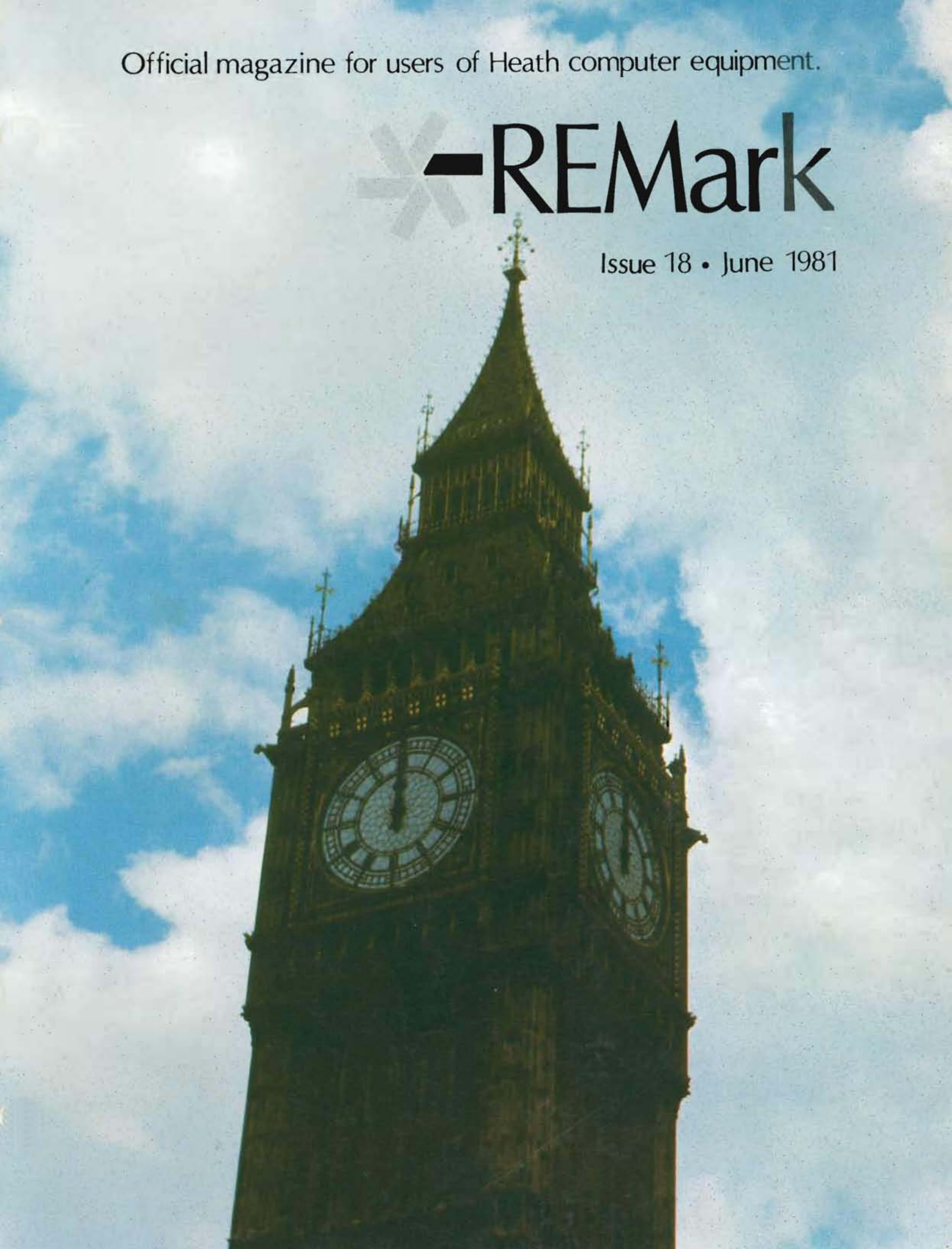
Official magazine for users of Heath computer equipment.

REMark

Issue 18 • June 1981



on the cover . . . .

Big Ben at high noon.

Photo by Gerry Kabelman

on the stack

>CAT

Know Your Group 3

Using the Epson MX-80 Printer with the H89 3

HUG Bug 3

A KISS for Assembly Language Programming 4

Screen Formatting on the H11 7
Ed Judge

Extended Configuration Mod for HDOS 1.6 9

MBASIC POKE ... a No-No? 10

Buggin' HUG 14

Listings Available Shortly for HDOS 2.0 15

New HUG Softwared 16

HUG Product List 17

Number Base Conversions in FORTH 19

The Versatile LPH24.DVD 20

HUGBB Via MicroNET 22

HUGBB Via "SOURCE"? 23

Real-time Functions under HDOS MBASIC 24

Disk Catalogs from BASIC or Tiny Pascal 25
Luis E. Suarez

Comments on the FORTH Language 27

A Caution from Q. A. 29

Using Double Sided Drives on the H17 30

Local HUG News 31

"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

Know Your Group

Many of the same questions arise from old members and new in regard to what the Heath Users' Group is and what the policies are.

HUG can be viewed as an information exchange area for software products, articles, and ideas that are submitted by you the member. As stated on page one of the Handbook, "the Users' Group is intended to provide services to users at low cost and requires the cooperation of its members to accomplish this task."

We receive letters and calls asking why we don't print articles on certain topics or special programs for a specific computer. We only print what is submitted or what we have "popular" requests for. So...if you have an article or program you would like to share...send them in!

Speaking of submittals, one misunderstanding keeps popping up. Only when your program is published as a portion of a HUG product do you receive a one year extension of your membership. If you are not a member, then you receive a one year free membership for your efforts. When your program is received, it is reviewed and you will then be notified by letter. Please keep in mind that your program may not be used immediately. However, this does not mean that it will not be used in the future. Our software team may be on other projects when your program is sent in. Again, we publish what has been submitted based on the majority of the interest level.

One big question is product orders. We accept CASH WITH ORDER ONLY! Heath and HUG orders should be kept separate as this will speed the process considerably. Orders should be on the green HUG Order Forms. Be sure to let us know if you should need some. Removing the label from your REMark to place on the order form is unnecessary also. However, don't forget to write in your ID number.

ID cards for new and renewed memberships have been a problem as many of you well know. Membership cards, until now, have been printed once a month. They will be printed twice a month shortly. Anyone who has not received their card should contact us.

All in all, your group has been through a lot of changes in the last six months or so and we are working hard to "debug" our system. We thank you for your continued support and cooperation.

NS:

Using the Epson MX-80

When connecting an EPSON MX-80 printer to the Heath H-89 via the H88-3 serial board, the following jumper and switch settings on the EPSON RS-232 board should be observed:

JUMPERS		DIP SWITCH	
JSR	ON	PIN 2	OFF
JC	OFF	PIN 5	N/A
JNOR	OFF	PIN 6	OFF
JREV	ON		

In addition, the interface cable should have PIN 11 on the printer connected to PIN 4 on the H-89.

This info was supplied by:

Thomas E. Sullivan
5900 Lockton Lane
Fairway, KS 66205

HUG Bug

Joe Mullins of Worthington, OH found a BUG in the SORTER program on HUG disk part number 885-1044. The error causes the program to self destruct if the input file happens to contain a number of bytes evenly divisible by 256. The patch for this situation is as follows:

ADDRESS	DATA
043.042	303 132 046
045.152	130 040 055 040 061 057 061
045.161	066 057 070 061 212
046.132	302 115 043 066 000 311

Use the HDOS PATCH program to make these changes, hitting CONTROL-D after each of the four series of changes is made, and a second CONTROL-D after the last of the four.

Making the correction in Assembly Language is much easier:

```
INSERT THE LINE          POP     H
BEFORE          DONE     STC
```

Joe's "BUG" was corrected by the original author:

William W. Moss, M.D.
1507 Riverview Lane
Bradenton, FL 33529

A KISS for Assembly Language Programming

THE LONE CASSETTE RANGER

By popular demand HUG will present here a summary of the steps necessary to perform the "BEEP" program outlined in Issues 15, 16, and 17 of REMark for the CASSETTE user. For the most part, it should be explained that the program used in those issues will operate in the same manner when completed using TED-8 and HASL-8. Further, the commands for TED-8 are very similar to the commands used by EDIT under the disk operating system.

For those of you that are completely new to Assembly Language Programming, please review the articles presented in Issue 15, 16 and 17 of REMark. The information contained in these articles will be used here and only altered slighty for you people with cassette equipment only. For those of you with disks, please note the differences in the program presented here as it will operate differently (does not return to HDOS) than the earlier version. Disk users' may choose to assemble this program for a little practice with the EDITor and ASM.

If you are confused already, let's see if I can give a few more details on TED-8 and HASL-8. TED-8 is the Text EDitor used for entering our .ASM file with a CASSETTE DEVICE as described in Issue 16 of REMark. It would follow then, that the HASL-8 or Heath ASsembLer would be used to "put our .ASM file together" into the .ABS file described in Issue 17 of REMark. With few exceptions, TED-8 is identical to the HDOS EDIT. EDIT, in fact, was "built" from TED-8. We will discuss these differences as we go to ensure that our "tape" version of "BEEP" will work. The HASL-8 Assembler is another story, however. When we get to the point of "building" the finished program, we will discuss the HASL-8 format in detail by referring to the software documentation to determine the particular system configuration that you require.

To simplify the "task" of making an .ASM file and the .ABS file on cassette, you may wish to configure the tapes at this time. Follow the instructions found in your software documentation manuals. Further, it is suggested that you place the TED-8 Editor and the HASL-8 Assembler on one tape cassette with TED-8 first and HASL-8 second as this is the order of usage. Use another cassette for storage of the .ASM file and yet another cassette for the .ABS file. Clearly mark each of the cassettes in the following manner to ensure you have placed the correct tape in cassette player when asked:

```
TAPE #1.......TED-8 & HASL-8
TAPE #2.......TED-8 .ASM FILES
TAPE #3.......HASL-8 FINISHED PROGRAM
```

The order of the above tapes is roughly the order of usage. Later we will:

1. LOAD TED-8 (TAPE #1)
2. SAVE BEEP.ASM (TAPE #2)
3. LOAD HASL-8 (TAPE #1..program #2)
4. LOAD BEEP.ASM (TAPE #2)
5. SAVE BEEP.ABS (TAPE #3)

These five important steps should be kept in mind as you read the additional information contained in this article.

As was described earlier, we will now enter TED-8 using the normal "tape-load" procedure you have been acquainted with. TED-8 is loaded the same way that BASIC is loaded. Once you have loaded the TED-8 into your machine and pressed "GO" the response will be similiar to the HDOS EDIT. TED-8 will return the prompt:

```
--
```

In HDOS the prompt indicated that we were ready to <I>nsert text or begin typing our program. In TED-8, however, we must first set the TAB's for the four columns (i.e. LABEL field, MNEMONIC field, ARGUMENT field, and the COMMENT field) before beginning the typing process. This little procedure is accomplished by typing:

```
--TAB,9,17,25,33      (RETURN)
```

Actually, you may pick the tab spacing to be any value. I have used the normal convention used by HDOS so that the program you type will appear as described in Issue 16.

Now we are ready to begin the text <I>nsert process. At the prompt (--) type:

```
--<I>nsert  (RETURN)
```

You are now in the <I>nsert mode. Type the program as described in Issue 16 page 4 fig. A of REMark. The same commands that were used in Issue 16 will also work under TED-8 up to the point of saving the program. If you have problems, refer to Issue 16 until you have completed typing in the program. OH! By the way, since there is no disk system to mess with, one line of text can be eliminated and one must be re-written. The following line should to changed as shown:

```
JMP     HDOSP     should now read:

HLT       (this means HaLT or stop)
```

The line that now reads:

```
HDOSP   EQU     040100A
```

should be eliminated

Continue typing the text exactly as shown with the exception of the two lines mentioned above. If you make a mistake, correct the line as described in Issue 16 with the variety of commands used by the TED-8 Editor keeping in mind that both EDIT and TED-8 are similiar.

SAVING THE TEXT FILE.....

