH would be debited and A/R would be credited for the amount of the Payment. All that is entered, however, is the "*PAYMENT" and the amount paid . . No double-entry is made. Due to this procedure and the invoice number system, the SBPIII package will be compatible with "Vouchers", if that relates to your business.

Again, do not expect more than it can do and you will have little difficulty using this package. As you do use it, you will find there are many "neat" things that you can do with it that can not be explained in the documentation.

We have had a couple of calls already, that the SBPIII is not what they want and want to send it back. Look at this realistically!! There are big businesses that spend tens of thousands of dollars on their business packages and have full-time programmers to work out any "bugs". We do not promote that this package will work for every small business and we do not project that it is bug

free. My point is, this is a nice little package and it will work as written and documented, but if it is not what you want . . DON'T BUY IT!! We have included enough of an "overview" that you should know whether you can use this package in your business.

We will support the SBPIII as written and documented. Many of you make your own modifications to fit your business . . this is great, but we cannot support your modifications. We just cannot take the time because we would not fulfill our other obligations.

Thank you for your understanding.

The following is the Main Menu of the SBPIII:

```
  1 = Return to STARTUP MENU
  2 = Issue Invoices -or- Credit Memos
  3 = Accounts Receivable Maintenance
  4 = Accounts Receivable Reports
  5 = Print Statements
  6 = Print Sales Report
  7 = Sales Bargraph
  8 = Mailing Labels
  9 = Cost of Goods and Expense Ledger
 10 = Accounts Payable
 11 = Print Profit and Loss Statement
 12 = Print Balance Sheet

      Enter Selection No. (1 to 12) <END>
```

Corrections to SBPIII

885-1071

Modifications to the Small Business Package III which should be done to the following programs if they are version 06.23.81. The version number may be found in line number 10 of all programs.

```
INV1.BAS        PROGRAM Disk
    1240 PRINT"       Taxable (Y or N) <Y>";:GOSUB 1720:TX$=A$:PRINT A$;
    1630 IF A$=CHR$(13) THEN ZS$="N"

POST.BAS        PROGRAM Disk
      30 CLEAR 5000:WIDTH 255:DEFINT A-Z:DEFSNG B,T:ON ERROR GOTO 2580
    1430 NF$=X$:GOTO 1520

SALES.BAS       PROGRAM Disk
     250 IF A$="" THEN A$=DG$:GOTO 260
     255 DG$=A$

CONVERT.BAS     PROGRAM Disk
     100 SY$=INPUT$(1):PRINT :IF SY$="1" THEN SY$="SY0:":SZ$="SY0:":GOTO 130
     555 IF DS$="0" AND INV=0 THEN 590
    1165 IF ERR=65 AND ERL=990 THEN RESUME 950
     620 IF YN$="Y" OR YN$="y" THEN 630 ELSE 650

INV2.BAS        DATA Disk I
     100 PRINT :SX#=0:TS#=0:'    ** Zero Sales Tax & Sales Sub-Total **
     470 FOR S9=1 to 14:PRINT #2,:NEXT S9:GOTO 750:'  Skip spaces for Credit Memo
     705 IF TX$<>"TX" THEN PRINT #2,:PRINT #2,
     810 IF PS$="S" THEN GOSUB 1010:GOTO 870
     830 IF LEFT$(Q2$,1)="Y" THEN 870

BARGRAPH.BAS        DATA Disk I
     155 IF A$="1" THEN 170
     250 INPUT #1,SFM#:SFM#(X)=SFM#:Y1#=Y1#+SFM#(X):CLOSE #1
     255 NEXT X
     940 IF ERL=240 THEN RESUME 255

LEDGER.BAS          DATA Disk II
    1650 AB=AA
    1675 R2=R1\2:R3#=R1/2:IF R2=R3# THEN R2=0 ELSE R2=1
    1680 FOR R=1 TO R1\2+R2
    1685 IF R2=1 AND R=R1\2+1 THEN 1720
```

All changes or additions are underlined. Version numbers of the programs being modified should be changed to Version 07.20.81. The version is located in line #10 of all programs.

:GK:

BUGGIN' HUG

Hi Bob:

Enclosed is an example of USRDEF functions in MBASIC using the H-11 routines given on page 7, Issue 18 of REMark. Some of your readers might have difficulty in translating from H-11 BASIC to MBASIC.

```
10 REM DEFINE.BAS  usr defined functions in MBASIC (PG 7, ISSUE 18 OF REMark)
20 '
30 CLEAR 2000
40 READ Z:' Z=total number of items to be read, in this case 5
50 FOR N=1 TO Z:READ L(N),C(N),S$(N):NEXT N:' read each item of 3 into array
60 DATA 5,40,40,"  NAME... ",42,40,"  ADDRESS... "
70 DATA 44,40,"  CITY... ",46,40,"  STATE... ",48,40,"  ZIP... "
80 DEF FNX$(L,C,S$)=CHR$(27)+"Y"+CHR$(L)+CHR$(C)+S$:'  define the "function"
90 PRINT CHR$(27)+"E":'  clear the screen
100 FOR N=1 TO Z: PRINT FNX$(L(N),C(N),S$(N)):NEXT N:'  print the function
110 END
120 '
130 ' Remarks -
140 '
150 ' Note line 60.  Line 40 reads first data item (5) into Z.  Then line 50
160 ' reads the 3 parts of each item into array.  Since the "5" was already
170 ' read into Z, the first time thru the for-next loop puts "40,40,name" into
180 ' array, next time thru puts "42,40,address" into array etc.
```

Sincerely,
William (DOC) Campbell, M.D.
885 SmithBridge Road
Glen Mills, PA 19342

Dear HUG,

Bob Thomas' two programs for screen formatting in Issue 10 of REMark are great. For my own use I made two minor changes. Since we are running MBASIC which allows descriptive variable names, the various codes were assigned names to match those given in the H-19/89 operations manual so they are easier to recognize. Also, the list was extended to include entering and leaving graphics mode and disabling the 25th line.

Bob's use of the DEF FN statement for X,Y cursor positioning started the old gears grinding. In setting up screens for data entry, it is useful to define an area on the screen in reverse video to let the user know where the next data is supposed to go and how long it can be. This requires placing the cursor at the right position, entering reverse video, printing a string of "light" spaces the appropriate length, and returning the cursor to the beginning of the input area. I designed a second function which calls Bob's original function (!) and performs this task. Both the original function and the second calling function are presented here.

```
DEF FN C$(C1,C2) = Y$+CHR$(C1+31)+CHR$(C2+31)

DEF FN B$(X,Y,L) = P$+FNC$(X,Y)+J$+STRING$(L," ")+K$
```

In a program, it would be used like this:

```
PRINT FNB$(12,40,10);:REM SET 10-CHAR BLOCK AT LINE 12 POS 40
LINE INPUT "";A$:REM GET DATA FROM BLOCK
PRINT Q$:REM TURN OFF REVERSE VIDEO
```

This method has a lot of neat advantages. (1) It leaves no doubt in the user's mind concerning the positioning and length of data allowed. (2) Data is entered in reverse video -- the lighted area does not disappear when the user types in data. (3) The delete key restores the area in reverse video. (4) The user cannot move to the left of the originally defined cursor position with the delete key! (5) Prompts can be placed within the defined area through normal use of the INPUT or LINE INPUT messages imbedded in the command. (6) By alternating PRINT FNB$ and LINE INPUT statements, many different fields of related information can be entered from a "whole-screen" format. Used in conjunction with Bob's original FNC$, this provides a very powerful tool for "human interfacing".

Jim Ingram
804 North C Street
Broken Bow, NE 68822

Dear HUG,

In reading REMark, I am continually impressed by the knowledge and ability of the users who submit articles to you. There are apparently a lot of hobbyists who are really sharp. I am new to computers and the art of programming them, so I have a good bit of work, and fun, ahead of me.

I ordered UCSD PASCAL to run on my H-8, and really like the system and the language. It has a lot of capabilities which fit well with the things which I need to do. However, in comparison to BASIC, I have not found much software available in PASCAL, either for doing work or just examining as part of the learning process. Therefore, I would like to submit the following program so that HUG will have a start on a UCSD PASCAL Library. Hopefully others will make improvements to my program and submit some of their own as well.