As was accomplished with HDOS and EDIT, we must save our file on an output device. In this case, we will use the commands of TED-8 to place our file on a cassette player as the storage area. FIRST, review your text for any errors using the normal <P>rint commands outlined in Issue 16. Correct any errors using the commands described in the same issue. Once you are sure that the text is complete and accurate, set your cassette for the record mode as if you were making a BASIC "save". Then type:

```
--NEWOUT"BEEP.ASM"  (RETURN)
```

This process places the name of the "record" on your tape player. When you "hit" RETURN the cassette should have started to record. Shortly it will stop under the command of the computer. You now type:

```
--$SAVE   (RETURN)
```

Again the tape will begin to record. This time however, the actual text is being placed on the cassette. After the cassette player has stopped, one further command is required to complete the file. Type:

```
--STOP OUTPUT   (RETURN)
```

This last command places the "stop" output signal on the tape telling the computer that it has reached the End Of File (EOF).

NOTE: It would be a good idea to repeat the procedure outlined previously to ensure a valid copy has been made on the cassette. When making additional copies be sure to leave plenty of room on the tape "between" copies. (The advantages of this procedure will be discussed later.)

CHECKING THE FILE.....

The best way to ensure that we have done things right is to shut down the computer (erase the works!) then try "loading" our new program. To accomplish this task we first load TED-8 and set the TAB's as described. At the prompt, type:

```
--NEWIN"BEEP.ASM"  (RETURN)
```

The computer will then locate the correct record on your cassette tape before stopping. We then type:

```
--FILL   (RETURN)
```

This procedure "pulls" in the remainder of the file that was typed previously. If you now use the technique for printing the text (i.e. -- PRINT) you should now see your work restored in memory.

There is another technique used to check your finished file. As is common practice in BASIC, you may use the "VERIFY" command of TED-8 to check a finished file. I chose to use the loading technique to give practice with TED-8 as you will find a good understanding of "NEWIN" and "FILL" helpful for corrections or modifications to existing .ASM files on your cassette tapes.

USING THE ASSEMBLER (HASL-8)

Once we have completed the .ASM file that was called "BEEP.ASM" and saved it to tape, we are ready to assemble our finished program. HASL-8 the tape version of ASM the HDOS Assembler is loaded into the machine for this purpose. Remembering that HASL-8 is the second program on TAPE #1, install and load the Assembler as you have done in the past with BASIC and now TED-8. The tape version of the Assembler will begin prompting you with several questions that must be answered in order to "get ready" for the actual process.

In an effort to make our task as simple as possible, we will review only those questions and answers that will get the job done easily. For additional information on each of the questions and the various answers refer to "USING THE ASSEMBLER" in the HASL-8 documentation section of the Software Reference Manual for your tape system.

After the "load" for the HASL-8 Assembler is complete, type "GO" as was done for BASIC and TED-8. HASL-8 will respond with:

. (indicating the Assembler is ready.)

Hit return. The assembler will then ask:

LISTING TO PRINTER (Y/N)? <N>

The brackets (<N>) around the "N" indicate that a "RETURN" without typing anything will be the "default" to "N" or no. For the question above and remaining questions type return unless specified otherwise. (The return here would place the listing on your terminal instead of a printer.)

Next, HASL-8 will ask:

BINARY (Y/N)? Y (RETURN)

Answering "YES" to the above question will place the generated "binary" into your machines memory when the actual assembly begins. The next question will apply if you have two cassettes in your system and you wish to save our program on tape directly. However, for this "run" we will answer "NO" and save this area of the Assembler for later discussion.

BINARY TAPE (Y/N)? type: N

By answering "NO" to this question, the Assembler will place the finished program in our machine's memory. After we have run the program, you may wish to repeat the procedure. On the second attempt, answer "YES" if you wish to save the program on TAPE #3.

Next HASL-8 will respond with:

INPUT?

You now place TAPE #2 in the cassette for "loading" of "BEEP.ASM" and type the following information:

INPUT?BEEP.ASM

If you have pushed the "play" button on the cassette, the cassette will begin looking for our program. Once it finds the program a response will appear as:

FOUND BEEP.ASM

The cassette will continue until a new message appears:

REWIND SOURCE TAPE TYPE CR WHEN DONE.

"CR" is the RETURN key. One little trick can fool the machine into thinking you have rewound the cassette. If you made more than one recording of the program as suggested earlier, hit the RETURN key as quickly as possible after the message "REWIND SOURCE TAPE...". The Assembler will read the next copy as if you had actually rewound the tape!

As the cassette continues, yet another message will be printed as:

FOUND BEEP.ASM
POSITION PAPER. TYPE CR:

Since we have not used the printer, just type RETURN to continue the actual assembly. You will begin to see the results of our efforts at this time. And, if all of our steps were correct, you will get the final message like this:

STATEMENTS = xxxxx
FREE BYTES - xxxxx
NO ERRORS DETECTED.

You have now finished your first Assembly Language Program!!!!!

MAKING THE PROGRAM RUN.....

Since there are H-88 owners and H-8 owners out there, two methods are described below to get "BEEP" running.

H-8

To make "BEEP" run use the following steps:

1. reset the H-8
2. push the "REG" key then the "P" key
3. push the "ALTER" key
4. enter the following: 100000
5. push the "ALTER" key
6. push the "GO" key

The H-8 should now respond by running the program "BEEP" as described in Issue 17 of REMark.

H-88

To make "BEEP" run use the following steps:

1. push the "RESET" and "SHIFT" keys
2. at the H: push the "P" key (The H-88 will respond: Program Counter)
3. directly after "Program Counter" type 100000
4. push the "RETURN" key
5. push the "G" key (The H-88 will respond with: Go)
6. push the "RETURN" key

The H-88 should now respond by running the program "BEEP" as described in Issue 17 of REMark.

VECTORED TO PAGE 13

Screen Formatting on the H11

In writing programs for my H-11, I have developed a few techniques for screen formatting and input data checking and processing that might be of some interest to other readers.

These techniques can be used easily in BASIC, FORTRAN, or PASCAL, however I will use BASIC as the example Language since the string handling abilities are the most straight forward. The techniques make writing screen- oriented programs easier, and they impart a professional look to the finished product.

The use of user defined functions (UDF) takes the place of subroutine calls by name in BASIC. For instance, if an input is to be checked, many people use a line to check for each possible input. This is a lot of work, and it can be avoided by using the following techniques.

To begin:

```
        DEF FNI(I$)=POS("YES yes NO no ",I$,1)
                  . . . . .

    XXX PRINT FNC$(<LINE #>,<PROMPT>); \ INPUT I$
        IF FNI(I$)=0 THEN PRINT <ERROR MSG> \ GOTO XXX
        IF FNI(I$)<= 7 THEN <YES ACTION>
        GOTO <NO ACTION>
```

This is a simple and compact method that only allows certain input to be accepted. Note that both "Y" and "YES" are accepted in upper and lower case. The comparison string in the UDF can be any expected input, and if the comparison must be exact, an obvious modification -- the simple addition of a length check -- can further restrict the input. The FNC(#,$) will be explained later, but all it does is center string $ on line #.

On the printing of formatted screens, I use other simple UDF's to make life a lot easier. First I set up an array, usually C$(20), to hold the control codes for my CRT (in my case, both an H-19 and a VT100). I make this array COMMON so that it can be CHAINed to other modules of the program and not have to be redefined. Now I define the functions I use most often.

To print a string S$ at line L, column C:

```
DEF FNB$(L,C,S$)=<X-Y CONTROL STRING>+S$
```

To blank out string S$ at L,C:

```
DEF FNB$(L,C,S$)=FNA$(L,C,SEG$(B$,1,LEN(S$)))
B$="        <ALL BLANK>            "
```

To clear and center a string S$ on line L:

```
DEF FNC$(L,S$)=FNA$(L,INT((80-LEN(S$))/2),<CLEARLINE STRING>+S$)
```

The above makes headings very neat looking. The fact that the line is cleared before the string is printed is very useful in this and other UDF's. Also, putting the cursor HOME (0,0) is useful in preventing spurious <CR>s from shifting screen up and out of register.

To clear line L:

```
DEF FND$(L)=FNA$(1,1,<CLEARLINE STRING>)
```

To center an error msg E$ on line E in Reverse Video:

```
DEF FNE$(E,E$)=<RVID STRING>+FNC$(E,E$)+<NRMVID STRING>(+<BEL>)
```

The bell is optional.

If you use A=SYS(1) for immediate response, the following is useful in the same manner as the first example:

```
DEF FNJ(A)=INT((POS("YyNn...",A,1)/2)+.5)

XXX  PRINT <PROMPT>; \ A=SYS(1)
     IF FNJ(J)=0 THEN PRINT <ERROR MSG> \ GO TO XXX
     ON FNJ(J) GOTO <YES ACTION> , <NO ACTION>
```

This looks very snazzy with a nice screen. The string may be extended with a little care. <CR>'s give a little problem, but is easy to solve. I use the 25th line for <ERROR MSG> on my H-19, and I set top & bottom to define a scrolling area that only changes the active area on my VT100.

A subroutine for checking to see if the input is all numbers:

```
     FOR L1=1 TO LEN(I$) \ L$=SEG$(I$,L1,L1) \ A=ASC(L$)
     IF FNN(I$)=0 THEN XXX
     M=-1 \ RETURN
XXX  NEXT L1 \ M=0 \ RETURN
```

where DEF FNN(N$)=POS("0123456789",N$,1)

Test for M=0, then use "ON M+1 GOTO"

A subroutine for checking to see if the input is all letters:

```
     FOR L1=1 TO LEN(I$) \ L$=SEG$(I$,L1,L1) \ A=ASC(L$)
     IF FNL(L$)=0 THEN XXX
     M=-1 \ RETURN
XXX  NEXT L1 \ M=0 \ RETURN
```

where DEL FNL$(I$)=POS("ABCD...XYZabcd...xyz",I$,1)

The same comment applies. This routine can be easily modified to test for just upper or lower case.

To just check for one letter of either case of letter L$ in the first letter of the string I$.

```
DEF FNK(L$,I$)=POS(L$+CHR$(ASC(L$)+32),SEG$(I$,1),1)
```

Some other UDF's out of any context, but useful:

If your BASIC does not have PRINT USING, some functions can be imitated:

DEF FNF$(Z,Z$)=SEG$("00...00",1,Z-LEN(Z$))+Z$ will pad a string with the leading zeros or (asterisks), and leading spaces or a dollar sign can be easily added.

DEF FNG$(C,S$)=((C-POS(S$,".",1))<CURFWD>+S$) will align the decimal point on colume C. Again, spaces or dollar signs are easily added.

Other subroutines are easily constructed to accomplish specific tasks by using three basic procedures.

One final technique to relate is how to easily print a CRT screen. I use the following method.

First, declare in data statements, the number of fields and the L,C, and S$ for each field. Other data may be added as needed.

```
DATA <NO. OF FIELDS>
DATA 10,12,"FIRST NAME : "
DATA 11,13,"LAST NAME : "
```

Then read them into an array:

```
READ Z  <NO. OF FIELDS>
FOR N=1 TO Z \ READ X(N),Y(N),Z$(N) \ NEXT N
```

Then clear the screen:

```
DEF FNZ$(L)=FNA$(0,L,<CLEAR TO END OF SCR>)
```

clears from the line L to the end of the screen.

Then print the screen:

```
FOR N=1 TO Z \ PRINT FNA$(X(N),Y(N),Z$(N); NEXT N
```

If the string to be constructed and printed is complicated, use another UDF to construct it.

The control of carriage returns is a problem, so avoid using any line below 22 until you get a little experience, as spurious <CR>'s can be very frustrating. They can be controlled with a little thought and experiment.

When a choice is made, you can print the chosen field in RVIDEO for ease of viewing and avoidance of operator error. In my programs, the operator is then asked for confirmation . If a <CR> is then entered, the RVID field is then reprinted in NORMVID and another response will be accepted.

Once you get into the swing of using UDF's you can really spruce up your programs, making them more convenient and less prone to errors. The related methodology opened by these techniques is too enormous to convey in a small article, but developing your own pet methods is most of the fun>

Ed Judge
30 Autumn Drive RFD
Northampton, MA 01060

Extended Configuration Mod for HDOS 1.6

HDOS Version 1.6 will not allow the top 8K of memory to be enabled after the installation of the Extended Configuration Option. Thus, if you intend to run large programs requiring the entire 56K of memory (such as HUG's DND) the following patch is required to properly "size" memory when using the ECO and Version 1.6.

BYTE	FROM	TO		
89	21	21	LXI	H,oFFH
8A	97	FF		
8B	27	00		
8C	2E	25	DCR	H
8D	00	7E	MOV	A,M
8E	24	34	INR	M
8F	7E	BE	CMP	M
90	34	77	MOV	M,A
91	BE	CA	JZ	278CH
92	77	8C		
93	C2	27		
94	8E	00	NOP	
95	27	00	NOP	
96	2B	00	NOP	

The above patch to the first sector (track 1/sector 3 using SUPERDUMP or track 1/sector 2 using DUMP) of HDOS.SYS will correct for the following situation:

When HDOS is booted up, it sizes memory by trying to modify the first byte for each page. Since the test looks for the first byte that it cannot change as the delimiter for memory size, and the machine contains RAM for the entire 64K with the ECO, the test will continue forever! The patch enables HDOS to begin at the top of memory and work down. Therefore, the test is looking for the first usuable address of RAM as "high memory". It would follow then, that if the first memory address is the highest byte, 64K is the "definition" of memory and the test ceases.

Thanks to:

William W. Moss, M.D.
1507 Riverview Lane
Bradenton, FL 33529

MBASIC POKE ... a No-No?

How many of you have been told not to use the POKE function of Microsoft BASIC? I have been told that and I have also told others that . . because it is unsafe and you may POKE some address you are not supposed to. Well, this is indeed true, but maybe we don't really understand how POKE works. Why not take a look at POKE and find out what address we can POKE and what nifty neat things we can do with the POKE command? That is what Bob and I set out to do these last couple of weeks; to understand the POKE command of MBASIC!

To understand where we CAN POKE, we better first know where we CANNOT POKE! So, how can we do this? Well, we must know what is in memory at the time we are running MBASIC, that way we can determine what is not to be changed by POKEing around.

We will use the assembly language program, BEEP, which Bob has spent Issues 15, 16, and 17 of REMark explaining to our beginner assembly programmers (including me). Before we jump right into using POKE, we must first look carefully at HDOS and MBASIC. Fortunately, HDOS 2.0 makes part of our job easy.

First, I better stop and set some parameters so that this will not get too complicated. We will assume a 48K of RAM machine with HDOS 2.0 as the operating system. Our example program will operate on an H8 or H89 with normal HDOS configuration running MBASIC version 4.82.

As most of you know already, a 48K machine actually has approximately 56K of memory. The first 8K (approximately) is the ROM (READ ONLY MEMORY). This is where PAM-8 for the H8 or MTR-88 for the H89 and part of the HDOS memory allocations are stored. (PAM-8 or MTR-88 being the operations that tell your machine what it is to do at powerup time. The HDOS stored in this area is part of the system features e.g. the number of READ/WRITE errors, and the system date, etc.) It is in this area that we NEVER want to POKE for any reason. The actual addresses begin at obviously zero (0) address and uses up to 8832 (decimal) or 042200 split-octal. (Note chart.) That takes care of the 8K of ROM. Now what about the remaining 48K of RAM (RANDOM ACCESS MEMORY)?

Just because we have approximately 48000 bytes of READ/WRITE memory does not mean that we are able to POKE anywhere we choose. The 48K of RAM is used by more than user programs. In fact the remaining HDOS is loaded into HIGH memory at bootup. Well, should I say most of it.

For the purpose of explaining HDOS as it relates to RUNTIME MEMORY ALLOCATION, we will divide HDOS into 5 parts. The first part of HDOS we explained above, that which is stored in LOW memory along with PAM-8 or MTR-88. Now the last four parts are loaded into HIGH memory beginning at the very top of physical memory. Please note the chart. . . The permanently resident HDOS, the device drivers, and the two HDOS overlays make up the four parts of HIGH memory resident HDOS.

Now it would be nice to know what is the address of the top of physical memory. What address is it exactly? How much of HIGH memory does HDOS actually use up? Well, here is where HDOS 2.0 comes in very handy! The STAT command in HDOS 2.0 shows the memory usage of HDOS, the first part of which is as follows:

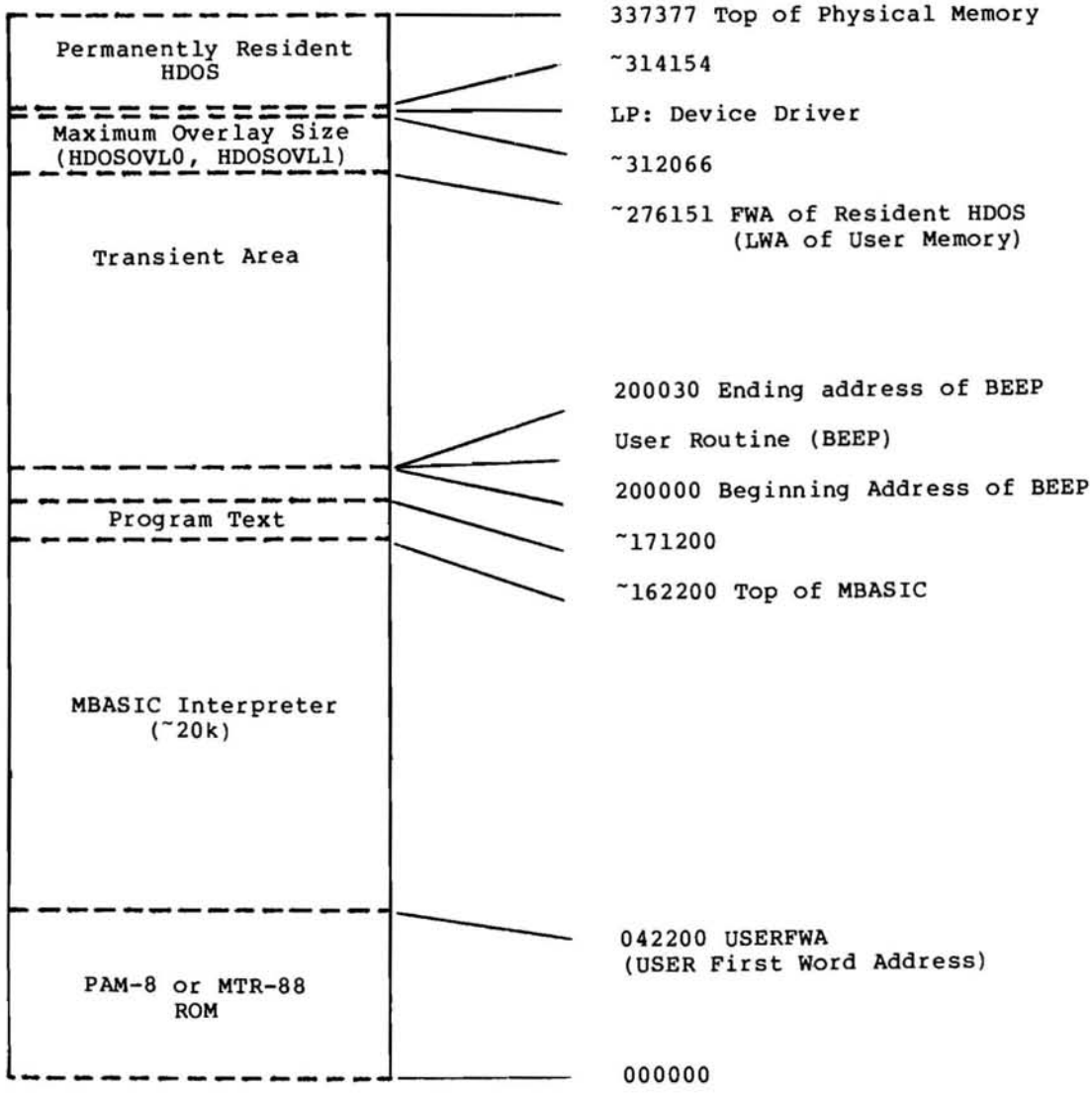
Memory Usage

Physical Memory Limit:	337377
HDOS Lower Bound:	314154
Maximum Overlay Size:	013315

From these three split-octal numbers we are able to determine the exact First Word Address (FWA) of HDOS. This process is much more difficult to determine with HDOS 1.6 but in either case we will leave a safety factor so as not to accidently POKE into an address where an overlay might be called into memory. This probably needs some explanation.

Just briefly, an HDOS overlay is part of the operating system that is not needed to do the minimun operations of the system. The overlays are called into memory from the disk as they are needed. Therefore, even though the overlays are not in permanent resident HDOS, they may indeed be called into use sometime from the system disk.

One more part of HDOS to keep in mind are the device drivers! These are not



HDOS-MBASIC MEMORY MAP
(~ = Approximately)

loaded into memory unless you LOAD them. As you know you cannot print to your LP: device while in MBASIC if you have not LOADed it first. We, therefore must take loaded device drivers into account also.

With all this in mind we are able to look at the chart and see where the four remaining parts of HDOS are located on our memory map. With the help of STAT, we found that our top of physical memory is located at 337377Q (Q is defined as split-octal). The HDOS lower bound excluding device drivers and either overlay is located at 314154Q. With the LP: device driver the lower bound is 312066Q, while including LP: and the two overlays will bring us to our lowest bound of 276151Q.

All of this background just to determine where the FWA of HDOS lies in a 48K system. Yes, but it is very important background because we have narrowed down the area even more of where we CANNOT POKE.

While running MBASIC, we actually only have one more area that we must not POKE into and this is where the MBASIC interpreter and the MBASIC program you are running is stored. This is obviously

another area in which we must be very careful because each program that is run uses different amounts of memory. The MBASIC FRE(0) command will show us how much of RAM is free and not used by the MBASIC interpreter and user program. This will give us a good approximation of where the top of MBASIC could be located in memory. I do not think that MBASIC recognizes the HDOS overlays as part of memory . . so beware of the FRE(0), if you are referring to it.

When the MBASIC interpreter is loaded it locates itself into the lowest address that is available to it. Well, as many of you know this is the users first word address (USERFWA), which is at 042200Q. MBASIC will then take as much memory that is left for user programs, string space, and variables. However, in most cases, as in our example program, MBASIC does not need or use all of the remaining memory but yet it "claims" it, that is the available memory allocation "DEFAULTS" to MBASIC.

But if MBASIC "claims" all remaining memory, how can we use part of the memory for our own use? Well, somehow we must "delimit" MBASIC or set the top of MBASIC at a predefined address. (We must again be careful to allow for user programs, string space, and variables.) We have two ways of setting the top of MBASIC; one, is to use the /M flag while loading MBASIC or, two, by using the CLEAR statement within our MBASIC user program. The syntax of the /M and the CLEAR statement is as follows:

/M:address1(address2)

CLEAR string-space, address1(address2)

Address1 is the address assigned to the top of MBASIC, address2 is the address assigned to the LWA of user memory, string-space is the number of bytes allocated for string storage. The second address, address2, can be omitted which will default all of the remaining memory above the top of MBASIC, address1, for user defined subroutines.

If MBASIC takes approximately 20K of memory for the interpreter, and we can give an approximate size of our MBASIC user program, we can estimate our last boundary where we are not allowed to POKE. Ah ha, but now we have narrowed the memory map down to know where we can POKE. We must always allow for a safety factor on both the lower bound bordering MBASIC and the upper bound bordering HDOS.

For our example program, we can determine the approximate areas that we can POKE into by estimating where we should not POKE. While doing this, I again emphasize the importance of the safety factor on our boundary limits. MBASIC occupies approximately 20480 bytes, our example POKE program will occupy about 1792 bytes (approximately 7 sectors times 256 bytes). This will set our lower limit address at about 171200Q. We have determined through STAT that our upper limit address is 276151Q.

Just to be safe we will begin our POKEing at 200000Q for 30 bytes, which is approximately the size of the BEEP program. However, as mentioned earlier, we could just use "address1" but for the "experience" we will use both. Please note we are no where near either MBASIC or HDOS.

I think we are finally ready to go through our example program. Oops, before we go on we better understand that MBASIC POKEs octal codes not split-octal. Now we are ready! (see FIGURE A.)