One of the nicest parts of the UCSD PASCAL system is the E)ditor , which I found to be easy to learn how to use. While I haven't mastered everything it can do, it has done everything which I have needed done so far, except that I needed a program to correctly arrange and page text which I wanted to write out on my printer. I have therefore written the enclosed program which will do the following:

1. Transfer a text file on disk to the printer
2. Page the output, numbering the pages
3. Allow you to choose the number of blank spaces at the top of page one, if your printer will allow you to use a sheet of letterhead paper
4. Single-,Double-,or Triple-space the output
5. Offer the user a chance to check that the paper is properly in the printer before actually printing
6. Do a little useless foolishness on the CRT
7. Use variable and file names which hopefully help in demonstrating the workflow of the program so that someone else who is also learning may be better able to understand what happens in the program.

As a novice, I appreciate what HUG does for its users. Keep up the good work.

Steve Hagins
P.O. Box 1260
Enterprise, AL 36331

PS. UCSD PASCAL allows the user to shift the keypad for use with the editor as part of a file called SYSTEM.STARTUP. If the individual user has shifted his keypad, and if he wants to enter instructions to the program PrinText through the keypad, he will have to modify the program with instructions to "shift" and "unshift":

```
    WRITE(CHR(27){ESC},CHR(117){u}); {unshifts the keypad}
    WRITE(CHR(27){ESC},CHR(116){t}); {shifts the keypad}
PROGRAM PrinText;

 {BY STEVE HAGINS, 113 REDWING DRIVE, ENTERPRISE, AL 36330}

   CONST          PAGEFULL = 60;
                  BLANK    = '  ';

   VAR            DISKFILE,OUTFILE   : INTERACTIVE; {FILES}
                  SPACING : 1..3;
                  LINENUM,PAGENUM,I : INTERGER;
                  CH        : CHAR; {ELEMENTS OF THE FILES}
                  FILENAME  :STRING; {DIRECTORY NAME, INCLUDING UNIT}

PROCEDURE INTRO;
 BEGIN
 WRITELN(BLANK:10,'PRINT PROGRAM');WRITELN;
 WRITELN('THIS PROGRAM WILL WRITE TO THE PRINTER A TEXTFILE STORED ON');
 WRITELN('DISK. IT WILL ALSO PAGE THE OUTPUT, NUMBER THE PAGES, AND');
 WRITELN('DOUBLE OR TRIPLE SPACE THE OUTPUT.  YOU MAY ALSO SELECT THE');
 WRITELN('NUMBER OF LINES AT THE TOP OF THE FIRST PAGE,  IN CASE YOU ARE');
 WRITELN('PRINTING ON A SHEET OF LETTERHEAD STATIONERY.');
 WRITELN;WRITELN('PLEASE ENTER THE NAME OF THE FILE TO BE PRINTED.');
 WRITELN('FOR EXAMPLE, "LETTER.TEXT", OR "#5:SUPER.TEXT"');
 WRITE('FILENAME -->');
 READLN(LINENUM);WRITELN;
 PAGENUM := 1;
 END;

BEGIN
INTRO;
 RESET(DISKFILE,FILENAME);
 REWRITE(OUTFILE,'#6:FANTASTIC.TEXT');
 PAGE(OUTFILE);
 WRITELN('THE SYSTEM HAS ENGAGED THE PRINTER. PLEASE CHECK TO SE THAT');
 WRITELN('THE PAPER IS AT THE TOP OF THE PAGE.  DO NOT--ADJUST THE PAPER');
 WRITELN('MANUALLY UNTIL YOU HAVE TURNED THE PRINTER OFF.');WRITELN;
 WRITE('HOW DO YOU WANT THE OUTPUT SPACED? 1-2-3? --->,);
 READLN(SPACING);
 FOR I := 1 TO LINENUM DO WRITELN(OUTFILE);
 READ(DISKFILE,CH);
 WRITELN;WRITE('HERE I GO');
 WHILE NOT EOF(DISKFILE) DO
   BEGIN
     WHILE NOT EOLN (DISKFILE) DO
       WRITE(OUTFILE,CH);
       READ(DISKFILE,CH);
       END;
   IF EOLN(DISKFILE);
     BEGIN
       READLN(DISKFILE);
       FOR I := 1 TO SPACING DO WRITELN(OUTFILE);
       LINENUM:= LINENUM + SPACING;
       WRITE('.');
   IF LINENUM > PAGEFULL THEN
     BEGIN
       PAGENUM := PAGENUM + 1;
       PAGE(OUTFILE);
       FOR I := 1 TO 4 DO WRITELN(OUTFILE);
       WRITELN(OUTFILE,BLANK: 35, 'PAGE ',PAGENUM:3);
       FOR I := 1 TO 4 DO WRITELN(OUTFILE);
       LINENUM:= 10;
     END;
   END;
 END;
 CLOSE(OUTFILE,LOCK);
 WRITELN;WRITELN('I AM NOW FINISHED.  YOU ARE WELCOME.');
END.
```