As you can see the MBEEP program is self documenting through the REM statements, therefore only a few things need to be covered here. Did you note the DATA, in octal, was identical code to Bob's BEEP program? The only difference is the RETURN to MBASIC rather than to HDOS. The Q=USR1(BEEP) is the statement that actually does the execution of the BEEPing. This is similiar in function to the PRINT statement of MBASIC, in that PRINT causes the action. The "(BEEP)" is arbitrary and may be any string between the parenthesis. One further idea is that you can enter the number of BEEPs, in octal, if you wish, just enter the "&" sign first e.g. &10.

That about completes this lesson on the use of the POKE and PEEK commands of MBASIC. For further information, refer to APPENDIX E - ASSEMBLY LANGUAGE SUBROUTINES and APPENDIX F - RUNTIME MEMORY ALLOCATION, of the MBASIC version 4.82 manual.

PLEASE . . . PLEASE NOTE: Be very careful when using the POKE command. If you will think about it, the POKE command is MEMORY dependent. Our example program was written for a 48K machine. Where is HDOS located in a 32K machine? Sure, it locates itself in HIGH memory again but HIGH memory in a 32K machine is 237377Q and therefore, the FWA of HDOS maximum overlay size would be at 176151Q which is below our 200000Q starting point for our BEEP subroutine. Guess what would happen if you used this program on a 32K machine? On a 48K machine, if the MBASIC program is fairly small, you could have about 10K of RAM for assembly language subroutines. If it is a large program,

such as DUNGEONS AND DRAGONS, even a 56K machine has ZERO area for user subroutines.

So, I guess the bottom line is . . POKE can be used affectively but it must be used carefully and when swapped or traded be sure to specify the amount of memory needed to run that particular program.

(We obviously could not go into any more detail on many items that many of you might know e.g. POKEing into the available RAM area below the USERFWA of 042200Q.)

We have had many calls and letters asking for an article explaining the PEEK and POKE functions of MBASIC. We hope this has been helpful and informative.

<TLJ>

FIGURE A.

```
5 REM:            LOAD THE ASSEMBLY LANGUAGE PROGRAM AT ADDRESS 100000 OCTAL
10 CLEAR100,32768(32798):REM: CLEAR ADDRESS 200000Q TO 200036Q
20 LET A=&100000:         REM: A = FIRST ADDRESS (OCTAL) IN MEMORY WE WILL POKE
30 READ D:                REM: D = THE INSTRUCTIONS AND DATA (OCTAL) FROM THE "BEEP" PROG
40 IF D=999 THEN 100:     REM: 999 INDICATES THE END OF DATA
50 POKE A,D:              REM: PUT DATA INTO MEMORY LOCATION A
60 LET A=A+1:             REM: INCREASE "A" ADDRESS BY ONE FOR NEXT MEMORY LOCATION
70 GOTO 30:               REM: GO GET THE NEXT DATA "BYTE" TO BE PLACED IN MEMORY
80 DATA &6,&5,&315,&136,&2,&26,&377,&36,&377,&35,&302,&11
90 DATA &200,&25,&302,&7,&200,&5,&302,&2,&200,&311,999
100 REM:
110 REM:          THE FOLLOWING MBASIC PROGRAM DEMONSTRATES THE USE OF
120 REM:          THE MACHINE LANGUAGE CALL TO THE PROGRAM "BEEP" PLACED
130 REM:          IN MEMORY ABOVE.  FURTHER TO DEMONSTRATE "POKE" THIS
140 REM:          SECTION WILL ASK FOR THE TOTAL NUMBER OF BEEPS THAT
150 REM:          WILL BE USED AND POKED INTO ADDRESS 100001 (OCTAL) FOR
160 REM:          THE PROGRAM "BEEP" TO ACCOMPLISH.
170 INPUT "HOW MANY BEEPS WOULD YOU LIKE ";B
180 POKE &100001,B:       REM: PUT NUMBER OF BEEPS AT 100001 (OCTAL) IN MEMORY
190 DEFUSR1=&100000:      REM: DEFINE STARTING ADDRESS OF BEEP FOR MBASIC
200 Q=USR1(BEEP):         REM: "Q" PERFORMS THE OPERATION OF RUNNING "BEEP"
210 REM:
220 REM:          THIS NEXT PORTION OF THE PROGRAM DEMOSTRATES THE VALUE
230 REM:          OF THE PEEK COMMAND.  YOU WILL NOTICE THAT WE WILL PEEK
240 REM:          AT LOCATION 100001 (OCTAL) TO DETERMINE THE VALUE THAT
250 REM:          WE HAD POKED THERE WHEN ASKED FOR THE NUMBER OF BEEPS.
260 REM:
270 LET V=PEEK(&100001): REM: V = THE MEMORY VALUE (DECIMAL) OF 100001 (OCTAL)
280 PRINT "THE VALUE OF THE ORIGINAL NUMBER OF BEEPS IS ";V
290 END
```

VECTORED FROM PAGE 6

NOTE: Answering <Y> to BINARY TAPE requires that two cassettes be used, one for play and one for record. If you are using one cassette player, you would perform a MEMORY DUMP at the starting address of 100000 as if you were making a configured BASIC tape. If you intend to do a MEMORY DUMP, perform this step NOW as the program is still in the memory of your machine!

TWO CASSETTES.....

As was described earlier, answering "Y" or yes to "BINARY TAPE" will produce a "stored" program on tape that can be now loaded and run just as if it were BASIC, TED-8, or HASL-8. Repeat the process described for "USING THE ASSEMBLER" in this article. This time however, answer:

BINARY TAPE (Y/N)? type: Y

This change will produce the program "BEEP" on tape rather than in memory. All other steps will remain the same except you would not run this program as described earlier. Remember to use TAPE #3 when storing the program. Also, remember to place the cassette recorder in the "record mode".

Now that you have a finished program on TAPE #3 it can be placed in either the H-8 or the H-88 by "load" as with BASIC, TED-8, HASL-8, and finally BEEP!

I hope the above information will be useful to those of you with tape systems. Further, those of you with disk drives may want to try the changes described to see the effect of the different program on your machines.

BE:

BUGGIN' HUG

Dear HUG,

After many hours of converting programs from HDOS MBASIC to CP/M MBASIC I have found some short cuts that make the task a bit easier.

First, CP/M MBASIC file structure for random files supports random I/O files but defaults at a record length of 128 bytes. HDOS MBASIC supports a 256 byte random I/O. With CP/M you can change the default random buffer using the /S switch when loading MBASIC 5.2 (MBASIC /S:256). When MBASIC loads, you can then open your file for a 256 byte file: OPEN "R",1,"FILE,256. You must include the "256" after the file name!! This method is documented in the MBASIC users' manual but it is well hidden. (/S: is described on page A-3.)

The second item is TT: and LP: output. In HDOS MBASIC two dual-purpose routines can be used to send data to the printer or to the terminal. This is done using files. To write to the printer you can OPEN "O",1,"LP:" and use PRINT#1 to send data to the LP:. You can also OPEN "O",1,"TT:" and use the same routine to send to the screen.

Under CP/M the above procedure DOES NOT WORK!! One can however, trick CP/M and write a dual-purpose routine WITHOUT using the LPRINT statement. Memory location 3 in standard CP/M contains a current device flag. In normal CP/M this value is 0 for TTY: and 1 for LST:. Under Heath CP/M these values are not zero and one but they are one unit apart.

To accomplish the above task you can define a variable at the beginning of your program called PR. PR=PEEK(3). When you want to print to the screen POKE 3,PR. To send a print statement to the printer you can POKE 3,PR+1. POKE 3,PR+1 will completely disable your terminal. You must therefore, POKE 3,PR as you would close a file in HDOS. Please note the following examples:

```
10 LINE INPUT"(TT:) or (LP:)? ";DEV$
20 OPEN "O",1,DEV$
30 PRINT#!1,This is a print out to ";DEV$
40 CLOSE#1:GOTO 10


5  PR=PEEK(3)
10 LINE INPUT"(TT:) or (LP:)? ";DEV$
20 IF DEV$="LP:" THEN POKE 3,PR+1
30 PRINT"This is a print out to ";DEV$
40 POKE 3,PR:GOTO 10
```

Robert Wild
286 Littleton Apt. 306
West Lafayette, IN 47906

Dear HUG,

Here is a patch for XBASIC that allows lower case for my H-11 with the H-19 video terminal.

A study of the reference text showed that bit 14 of location 44 controlled the conversion of the lower to upper case characters. On a hunch I loaded XBASIC, hit "break" and opened location 44. Thus 44/120000. I entered 160000 to enable bit 14 and hit "P" for proceed and was rewarded by the appearance of lower case functions and graphics in XBASIC.

A dump of the BASIC Language seemed to suggest that block 0 was probably an absolute section so I loaded XBASIC into the patch file and found that 44 did contain 120000. Upon entering 44/120000 160000 I have been able to use the lower case functions since.

Frederick W. Kent
333 Liberty
Conneaut, OH 44030

Dear HUG,

As an Amateur Radio Operator, one of the things I was looking forward to for my new H-89 was using it to replace my old Model 19 TTY. I was very disappointed upon receiving the HUG Catalog to find that the RTTY Program (885-1023) was for the H-8 ONLY! After some study, I could find no reason why the program should not run the H-89 other than I/O port addressing and the loss of the H-8 front panel status indications. I concluded that the port address should be able to be changed in software, and I could live without the status indications.

As a novice working with the computer software, I had no trouble changing the port address. Using only one disk drive, "PIP" was used to review the .ASM and .ACM files to determine required changes. The three files requiring port address cahnges were transferred to another disk containing the "EDIT" Program using "ONECOPY". Using "EDIT", the changes are as follows:

1. FILE: INPUT 8. ASM

   LINE: UPORT EQU 070Q

   CHANGE: 070Q TO 330Q

2. FILE: OUTBAUD, ACM

   LINE: TTY EQU 070Q

   CHANGE: 070Q TO 330Q

3. FILE: CWROU. ACM

(a) LINE:: CWSPRT EQU 075Q

   CHANGE: 075Q TO 335Q

(b) LINE: CWPRT EQU 074Q

   CHANGE: 074Q TO 334Q

Again, using "ONECOPY", the three revised files plus all remaining original .ACM files were transferred to another disk following the sequence on the original disk. Because of disk space, it was necessary to call on a friend who has two drives in order to reassemble the program producing RTTY.ABS now usable on the H-89 port 330Q. Finally, RTTY.ABS was copied to a bootable disk for everyday use.

I am very pleased with this HUG Program. It works fine! Lack of status indications is not much of a problem since other station equipment provides "on the air" and "output activity" indications. The only thing missing is the actual number readouts. My Model 19 is for sale.

In keeping with the stated basic purpose of HUG, it is suggested that this information be made available to others via REMark, and additionally, HUG should make available a disk (885-1023/H-89) with the same changes. It seems to me that this is not a program change as such. It is only adapting an existing HUG product for use on the H-89.

EDITOR'S NOTE: Mr. Anderson called to inform us that the UPORT change is the only change necessary to run on the H89. This was confirmed by Howard Nurse.

Robert R. Anderson, K2BJG
69 Page Drive
Oakland, NJ 07436

Dear Bob,

Thanx for the neat information on the RTTY package. Gerry Kabelman now has your data and will add it to the future production of the HUG part 885-1023. We really appreciate your help with this package as I am sure other Amateur Radio Operators will that use the H-89 as their main computer.

Uncle HUG

FOR DOC CAMPBELL FROM MNET.....

Just a short note on a personal basis to let you know that, I for one, appreciate you and what you offer to the users' of the BB and the readers of REMark. A great number of questions that have occurred to me have been more than answered by reading your comments on the "BOARD" and by reading your articles in the magazine. It is people as yourself that make a hobby like "hacking" a real pleasure. Sometimes one can feel alone with his problems...just getting an H-89 up and running (kit) is no small task. Heath does provide support but it is a long distance call or a ten day wait for the mail. Sometimes the explanation is merely a repeat of what is already printed in the manual...we do not always think alike. Thank you again, sir, for making me feel like a friend even tho' we've never met nor even corresponded before.

BEST PERSONAL REGARDS ... Dave Harrah

Listings Available Shortly for HDOS 2.0

HDOS 2.0 source listings will be available shortly. The only way to distinguish the new listings from Version 1.6 is via the part number on the cover. Version 1.6 listings have the part number of 595-2466, while Version 2.0 listings will carry the part number 595-2466-01.

Customers wishing to upgrade their current listings to Version 2.0 are instructed to return their copies of the Version 1.6 listings along with a check for $50.00 to the following address any time after June 1, 1981:

Heath Company
Service Receiving Department
Benton Harbor, MI 49022

Be sure to include with your order the new part number so that you will receive the correct information.

New HUG Software

885-1095 HUG SY: Device Driver $30.00

The HUG SY: Device Driver is a replacement for the standard Heath SY: device driver for H17/H77/H87 mini-floppies written for HUG by UltiMeth Corporation. It offers the following features:

1) A 35% reduction in time to load large programs (e.g. MBASIC) and in copying large files using PIP.

2) Individually SETtable step times for SY0:, SY1:, and SY2:.

3) The ability to SET the time interval after a disk I/O operation that the read/write head stays loaded. This allows the head to remain loaded between rapid I/O operations (as when loading MBASIC programs or when editing files), reducing head and media wear.

6) The ability to SET the time interval the motor stays on after disk I/O.

7) The ability to perform a media check during INIT, eliminating the need for TEST17 except for drive check-out.

8) Recording of the step time in the boot track during INIT, resulting in up to a 50% reduction in boot time for fast drives.

9) Improved error recovery which temporarily increases the seek step time during the error retry of a disk operation. This allows the step time to be SET to give the fastest usable seek rate, and still handle an occasional error.

10) Circumvention of a bug in the H17 ROM disk read routines.

11) An attempt to initialize a write protected diskette is detected when the disk is inserted in the drive during INIT, not when initialization is complete.

All of the above are supported under HDOS 2.0 without any hardware or software changes except for the replacement of SY.DVD on your system disks. This device driver also supports increased disk capacity as follows:

1) Support for dual-sided drives (the H8 requires the extended configuration option to run dual-sided drives), including the ability to detect, read, boot, and write a single sided diskette on a double sided drive.

2) Support for 80-track per side drives, including the ability to detect, read, and boot (but not write) a 40-track diskette in an 80-track drive.

As above, these features do not require any hardware or software changes except the replacement of SY.DVD, along with the replacement of one or more of your drives. An 80-track double sided drive can store 4 times as much data as the 5-inch drives now equipped in your computer. Three of them will give you 1.2 megabytes of on-line storage on inexpensive 5-inch diskettes. The pin-outs and screw holes on most 5-inch drives are the same, so replacement is fast and easy.

NOTE: More specific information on drives is supplied with the documentation included with the HUG SY: Device Driver, but if you wish to purchase drives while waiting for your copy, the following information is given. If you buy 80-track drives, they must be the 96 TPI (Tracks Per Inch) type. Some 80 track drives are 100 TPI, and some 96 TPI drives are only rated at 77 tracks. Tandon Magnetics and Micro Periferals, Inc. manufacture 80-track 96 TPI drives, single and double sided.

The HUG SY: Device Driver comes complete with the source listing and instructions for re-assembling the driver if you wish to make changes. Complete documentation is also provided. If you have technical questions concerning this device driver, direct them to:

Dean K. Gibson
UltiMeth Corporation
24025 Fernlake Drive
Harbor City, CA 90710
(213) 539-4276 (9 AM to Noon Pacific Time)

885-1094 HDOS Fig-FORTH 2 disks $40.00

885-1208 CP/M Fig-FORTH 2 disks $40.00

HUG FORTH is an implementation of the Forth Interest Group's FORTH for the 8080. The HDOS and CP/M versions are virtually identical. Both use track-sector disk access that is independent of the operating system, so that HDOS FORTH can read "screens" (FORTH files) written by CP/M FORTH and vice versa. (This does not mean that you can use FORTH to copy regular HDOS and CP/M files between operating systems, though.) HUG FORTH includes an 8080 assembler for

including assembly code in FORTH definitions, and a complete version of the Fig Editor, including MATCH in high level FORTH and all string commands. A SAVE command lets you save any words you add to the protected dictionary as part of FORTH itself on disk. Both versions support output to a printer.

HUG FORTH requires an HDOS or CP/M system with at least 32k RAM and at least 1 5-inch disk drive (2 on CP/M). For more information on FORTH, see the article Comments on the FORTH Language in this issue of REMark.

885-1097 Educational Quiz Disk $20.00

ATTENTION: Educators, students and those interested in learning from their computer.

Your computer can teach addition, subtraction, multiplication, division, ratios, spelling and word usage with the 885-1097 Educational Quiz Disk.

This disk includes programs which teach the above subjects and includes a special reward if the student gets a passing grade.

The word usage, spelling and ratio quizs are written in a format that allows changing or adding to the quiz with little programming knowledge.

The 885-1097 disk requires HDOS, MBASIC and an H89 computer or H8 computer with the H17 disk drive and the H19 terminal.

A PROLOGUE.SYS and MENU.BAS are included with this disk to make the disk accessible to the youngest user.

The quizes included on this disk are intented for students in the grades one to four except the word usage and the ratio quizes. These two quizes are on the level of six or seventh grade.

885-1096 MBASIC Action Games $20.00

Take a scenic drive, destroy your opponent's tank, shoot the enemy planes down, surround your opponent, blast your way out or just doodle a while with this MENU driven HDOS MBASIC games disk.

The 885-1096 disk comes with its own PROLOGUE.SYS and a linking MENU.BAS to allow Turn-Key type operation. You will have to supply HDOS and MBASIC.ABS. You also need the H89 computer or H8 computer with the H17 disk drive and H19 terminal.

The action within several of the games included on this disk is created by a real-time user-defined function. This user-defined function is explained on page 24 of this issue of REMark.

The action games are Tanks, Planes, Surround and Scene Drive.

Don't be fooled by the SCENIC DRIVE, as you will have to remain on one of the most crooked roads in AMERICA. This road is almost as bad as the world famous Lombard Street of San Francisco. Try negotiating this road at 55 miles per hour, many have tried but few have made it past the hair-pin curves.

The AIRPLANE game has enemy planes flying overhead and your mission is to shoot as many as possible with your gun and guided missle.

The TANK game is for two players and the object is to shoot the other's tank and to avoid the large mine field.

The BLAST game is also for two players. Each trying to blast his way out without being blown up, by getting too close to one of the mines when they explode.

The SURROUND game is a really tough one in that the two opponents are trying to get the other into a corner causing their opponent to destroy him/herself.

The DOODLE program is just that, doodling. You can draw pictures on the terminal and then save them to disk. Retrieval is done in HDOS by typing the file to the terminal.

HUG PRODUCTS LIST

Part Number	Description	Selling Price
	CASSETTE SOFTWARE (H8 and H88)	
	MISCELLANEOUS COLLECTIONS	
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
885-1057	Tape V Cassette	$ 9.00
885-1058	Volume V Documentation and Program Listings	$ 12.00

UTILITIES

885-1034	Character Ed Cassette H8 Only	$ 11.00
885-1035	ED/ASM/DEBUG Cassette H8 Only	$ 11.00

PROGRAMMING LANGUAGES

885-1039	WISE on Cassette H8 Only	$ 9.00
885-1040	PILOT on Cassette H8 Only	$ 11.00
885-1045	FOCAL Cassette H8 Only	$ 11.00
885-1085	PILOT Documentation	$ 9.00

AMATEUR RADIO

885-1027	Morse8 Cassette H8 Only	$ 14.00
885-1028	RTTY Cassette H8 Only	$ 11.00

HDOS SOFTWARE (H8 with H17 or H89)

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
885-1060	Disk VII H8/H89	$ 18.00
885-1062	Disk VIII H8/H89 (2 Disks)	$ 25.00
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 or B H BASIC	$ 18.00
885-1068	Disk XII MBASIC Graphic Games	$ 18.00
885-1088	MBASIC Graphic Games	$ 20.00
885-1093	DND Game for HDOS MBASIC and H89 or H8/H17/H19	$ 20.00
885-1096	MBASIC Action Games MBASIC and H89 or H8/H17/H19	$ 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
885-1061	TMI Load H8 Only	$ 18.00
885-1063	Floating Point Disk H8/H89	$ 18.00
885-1065	Fix Point Package H8/H89 Disk	$ 18.00
885-1075	HDOS Support Package H8/H89	$ 60.00
885-1077	TXTCON/BASCON H8/H89 Disk	$ 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
885-1092	RDT Debugging Tool H8/H89 Disk	$ 30.00
885-1095	HUG SY: Device Driver HDOS 2.0	$ 30.00

PROGRAMMING LANGUAGES

885-1038	WISE on Disk H8/H89	$ 18.00
885-1042	PILOT on Disk H8/H89	$ 19.00
885-1059	FOCAL-8 on Disk H8/H89	$ 25.00
885-1078	HDOS Z80 Assembler	$ 25.00
885-1085	PILOT Documentation	$ 9.00
885-1086	Tiny Pascal Disk	$ 20.00
885-1094	HUG Fig-Forth H8/H89 2 Disks	$ 40.00

BUSINESS, FINANCE AND EDUCATION

885-1047	Stocks H8/H89 Disk	$ 18.00
885-1048	Personal Account H8/H89 Disk	$ 18.00
885-1049	Income Tax Records H8/H89 Disk	$ 18.00
885-1051	Payroll H8/H89 Disk	$ 50.00
885-1055	MBASIC Inventory Disk H8/H89	$ 30.00
885-1056	MBASIC Mail List H8/H89 Disk	$ 30.00
885-1070	Disk XIV Home Finance H8/H89	$ 18.00
885-1071	SmBusPkg III 3 Disks H8/H19/H89	$ 75.00
885-1091	Grade and Score Keeping	$ 30.00
885-1097	Educational Quiz Disk MBASIC and H89 or H8/H17/H19	$ 20.00

AMATEUR RADIO

885-1023	RTTY Disk H8 Only	$ 22.00
885-1052	Morse8 Disk H8 Only	$ 18.00

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 (version 1.43 -- ORG 4200H)

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

CP/M SOFTWARE (version 2.2 -- ORG 0)

885-1207	TERM and H8COPY	$ 20.00
885-1208	HUG Fig-Forth H8/H89 2 Disks	$ 40.00

MISCELLANEOUS

885-0017	H8 Poster	$ 2.95
885-0018	H89 Poster	$ 2.95
885-0019	Color Graphics Poster	$ 2.95
885-4	HUG Binder	$ 5.75

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

DID YOU KNOW?

There are approximately 175 programs available to the computer hobbyist via Volumes I to V of the Software Documentation offered by your Heath Users' Group. These programs, although designed for the tape user, can be easily modified and saved to disk. The programs vary in both content and variety and can serve as comparisons for that special project you may want to write.

Number Base Conversions in FORTH

by Glen B. Haydon, M.D.
Box 429, Star Route 2
La Honda, CA 94020

I was amused at the BASIC programs for decimal to binary conversion included in REMark issue #15. It illustrates the complexity of BASIC programs.

In FORTH, if "BINARY" is not already defined, its definition can be easily written:

```
: BINARY 2 BASE C! ;
```

Use of this FORTH word for number conversions is illustrated in the terminal session (which was echoed at the printer for the listing following this article). The only apparent time required is for the output even if a printer is not used. Compare the speed of this with the programs in BASIC.

FORTH is usually used with numbers in decimal notation. Therefore, one need simply enter the number to be converted followed by "BINARY U." and the binary value will be printed. Then enter a binary number followed by "DECIMAL U." and the decimal value will be printed. The FORTH word "U." causes the number to be printed as an unsigned 16-bit number. If the usual word for print, ".", is used, any number over 32767 decimal is printed as a two's compliment negative number. To return to the usual number format without converting a binary number back simply enter "DECIMAL". This is much shorter than either BASIC program.

Because of the problems of converting among the various number systems used in programming, I devised a FORTH program (listed below) which defines two H19/H89 function keys, f1 and f2, to convert any number in any base to its value in each base. Arithmetic can be done among numbers in various bases and the answer presented in all bases. The second function key, f2, is needed for split octal entry, which is actually entered as two octal numbers. You should enter OCTAL first to set the base to octal. Use f1 to enter any other base.