Dear Terry,

I have modified the SBPIII Ledger program so that it does not print continuously but separates into pages when you have more that 10 expense items in the ledger. Six lines of code were required.

```
1065 PRINT CS$:FOR C8=1 to 3:PRINT #2,:NEXT C8
1085 NL=NL+5
1255 NL=NL+3
1315 NL=NL+1
1361 NL=NL+2
1362 IF NL>53 THEN PRINT:FOR C8=1 TO 66-NL:PRINT #2,:NEXT C8:NL=0
```

I hope this modification will be of interest to someone.

Thanks for your wonderful help recently.

Russ Kennedy
Rye, NY

Using an Extended Capacity Drive as SY0:

This article will explain the procedures involved to use an extended capacity drive as SY0: with the HUG SY: device driver (885-1095). By extended capacity, I mean a double sided and/or 80 track 5-inch disk drive. The HUG SY: device driver supports such drives, but setting up one as the system drive can be a bit tricky.

First, I should point out that if you have an H89, you should not use an extended capacity drive in the H89 cabinet (because of the signal-to-noise ratio there), so you will have to use one of your outboard drives (H77/H87) as SY0:. The difficulty in using an extended capacity drive as SY0: comes from the fact that when you initialize a disk with the new SY: on your system disk, information is written in the boot track about the drives. This information is taken from SY: and is what you specify with the SET command. Remember that a SET option does not take effect until you reboot the disk, so you can SET SY0: to the parameters of the new drive, but you will not be able to re-boot your system disk in SY0:. The solution is to boot the disk in another drive.

I will present two procedures for using an extended capacity drive in SY0:. The first method requires two disk drives (one standard, the other extended capacity) and the Heath ORG-0 ROM set (if you have an H89) or the Extended Configuration Option (if you have an H8). If you have a third drive, it will not be used in this procedure. Now carefully do the following steps.

1. Configure the drives so that the standard drive is SY0: and the extended capacity drive is SY1:. By "configure", I mean set up the programming jumpers on the drive PC boards.

2. Prepare a standard size system disk with the new SY: on it, along with INIT, SYSGEN, and all files required to do a SYSGEN. Since we will be using two drives in the SYSGEN, ONECOPY will not be used, but SYSGEN will still look for it and crash if it is not there. You can rename another file to ONECOPY.ABS to fool it.

3. Boot up on the system disk you have prepared, and SET SY0: to the parameters of the extended capacity drive (number of sides, tracks, and the step rate).

4. Exit HDOS with BYE.

5. Turn off the computer and drives and swap the programming jumpers in your two drives so that the extended capacity drive is SY0: and the standard drive is SY1:.

6. Turn on the computer and drives and insert the system disk into the standard drive. If you have an H89, type the letter B and the number 1 and hit RETURN. If you have an H8, your Extended Configuration Option should be set up with the H17 as the primary device. Enter (on the front panel) REG AF ALTER 001000 ALTER, and hit GO. The disk in SY1: should boot up normally.

7. Insert a good blank disk into the extended capacity drive and run INIT.

Use either the standard INIT or INITAUTO supplied with the HUG SY: disk. When it asks for a device, enter SY2:. Answer YES to "Double sided" if you set SY0: to two sides in step 3. Answer YES to "Media check" just to be sure everything will be OK later on.

8. After the initialization is complete, type two control-D's and hit RETURN. Hit RETURN again to reboot your system disk.