With FORTH, after you load this one-screen program, all number conversion problems are solved. Lines 4 and 5 define the words "BINARY" and "OCTAL", which change the base for inputting and outputting numbers. FORTH uses the variable BASE to store the current input-output base as a number (2 for binary, etc.). On lines 5 and 6 are words to convert octal to split octal (O-SO) and vice versa (SO-O). As mentioned before, split octal values are treated as two separate octal values. The FORTH word "PRINT" beginning at line 8 does all of the work. The columns are labeled and then the output is formatted for each base and printed. The current base is saved in the return stack (>R) and restored after conversion. Finally, there are the two function key definitions. These keys are non-printing, and do not show up on the listing. The comments in parentheses indicate that the function keys have been entered.

Following the screen listing is a session illustrating first simple conversions as mentioned above, then a sample run of the program. Since the function keys do not print, comments show where they have been entered.

```
40 LIST
SCR # 40
  0 (  NUMBER BASE CONVERSIONS IN FORTH by Glen B. Haydon, M.D. )
  1 (  Modified by P. Swayne )
  2
  3 [COMPILE] FORTH DEFINITIONS DECIMAL
  4 : BINARY 2 BASE C! ;
  5 : OCTAL 8 BASE C! ;
  6 : SO-O SWAP 256 * + ;
  7 : O-SO OCTAL 0 256 U/ 7 .R ." ." 3 .R ;
  8 : PRINT CR CR DUP DUP DUP DUP DUP CR ." SPLIT-OCTAL      HEX"
  9         ."      DECIMAL     OCTAL              BINARY"
 10         CR O-SO HEX 0 9 D.R 0 12 DECIMAL D.R
 11         0 10 OCTAL D.R 0 20 BINARY D.R HEX CR ;
 12 :  ( F1 ) BASE C@ >R PRINT CR R> BASE C! DROP ;
 13 :  ( F2 ) BASE C@ >R SO-O PRINT CR R> BASE C! DROP ;
 14 ;S
 15
```

```
Sample Run of the Number Base Conversion Program

65 BINARY U. 1000001 OK

1100 DECIMAL U. 12 OK

7  ( fl struck )

SPLIT-OCTAL      HEX     DECIMAL     OCTAL             BINARY
      0.  7        7           7         7                111

OK
HEX D  ( fl )

SPLIT-OCTAL      HEX     DECIMAL     OCTAL             BINARY
      0. 15        D          13        15               1101

OK
DECIMAL 7 HEX D +  ( fl ) ( add 7 decimal to D hex and print result )

SPLIT-OCTAL      HEX     DECIMAL     OCTAL             BINARY
      0. 24       14          20        24              10100

OK
OCTAL 42 200  ( f2 )

SPLIT-OCTAL      HEX     DECIMAL     OCTAL           BINARY
     42.200     2280        8832     21200     10001010000000

OK
377 377  ( f2 )

SPLIT-OCTAL      HEX     DECIMAL     OCTAL           BINARY
    377.377     FFFF       65535    177777    1111111111111111

OK

                                                      EOF
```

The Versatile LPH24.DVD

Mr. Ted Andrews
672 Washington Ave.
Havertown, PA 19083

To those of you that have received, or are about to receive HDOS 2.0, may I suggest that you examine the entire package of disks carefully.

Like many HEATH users, I have been looking for a set of good Device Drivers for my various line printers. Unlike many printer device drivers supplied by other venders, the HEATH supplied LPH24.DVD, accurately Tabs, Spaces, Form Feeds, and otherwise prints a true copy of any ASCII file. The only short coming is this driver only responds to a +12vdc printer buffer full indication on the RTS lead from the printer.

Now, true to the HEATH standard, there is a bonus for us on the HDOS 2.0 Driver Source disk. A listing of the nicest LP.DVD anyone could want, "LPH24.ASM".

I use three different types of terminals for producing hard copy. The first is a H24 type, which sends a high (+12v) on its RTS lead to the computer to indicate a full buffer. The second is a DIABLO type printer that sends a low (-12v) on its

RTS lead to the computer to indicate a full buffer. The third is a 43 BSR Teletype that sends a Control-S to the computer to suspend printing when its buffer is full, and then sends a Control-Q to restart printing as the buffer empties.

My solution on how to support all three types of printers is as follows:

1) This is a partial listing of LPH24.ASM from HDOS 2.0. In "WAIT0" the JNZ WAIT0 sets up a do-nothing loop any time the printer places a high (+12v) on its RTS lead.

```
     TITLE     'HDOS LP: DEVICE DRIVER, H-24 (TI 810)'
***  LPDVD - LINE PRINTER DEVICE DRIVER
*
*    G. A. CHANDLER           24-AUG-78
*


WAIT      EQU       *
          PUSH      H

WAIT0     LDA       S.CAADR+1
          ANA       A
          JNZ       WAIT3          IF CTL-Z,-A,-B,-C  HIT

          LDA       TLP.POR
          MOV       H,A
          MVI       L,UR.MSR
          CALL      IN
          ANI       UC.CTS
          JNZ       WAIT0          BUFFER FULL, RTS= +5 TO +15 VDC (H14 & 24)

WAIT3     POP       H
          RET
          EJECT
```

2) This is a modified (*) listing with the JZ WAIT0 providing a do-nothing loop whenever the printer places a low (-12v) on its RTS lead.

```
WAIT0     LDA       S.CAADR+1
          ANA       A
          JNZ       WAIT3          IF CTL-Z,-A,-B,-C  HIT

          LDA       TLP.POR
          MOV       H,A
          MVI       L,UR.MSR
          CALL      IN
          ANI       UC.CTS
          JZ        WAIT0          * BUFFER FULL= -5 TO -15 VDC (DIABLO 1640)

WAIT3     POP       H
          RET
          EJECT
```

3) This is the modification (*) that performs a do-nothing loop on WAIT5 from receipt of a CTRL-S to receipt of a CTRL-Q.

```
WAIT0     LDA       S.CAADR+1
          ANA       A
          JNZ       WAIT3          IF CTL-Z,-A,-B,-C  HIT

          LDA       TLP.POR
          MOV       H,A
          MVI       L,UR.RBR       * RCVR BUFFER REGISTER ADDR
          CALL      IN               READ IT
          ANI       7FH            * STRIP PARITY
          CPI       CTLS           * IS IT A CTRL-S
```

```
          JE        WAIT5          *   YES, GOTO WAIT5

WAIT3     POP       H
          RET

WAIT5     LDA       S.CAADR+1      *
          ANA       A              *
          JNZ       WAIT3          * IF CTL-Z,-A,-B,-C  HIT
          LDA       TLP.POR        *
          MOV       H,A            *
          MVI       L,UR.RBR       * RCVR BUFFER REGISTER ADDR
          CALL      IN             * READ IT
          ANI       7FH            * STRIP PARITY
          CPI       CTLQ           * IS IT A CTRL-Q
          JE        WAIT3          *    YES, GOTO WAIT3
          JMP       WAIT5          *    NO, WAIT FOR IT
```

This device driver provides all of the normal LPH24.DVD commands. SET 'HELP', 'BAUD', 'FORM or NOFORM', 'PAGE', 'LENGTH', 'PORT' and 'WIDTH' are all provided as well as the H14's 'LPI'.

The only unusual requirement for using this device driver is if you are not driving an H24 type printer a mysterious character will be printed on line 1, column 0. This can be corrected by entering a 13 (ASCII Carrage Return) for the length, EG: SET LP: LENGTH 13.

HUGBB Via MicroNET

Well, another month has gone by and the HUGBB is still improving. Russel Renshaw of CompuServe continues to make enhancements to the existing Bulletin Board. We may actually be using that projected "new" Bulletin Board I have been advertising for the past few months. I do not know for sure if there will ever really be another Bulletin Board but if there isn't, that is "OK" by me. Our existing Bulletin Board is the envy of the other special interest groups on MicroNET. In fact, the Super Wizard uses our BB as an example to new interested groups.

For the past few months, I have been adding 70+ HUG members to the HUGBB member list. This is really something! The original Bulletin Board was never written to operate with this many users. The message base on the existing BB has a limit of 200 possible messages. We are reaching a level where this limit is reached during the middle of the week, the slow part of the week. If I do not monitor the BB periodically during the weekend, it would easily exceed this message base.

This minor problem does not concern me as this is just a programming modification to the existing software. What lifts my curiousity is why the continued interest in the HUGBB and similiar services from computer systems like MicroNET and the SOURCE?

As you might have read, we have just begun a new HUG Bulletin Board on the SOURCE. NO, we are not moving the HUGBB to SOURCE, nor are we giving up what we have taken so long to build on MicroNET. This service on the SOURCE is just a "Message Board" for the members of SOURCE who do not have the opportunity to get on to the HUGBB via MicroNET.

What does all this mean to you the user, whether or not you have already experienced this "new world" of Multi-user systems?

As we view our present industrial and economical "world", what are the things that stand out as being "long-lasting"? What industries are continuing to grow and demand expertise in engineering, production and marketing? Where is the best return on an investment? Transportation? No! Look at the auto or railroad industries. Space exploration? Well, maybe engineering but production and marketing expertise? How about communication? Ah ha . . now we're cookin'! What about computers? Why sure! This is obvious . . why else would you be reading this article! So, what am I getting at?

First, let's attack communication. An advertisement I am sure we have all heard via radio waves or seen via television is that "business calls" are much more

efficient than those long expensive business trips. What is this saying to us? Maybe that we are demanding better communication. What does this imply? Just the telephone? Not likely! Maybe this implies new and better communication devices! Is satellite television a communication device? Ah ha, what does that imply?

Second, what of this fascinating world we as HUG members are involved with everyday with these "crazy contraptions" called computers? Computers . . wow, that means everything from microprocessors on a single intergrated circuit to the most complex large system which occupies an entire building. Is one independent of the other? Impossible! Are these two different "worlds" of computers to be used to compliment each other? Yes, and they must if computer technology is to continue to develop and become more attractive.

Communication and computers . . . is there something with these two industries? Are they, working together, going to have a major impact on our future? If so, how soon? These are the questions that come to mind when I wonder why are more and more users looking to computer service systems such as MicroNET and the SOURCE.

I guess the answer is right there . . . communication and computers. What we see today on MicroNET or the SOURCE are large computer systems talking to microprocessors via telephone lines. Is this a preview of what will become common practice in the near future? Bob and I foresee that within two years much of our transfer of information, be it programs, utilities or games will be done via this type of media. Does this really seem possible? Well, at 1000+ new HUG members being added per year, just on MicroNET (and this is accelerating) . . what type of demand is this going to put on HUG, MicroNET, the SOURCE, and the telephone company? This does not include any other special interest groups which are strickly hobbyists. What of businesses?

As these demands increase, what will happen? Well, when calculators came out, as a student I could not afford to buy one, even with just the arthimetic operations. Now you can purchase programmable calculators for less than that. Do we project that the same will happen with computers via telephone lines. Again I mention satellite television . . is this another media means to improve and lower the expense of communication between computer systems?

I felt that I had about completed this article when just today, I was made aware of an interesting note on the HUG Bulletin Board. Paul Mayer (70040,645) left me a message telling me of a mini-CBBS system which he runs on his H8/H47/H17. This mini-CBBS is run solely on his H8 and is independent of any outside computer system. The mini-CBBS is a service that local Chicago users are able to access as a group. Is this another angle for information exchange of the VERY near future?

Well, have I answered my curiousity? Yes and no! Yes, because this might explain why more and more interest is being shown with MicroNET, the SOURCE, and other Bulletin Board services . . . No, because I have simply created new questions that cannot be answered today!

With all this in mind, I realize that for us to stay closed minded to the SOURCE and stay strickly with MicroNET is simply not fair to what is taking place "right under our noses"! The SOURCE has plenty to offer, as well as MicroNET. Many users cannot afford to purchase accounts to both . . We do not expect that. However, as HUG representatives we cannot afford not to "open that door" to the SOURCE and what we can learn from it.

*SYSOP <TLJ>

HUGBB Via "SOURCE"?

What's that you say . . a HUG Bulletin Board on the SOURCE? Yep!! . . . Well, sorta. The SOURCE has provided a spot on their POST Board for Heath users or anyone for that matter to exchange info.

This "Bulletin Board" is not intended to replace, take away, hinder or do any such thing to the HUG Bulletin Board on MicroNET. This is intended to provide an access to HUG via the SOURCE.