9. Give the command SYSGEN *.*, and when asked for a device, enter SY2:. The disk in the extended drive will be SYSGENed. When the SYSGEN is completed, reset your computer and try to boot up on the new disk. It should work normally. You can now use it as the source disk for initializing and SYSGENing other extended capacity disks. You can replace your other drive(s) with extended capacity drives, but don't forget to SET SY: accordingly.

The second procedure is for those without the new H89 ROMs or the Extended Configuration Option. You will need 3 drives, two of them standard and one extended capacity. Carefully follow these steps.

1. Configure the drives so that SY0: and SY2: are standard drives, and SY1: is the extended capacity drive. If you have an H89, SY2: should be the drive in the H89 cabinet. You should use dip switches instead of programming jumpers in SY0: and SY1:, because later you will have to change them with the power on. You should have the cover off of your H77 or H17 cabinet.

2. Prepare a standard size system disk as explained in step 2 in the first procedure. I will refer to this disk as the INIT disk. Prepare another standard size system disk with BOOT.ABS on it. I will call it the BOOT disk.

3. Boot up on the INIT disk and SET SY0: to the parameters of the extended capacity drive.

4. Exit HDOS with BYE. Boot up on the BOOT disk, and insert the INIT disk into SY2:. Give the command BOOT SY2:. The INIT disk should boot up normally.

5. Change the dip switches on SY0: and SY1: so that the extended capacity disk is SY0:, and the standard disk is SY1:. The computer is on, so use caution.

6. Insert a good blank disk into the extended capacity drive and run INIT. When it asks for a device, enter SY1:. Anser YES to "Double sided" if you set SY0: to two sides in step 3. Answer YES to "Media check".

7. After the initialization is complete, type two control-D's and hit RETURN. Hit RETURN again to reboot.

8. Give the command SYSGEN *.*, and when asked for a device, enter SY1:. When the SYSGEN is completed, reset your computer and try to boot up on the new disk. It should work normally. You can now use it as the source disk for initializing and SYSGENing other extended capacity disks. You can replace your other drives with extended capacity drives, but don't forget to SET SY: accordingly.

PS:

HUGBB Stuff

For those of you, who are still wondering what happens on the HUG Bulletin Board, the HDOS Device Drivers Programmers Guide by Al Dallas, is just a sample. Granted this is an exceptional contribution from the BB but it is just a small part of the whole BB activity.

Subjects such as utilities, device drivers, games, languages, hardware modifications and others are all discussed on the HUGBB. If you want to get more involved with other Heath users' the HUGBB is the place to be right now.

The SOURCE HUG Message Board activity is still alittle slow, but we are showing more participation with each week. We expect things to pick up more this fall after the summer season comes to a close. Don't forget that SOURCE accounts are available through any Heath Store.

For any new HUG members that do not know what the HUGBB or HUGMB are, you may like to get the back issues of REMark, #15, #17 and #18. These issues explain the introduction and basics of getting an account with either MicroNET or the SOURCE.

The following messages were left on the HUGBB:

Another way to bypass the boot carrage/return and get the best of two worlds is to reduce the 30 second Heath default timer to 1 second. This is done via 'DUMP.ABS' at:

Track=0, Sector=2, Addr=2C.

The original value is 074Q or '3C'X. If this is changed to '01'X, no time is allowed to get to 'CHECK-SUM', so use '02'X. This provides a one second delay before auto-boot. If you are quick, you

have time to use the 'IGNORE' and 'CHECK' functions also. Remember, you get a one second delay after each action message.

Bob Pearce 70140,356

►

The M-H8 64K DYNAMIC MEMORY board from TRIONYX ELECTRONICS requires the following modification to operate properly with zero based CP/M in the Heathkit H8 computer:

1) Cut trace to U42-PIN 9, solder-side of board.
2) Raise PIN 11 on U42, component-side of board.
3) Connect a jumper wire from U42-PIN9 to U43-PIN 2.
4) Connect a jumper wire from U42-RAISED PIN 11 to U50-PIN 8.

This change is a design improvement which should be incorporated on all TRIONYX M-H8 MEMORY boards.

Myron J. Seibold 70340,270
Director of Engineering
TRIONYX ELECTRONICS, INC.