This "Bulletin Board" is not really a bulletin board per se but actually simulates a message board. For lack of a better term I will refer to the "Bulletin Board" on the SOURCE as the "Message Board".

To access the "Message Board" (MB), use the POST function of the SOURCE. You may READ or SEND messages through POST. At the SOURCE prompt ">" you only need to enter the KEYWORDS . . POST READ HUG. This will take you directly to the HUGMB where you can READ any or all of the messages left to date.

Now this brings up a few interesting questions.

VECTORED TO PAGE 32

Real-time Functions under HDOS MBASIC

A real-time function may be used in HDOS MBASIC by using the user-defined function. This brief article is not to teach how to use the user-defined function rather to show an application for the user-defined function. See TLJ's article in this REMark for further information on user-defined functions.

The actual program is only a few lines long, however the use of these lines is not as difficult as it may seem at first. The important information is contained in lines 2000 to 2080. In these few lines the dimensioning and setup of the actual user defined function is done.

The comments within the listing explain the indivdual functions of each line so further information is felt best explained by actually using the routine.

Examples of this routine may be found in several HUG programs, including the following programs:

Program Name	HUG P/N
SINK.BAS	885-1068
PLANE.BAS	885-1068
LUNAR.BAS	885-1088
SCENCECAR.BAS	885-1097
TANKS.BAS	885-1097
SURROUND.BAS	885-1097
AIRPLANES.BAS	885-1097

```
10 '    .       .       INPUT.BAS       Real-Time For HDOS MBASIC
20 CLEAR 5000:' .       .       .       Clear String Space
30 WIDTH 255:'  .       .       .       Set Terminal Width
40 DEFINT A-Z:' .       .       .       Define A-Z as Integer
50 GOSUB 2010:' .       .       .       Set Input Buffer to Zero
60 '    .       .       .       .       .       .       .       .
100 PRINT TAB(25)A$:'   .       .       Game Functions
110 '   .       .       .       .       .       .       .       .
120 GOSUB 1000:'.       .       .       Get a Character if there was one
130 '   .       .       .       .       .       .       .       .
140 IF A$="E" THEN STOP:'       Check Input For What You're Looking For
150 '   .       .       .       .       .       .       .       .
160 GOTO 100:'  .       .       .       .       .       .       .
900 GOSUB 2070:'.       .       .       Zero Input Buffer Before END
910 END:'       .       .       .       .       .       .       .
920 '   .       .       .       .       .       .       .       .
1000 X=USR0(0):'.       .       .       X=ASCII Value of Input Character
1010 IF X>96 AND X<123 THEN X=X-32:'    Check For Lower Case Letters
1020 A$=CHR$(X):'       .       .       Make A$ Equal STRING Of X
1030 RETURN:'   .       .       .       Return Tto Where We Came From
1040 '  .       .       .       .       .       .       .       .
2000 '  .       .       .       .       Real Time Function for MBASIC
2010 DIM U0(3):'.       .       .       Dimension U0(3)
2020 U0(0)=&H36:'       .       .       Setup U0(0)
2030 U0(1)=&H1FF:'      .       .       Setup U0(1)
2040 U0(2)=&H77D8:'     .       .       Setup U0(2)
2050 U0(3)=&HC9:'       .       .       Setup U0(3)
2060 DEF USR0=VARPTR(U0(0)):'   .       Define USR0
2070 IF USR0(0)<>0 THEN 2070:'  .       Make Sure USR0 Equals Zero (0)
2080 RETURN:'   .       .       .       Return To Where We Came From
```

Disk Catalogs from BASIC or Tiny Pascal

Luis E. Suarez
PO Box 66994
Caracas 1061-A
VENEZUELA

The following programs open DIRECT.SYS to read the HDOS Directory. The program was first developed in Tiny Pascal and then converted to BASIC. I used Tiny Pascal because it is possible to code truly structured programs, from then on it is very easy to convert it to BASIC.

The file names are filed in DIRECT.SYS as follows:

```
  H      D      O      S
000110 000104 000117 000123 000000 000000 000000 000000

  S      Y      S
000123 000131 000123 000000 000000 000003 000360 000000

    000006 000022 000002 000145 000024 000150 000025
```

The above example corresponds to the first name filed in DIRECT.SYS. We are interested in the first 11 bytes that are devoted to filed names and in byte 15 which defines the FLAG. If no name is filed or the file has been deleted then the first byte will be octal 377. Everything finishes with byte octal 376. Let's see what could be done with those remaining bytes. The programs follow.

```
00015 REM : DIRECT.BAS by Luis E. Suarez
00025 REM : OPEN DIRECT.SYS AND LIST DIRECTORY
00035 REM : THIS ROUTINE COULD BE INSERTED IN ANY BASIC PROGRAM TO LIST THE
00045 REM : DISK DIRECTORY WITHOUT EXITING TO HDOS TO DO SO.
00055 REM : VALUES IN THIS PROGRAM ARE DECIMAL
00065 REM : VARIABLE B AT LINE 180 COULD BE USED TO DEFINE FLAGS.
00075 REM : DELETE ODD LINES TO SAVE SPACE.
00085 REM : DIMENSIONING VAR A$
00090 DIM A$(11)
00100 INPUT "CAT (L) OR CAT/S (S) ";I$
00110 PRINT CHR$(27)"E"
00120 C1=1: N=0: IF I$="L" THEN C1=0
00130 OPEN "DIRECT.SYS" FOR READ AS FILE #1
00135 REM : IF NO NAME, JUST READ ALL 23 BYTES UPDATE COUNTER AND BACK AGAIN
00140 B=CIN(1): IF B=255 THEN FOR A=2 TO 23: B=CIN(1): NEXT : N=N+23: GOTO 140
00145 REM : IF END OF FILE PRINT BLANK LINE AND FINISH.
00150 IF B=254 THEN PRINT : END
00155 REM : IF VALID, STORE CHARACTER IN B$
00160 B$=B$+CHR$(B)
00165 REM : READ BYTE 2 TO 8, STORE THEN IN B$ AND ADD A PERIOD
00170 FOR A=2 TO 8: B$=B$+CHR$(CIN(1)): NEXT : B$=B$+"."
00175 REM : READ BYTES 9 TO 11. STORE THEM IN B$. THIS IS THE NAME EXTENSION.
00180 FOR A=9 TO 11: B$=B$+CHR$(CIN(1)): NEXT
00185 REM : JUST READ BYTES 12 TO 14
00190 FOR A=12 TO 14: B=CIN(1): NEXT
00195 REM : READ BYTE 15. CHECK IF CORRESPONDS TO S FLAG.
00200 B=CIN(1): IF B>96 THEN C=1: IF C1=1 THEN C=0
00205 REM : JUST READ BYTES 16 TO 23
00210 FOR A=16 TO 23: B=CIN(1): NEXT
00215 REM : PRINT NAME ACCORDING TO FLAGS.
00220 IF C=0 THEN PRINT B$
00225 REM : CLEAR VARIABLE B$
00230 C=0: B$=""
00235 REM : UPDATE COUNTER. IF 506 BYTES WERE READ, READ NEXT 5 AND RESET COUNTER
00240 N=N+23: IF N=506 THEN FOR A=1 TO 6: B=CIN(1): NEXT : N=0
00245 REM BACK AND START AGAIN
00250 GOTO 140
```

```
{DIRECT.PAS     By Luis E. Suarez}
{WRITTEN IN TINY PASCAL}
{OPEN DIRECT.SYS AND LISTS CURRENT FILES WITH FLAGS}
{THIS PROCEDURE COULD BE INSERTED IN ANY PASCAL PROGRAM TO LIST}
{THE DISK DIRECTORY WITHOUT EXITING TO HDOS TO DO SO}


PROC DIRECT;            {DELETE THIS LINE TO RUN IT AS A PROGRAM}
     VAR FLAG, X, CHR, CHR1, CHR2: INTEGER;
     BEGIN
        WRITE(27,69);                   {CLEAR DISPLAY}
        FOR X:=0 TO %14 DO
         MEM[%76146+X]:=0;      {THIS ALLOWS NON ASCII CHAR. IN GET COMMAND}

        CHR1:=0;
        CHR2:=0;
        RESET('SY0:DIRECT.SYS');        {OPEN DIRECT.SYS TO READ}
        REPEAT                  {THIS ROUTINE READS BYTES IN GROUPS OF 23 EACH}
           GET(CHR);                    {READ FIRST BYTE}
           FLAG:=0;                     {FLAG WILL BE 0 IF NAME EXIST}
           IF CHR = %377 THEN           {CHANGE FLAG TO 1 IF THERE IS NO NAME}
                FLAG:=1;
           IF CHR = %376 THEN           {CHANGE FLAG TO 2 IF END}
                BEGIN
                   FLAG:=2;
                   CHR2:=1
                END;
           CHR1:=CHR1+23;               {BEGIN COUNTING GROUP OF 23 BYTES}

           CASE FLAG OF
                0: BEGIN
                        WRITE(27,106,9,CHR);    {WRITE FIRST CHARACTER}
                        FOR X:= 2 TO 8 DO       {READ BYTES 2 TO 8}
                           BEGIN
                              GET(CHR);
                              WRITE(CHR)        {WRITE IT}
                           END;                 {NEXT BYTE PLEASE}
                        IF FLAG=0 THEN WRITE(27,107,9,9,'.');  {WRITE PERIOD}
                        FOR X:=9 TO 11 DO       {READ BYTES 9 TO 11}
                           BEGIN
                              GET(CHR);
                              WRITE(CHR)        {WRITE IT}
                           END;                 {NEXT BYTE PLEASE}
                        FOR X:=12 TO 14 DO      {READ BYTE 12 TO 14}
                           GET(CHR);            {YOU COULD USE THEM}
                        GET(CHR);       {READ BYTE 15 (WHAT THE FLAG IS)}
                        CASE CHR OF             {DEFINE FLAGS}
                                %20: WRITE(9,9,9,'C');  {WRITE FLAGS}
                                %40: WRITE(9,9,9,'W');
                                %100: WRITE(9,9,9,'L');
                                %120: WRITE(9,9,9,'CL');
                                %140: WRITE(9,9,9,'LW');
                                %200: WRITE(9,9,9,'S');
                                %240: WRITE(9,9,9,'SW');
                                %260: WRITE(9,9,9,'SWC');
                                %300: WRITE(9,9,9,'SL');
                                %340: WRITE(9,9,9,'SLW');
                                %360: WRITE(9,9,9,'SLWC')
                           END;                 {END FLAG CASE}
                        FOR X:=16 TO 23 DO      {READ BYTE 16 TO 23}
                           GET(CHR);            {THEY COULD BE USED TOO}
                        WRITE(10)               {DO A LINE FEED}
                        END;                    {END VALID NAME CASE}

                1: FOR X:=2 TO 23 DO            {NO NAME. JUST READ THOSE BYTES}
                        GET(CHR)
                END;                            {END NO NAME CASE}
```

LISTING CONTINUED ON PAGE 32

Comments on the FORTH Language

FORTH is rapidly becoming one of the most popular computer languages for the advanced user. Many see it as surpassing Pascal for some applications, and it is already in wide use. For example, the Atari company is using FORTH to write programs for coin operated games, and many other manufacturers are using it in similar (hardware controlling) applications.

FORTH is probably one of the most unusual languages available for microcomputers. It combines a compiler, an interpreter, and an operating system in one. It is interactive like BASIC, and you can enter commands and have them executed immediately as in the BASIC command mode. It is a compiler that in some cases produces code more compact than assembly language. It is also an operating system and has its own way of creating and accessing disk files. It is an extensible language, which means that when you compile a program you have written, it actually becomes part of the language itself. Writing a program in FORTH consists of defining words, similar to the way fuctions and procedures are defined in Pascal. Like Pascal, FORTH is highly structured. Its compiler makes only one pass through the source (single pass compiler), and there is no GOTO, so everything must be defined before it is used. Words that have been defined can be used in further definitions until you arrive at a single word that does the whole job. You can run a program in FORTH by typing that final word.

FORTH is often referred to as a systems language because it is well suited to operating hardware systems. Most implementations of FORTH (including HUG FORTH) do not support floating point math, so it is not widely used in business or scientific calculations, but I predict that a floating point package will soon be available. Since FORTH is extensible, compiling the floating point package will make it part of the language. In a sense, FORTH is obsolescense proof.