►

SPECIAL TO SYSOP: If you want to use this in REMark, be my guest. TO ALL: In my filespace is PSW.DOC[70110,626]. PROTECTion levels are down. As a novice assembly programmer, I use DBUG quite a lot. I always had to stop and figure out what the 'F' REGISTER was telling me. I have written a table which shows the 32 possible values which the 'F' REGISTER can have and shows which FLAGS are set. It speeds up my DE-BUGGING time quite a bit. Hope it can help you. It is in ASCII, but download time is only a couple of minutes. It can be printed with any of the file print programs available. A small contribution to those who have contributed so many good programs on this NET.

Bill Richter 70110,626

The following is Bill's PSW.DOC file:

POSSIBLE FLAG REGISTER VALUES FOR DBUG

+--VALUE--+			D7	D6	D5	D4	D3	D2	D1	D0	<--BITS
DEC	HEX	OCT	S	Z	0#	AC	0#	P	1#	C	<--FLAGS
2	02	002	0	0	"	0	"	0	"	0	S = SIGN
3	03	003	0	0	"	0	"	0	"	1	Z = ZERO
6	06	006	0	0	"	0	"	1	"	0	AC = AUX. CARRY
7	07	007	0	0	"	0	"	1	"	1	P = PARITY
18	12	022	0	0	"	1	"	0	"	0	C = CARRY
19	13	023	0	0	"	1	"	0	"	1	0# = ALWAYS 0
22	16	026	0	0	"	1	"	1	"	0	1# = ALWAYS 1
23	17	027	0	0	"	1	"	1	"	1	
66	42	102	0	1	"	0	"	0	"	0	
67	43	103	0	1	"	0	"	0	"	1	
70	46	106	0	1	"	0	"	1	"	0	
71	47	107	0	1	"	0	"	1	"	1	
82	52	122	0	1	"	1	"	0	"	0	
83	53	123	0	1	"	1	"	0	"	1	
86	56	126	0	1	"	1	"	1	"	0	
87	57	127	0	1	"	1	"	1	"	1	
130	82	202	1	0	"	0	"	0	"	0	
131	83	203	1	0	"	0	"	0	"	1	
134	86	206	1	0	"	0	"	1	"	0	
135	87	207	1	0	"	0	"	1	"	1	
146	92	222	1	0	"	1	"	0	"	0	
147	93	223	1	0	"	1	"	0	"	1	
150	96	226	1	0	"	1	"	1	"	0	
151	97	227	1	0	"	1	"	1	"	1	
194	C2	302	1	1	"	0	"	0	"	0	
195	C3	303	1	1	"	0	"	0	"	1	
198	C6	306	1	1	"	0	"	1	"	0	
199	C7	307	1	1	"	0	"	1	"	1	
210	D2	322	1	1	"	1	"	0	"	0	
211	D3	323	1	1	"	1	"	0	"	1	
214	D6	326	1	1	"	1	"	1	"	0	
215	D7	327	1	1	"	1	"	1	"	1	

1 = SET
0 = CLEAR

*SYSOP <TLJ>

Heath Related Products

Scott Witt the author of the popular HDOS text editor PAGED has informed us that he has now converted PAGED for use under CP/M Version 2.2. The CP/M version contains the many editing and formatting features of the HDOS version plus some enhancements.

PAGED, is one of the few editors designed exclusively for use on the H89 computer and the H-19 terminal. It uses the unique features of the terminal, including the special function keys. The program's main attractions are its ease of use and automatic functions, making it much more powerful than standard text editors. Its single-key commands and extensive prompting make it one of the easiest editors to use.

The CP/M, Org 0 version of PAGED is available for $25.00 from Scott by writing him at 79 Old Haverstraw Road; Congers NY 10920.

UltiMeth Corporation (the author of HUG's SY: -- Dean Gibson), in cooperation with Magnolia Microsystems Inc. of Seattle, Wa., announces the availability of disk device driver software for HDOS 2.0 which supports Magnolia's single-board, soft-sectored, double density floppy disk controller for the Heath H89/Zenith Z89 computers. The software, in conjunction with the disk controller board, will support the following disk drives under HDOS 2.0:

1. Four eight-inch floppy disk drives (single/double-sided, single/double-density), each drive holding up to one-million bytes.

2. Four 5 1/4-inch floppy disk drives (single/double-sided, 40/80-tracks per side, single/double-density), each drive holding up to .66 million bytes.

For additional information on the Magnolia Controller and supplied documentation, contact:

Magnolia Microsystems Inc.
2812 Thorndyke Avenue West
Seattle, WA 98199
Phone: 206-285-7266
or: 800-426-2841

For specifics on Dean's new Disk Device Driver for the Magnolia controller, contact:

Dean Gibson
c/o Ultimeth Corporation
24025 Fernlake Drive
Harbor City, CA 90710
Phone: 213-539-4276

(Vectored from page 17)

885-0019	Color Graphics Poster	$ 2.95
885-4	HUG Binder	$ 5.75

CP/M is a registered trademark of Digital Research Corp.

Changing your address? Be sure and let us know since the software catalog and REMark are mailed bulk rate and it is not forwarded or returned.

CUT ALONG THIS LINE

HUG MEMBERSHIP RENEWAL FORM

When was the last time you renewed?

Check your ID card for your expiration date.

IS THE INFORMATION ON THE REVERSE SIDE CORRECT? IF NOT FILL IN BELOW.

Name _____

Address _____

City-State _____

Zip _____

REMEMBER — ENCLOSE CHECK OR MONEY ORDER

CHECK THE APPROPRIATE BOX AND RETURN TO HUG

RENEWAL RATES		NEW MEMBERSHIP FEE IS:
US DOMESTIC	$15 ☐	$18 ☐
CANADA	$17 ☐ US FUNDS	$20 ☐
INTERNAT'L*	$22 ☐ US FUNDS	$28 ☐

* Membership in England, France, Germany, Belgium, Holland, Sweden and Switzerland is acquired through the local distributor at the prevailing rate.

Local HUG News

Aloha Computer Club -- Jim Branchaud has been appointed President for the local computer group. Jim replaces Gerry Cramm who is being transferred to Camp Pendleton. Jim can be contacted by calling (808) 531-8843. Good luck Gerry and Jim on your new assignments!

The Toronto Heath Users' Group (THUG) has been meeting monthly since 1978 at the Heath Electronics Centre located on 1478 Dundas Street East in Mississauga, Ontario. The exchange of information and ideas has been very effective in helping users understand and expand their systems. For additional information contact Bill Smith at the Centre or, as Bill says, drop him a message via SOURCE MAIL at CL1483.

From MUG -- (The Mission Users' Group) meets on the last Sunday of each month at the Mission Heathkit Electronics Center. All area users are welcome to drop in. Meetings start at 2:00 PM and end between 6:00 or 7:00 PM. For additional information, contact Dave Kobets by calling (913) 362-4486. Dave can be contacted via SOURCE MAIL at TCG328.

SMHUG (Southwest Michigan HUG) is now up-and-running. This group meets on the fourthSaturday of each month at 1:00 PM in Room 1034 of Moore Hall on the campus of Western Michigan University. For information, contact Al Jacobs by calling (616) 349-3535 or write to AL at 623 Wildwood Place; Kalamazoo, MI 49008.

The Wichita Heath Users' Group meets on the second Sunday of the odd months (Jan,Mar,etc.) at 2:00 PM. Meetings are held in the East Pike Building at the corner of Webb and Kellogg in Wichita, Kansas. Please call David Horwitz at (316) 681-3456 between 6:00 PM and 9:00 PM for details.

From HUG "RI" -- The Warwick RI Heathkit Electronics Center opens its' doors to HUG "RI" on the second Wednesday of the month. HUG "RI" has a monthly newsletter. Meetings begin officially at 8:00 PM. However, since the store is open until 8:00, most members arrive between 7:00 and 8:00. For further details contact HUG "RI"; c/o Heathkit Electronics Center; 558 Greenwich Avenue; Warwick, RI 02886.

The Jericho Users' Group (Jeri-HUG) meets on the second Thursday of every month at the Heathkit Electronics Center located on 15 Jericho Turnpike; Long Island, NY 11753. For additional information, call Phil Levinson at (513) 334-8181. Phil can be contacted either on MNET or SOURCE. (MNET #70330163 SCR #TCV162)

Heath Users' Group
Hilltop Road
St. Joseph MI 49085

BULK RATE
U.S. Postage
PAID
Heath Users' Group

POSTMASTER: If undeliverable, please do not return.

885-2020