FORTH uses Reverse Polish Notation (RPN) for mathematical operations and for the whole language as well. For example, the FORTH equivalent of the BASIC statement

```
PRINT 1 + 1
```

would be

```
1 1 + .
```

The period (.) is one of the FORTH words for PRINT. (There are several, to meet various needs. For example, .R prints a number right justified in a specified field.) Note that each word in FORTH must be separated from other words by at least one space. You could not say "1 1 +.", because FORTH would see "+." as one word which may not be defined. The use of RPN can be a bit confusing at first, but if you work at it, it will eventually make sense. (Hewlett Packard users will feel right at home.) For example, in an IF statement, the thing to be decided upon comes before the IF. In BASIC (or most other languages) you say

```
IF A > B THEN do if true ELSE do if false
```

In FORTH, you would say

```
(A) (B) > IF do if true ELSE do if false
```

The variables A and B are in parentheses because variables are not normally used. Forth uses a last-in first-out stack for passing parameters which makes the use of variables almost unnecessary. If they are used, their values must first be placed on the stack. This is normally done using the FORTH word @ (pronounced "fetch"), so the above example becomes

```
A @ B @ > IF do if true ELSE do if false
```

NUMBERS

FORTH can perform operations with bytes (8 bits), words (16 bits) and long words (32 bits). The base used to input and output numbers can be readily changed in a program or in the interactive mode. That makes it easy to write programs that convert numbers from one base to another in FORTH (see "Number Base Conversions in FORTH" in this issue).

Mathamatical operations in FORTH are usually done using signed 16-bit integers, allowing a range of -32738 to 32737. The HUG implementation also supports addition and subtraction using signed 32-bit integers (for a range in excess of + or - 2 billion), and mixed multiplication and division (two 16-bit numbers multiplied to get a 32-bit product, or 32 bits divided by 16 bits to get 16 bits).

DISK I/O

As a disk operating system, FORTH is

somewhat primitive. It accesses the disk as "screens" of 1k bytes each. Since a Heath 5-inch drive can hold 100k bytes, each disk can hold 100 screens, which are numbered 0 to 99. Screens are accessed by number only, so there are no named files. (But you can define a word that loads in a program from particular screens, and in that sense files can have names.) Screen numbering continues through the drives, so a disk in SY1: would contain screens 100-199. If you put a program on screen 50 in SY0:, then transferred that disk to SY1:, the program would be on screen 150.

A SAMPLE PROGRAM

To allow you to compare FORTH to another language, I have translated the color graphics program from REMark issue #17 from Tiny Pascal to FORTH. The translation was very easy since both languages are structured. I learned from this that Tiny PASCAL, which compiles to an .ABS file, is about 2 or 3 times faster than FORTH, which compiles to a series of addresses that must be interpreted by an Address Interpreter. The address interpreter jumps to the routines pointed to by each address in the series, and these can again be more address lists or they can be executable machine code. A language which uses addresses pointing to addresses pointing to addresses, etc., is called a threaded language.

PS:

"BOUNCE" Color Graphic Program in FORTH

for the HA-8-3 Color Graphic Board

```
SCR # 30
  0 ( BOUNCE -- COLOR GRAPHIC PROGRAM IN FORTH )
  1 FORTH DEFINITIONS HEX
  2 ( DEFINE CONSTANTS )
  3 4 CONSTANT DKBLU     8 CONSTANT MDRED     0F CONSTANT WHITE
  4 0 CONSTANT PNT       300 CONSTANT PNTL    800 CONSTANT PGT
  5 800 CONSTANT PGTL    1000 CONSTANT PGTE   300 CONSTANT PCT
  6 20 CONSTANT PCTL     320 CONSTANT PCTE    380 CONSTANT SAT
  7 380 CONSTANT SNT     0 CONSTANT SGT       B8 CONSTANT VDPDAT
  8 B9 CONSTANT VDPCTL
  9
 10 ( DEFINE SHR AND SHL )
 11 : SHR       ( SHIFT RIGHT:  number count SHR )
 12         0 DO 2 / LOOP ;
 13 : SHL       ( SHIFT LEFT:  number count SHL )
 14         0 DO 2 * LOOP ;
 15 -->

SCR # 31
  0 ( BOUNCE COLOR GRAPHIC PROGRAM -- PAGE 2 )
  1 ( VIDEO RAM AND REGISTER CONTROL )
  2 : WVD      ( WRITE DATA INTO VDP RAM:  val adr WVD )
  3          DUP VDPCTL P!
  4          8 SHR 3F AND 40 OR VDPCTL P!
  5          VDPDAT P! ;
  6 : WVA      ( WRITE AN ADDRESS TO THE VDP:  adr WVA )
  7          DUP VDPCTL P!
  8          8 SHR 3F AND 40 OR VDPCTL P! ;
  9 : RVD      ( READ DATA FROM THE VDP RAM:  adr RVD data )
 10          DUP VDPCTL P!
 11          8 SHR 3F AND VDPCTL P!
 12          VDPDAT P@ ;
 13 : WVR      ( WRITE TO A VDP REGISTER:  val reg WVR )
 14          SWAP VDPCTL P!
 15          7 AND 80 OR VDPCTL P! ;  -->

SCR # 32
  0 ( BOUNCE COLOR GRAPHIC PROGRAM -- PAGE 3 )
  1 DECIMAL
  2 : DRAWBOX    ( DRAW A BOX, TOP SIDE OPEN )
  3          DKBLU 3 SHL
  4          24 1 DO
  5               DUP I 5 SHL 5 + PNT + WVD
  6               DUP I 5 SHL 27 + PNT + WVD
  7               LOOP
  8          23 5 SHL
  9          27 6 DO
```

```
 10              2DUP I + PNT + WVD
 11              LOOP DROP DROP ;
 12 -->
 13
 14
 15

SCR # 33
  0 ( BOUNCE COLOR GRAPHIC PROGRAM -- PAGE 4 )
  1 DECIMAL
  2 0 VARIABLE X  0 VARIABLE Y  0 VARIABLE VX  0 VARIABLE VY
  3 : THROW   ( THROW A BALL AND BOUNCE IT
  4         50 X !  10 Y !  15 VX !  0 VY !
  5         BEGIN VX @ VY @ OR Y @ 173 < OR WHILE
  6             Y @ 175 < IF VY @ 2+ VY ! ENDIF
  7             X @ VX @ + X !   Y @ VY @ + Y !
  8             X @ 208 > IF 416 X @ - X ! VX @ MINUS 4 / VX ! ENDIF
  9             X @ 48 < IF 96 X @ - X ! VX @ MINUS 4 / VX ! ENDIF
 10             Y @ 176 > IF 352 Y @ - Y ! VY @ MINUS 2 / VY ! ENDIF
 11             SAT WVA
 12             Y @ VDPDAT P!  X @ VDPDAT P!
 13             50 0 DO LOOP  ( DELAY )
 14             REPEAT ;
 15 -->

SCR # 34
  0 ( BOUNCE COLOR GRAPHIC PROGRAM -- PAGE 5 )
  1 HEX
  2 : IVDP    ( INITIALIZE VIDEO DISPLAY PROCESSOR )
  3       WHITE SGT 800 / SAT 80 / PGT 800 / PCT 40 /
  4       PNT 400 / 80 0   ( INITIALIZATION VALUES )
  5       8 0 DO I WVR LOOP
  6       PGT WVA   PGTL 0 DO 0 VDPDAT P! LOOP
  7       PNT WVA   PNTL 0 DO 0 VDPDAT P! LOOP
  8       PCT WVA   PCTL 0 DO I 10 MOD VDPDAT P! LOOP
  9       3C 7E FF FF FF FF 7E 3C   ( BALL VALUES )
 10       SGT 400 + WVA
 11       8 0 DO VDPDAT P! LOOP   ( INSERT THE BALL )
 12       D0 MDRED 80 64 64
 13       SAT WVA   5 0 DO VDPDAT P! LOOP
 14       C0 1 WVR ;   ( ENABLE VIDEO )
 15 -->

SCR # 35
  0 ( BOUNCE COLOR GRAPHIC PROGRAM -- PAGE 6 )
  1
  2 ( MAIN PROGRAM )
  3 : BOUNCE
  4          IVDP  ( INITIALIZE THE COLOR BOARD )
  5
  6          DRAWBOX  ( DRAW THE BOX )
  7
  8          5 0 DO        ( DO IT 5 TIMES )
  9                THROW   ( THROW THE BALL )
 10                1000 0 DO LOOP   ( DELAY )
 11                LOOP ;
 12
 13 DECIMAL  ;S  ( END OF PROGRAM )
 14
 15
```

EOF

A Caution from Q. A.

The Quality Assurance Department of Heath Company passed along additional information regarding head cleaning techniques for both disk drives and cassette recorders. Siemens, the manufacturer of the drives used in both the H-8 and H-89, strongly discourages using tape head cleaner or cleaning diskettes such as Scotch 3M 744D. They suggest that we USE ONLY SOFT COTTON VERY LIGHTLY SOAKED WITH ISOPROPYL ALCOHOL.

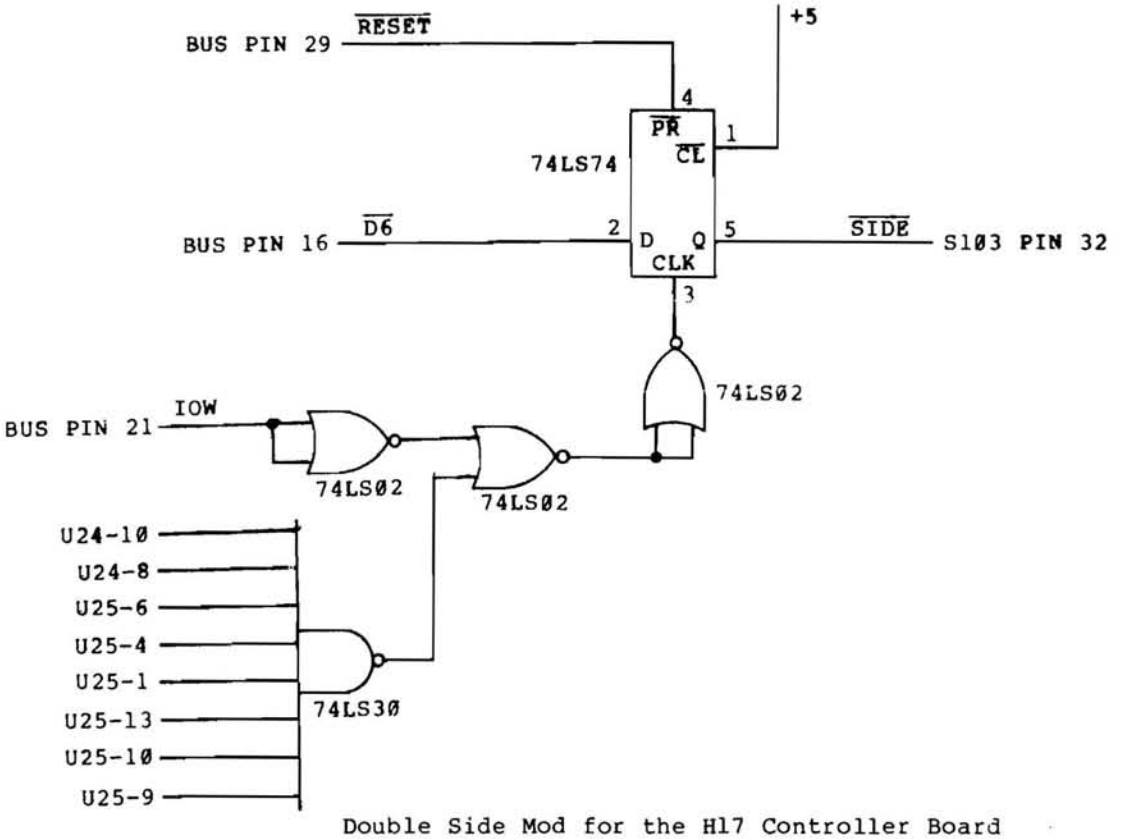
Using Double Sided Drives on the H17

With the new HUG SY: device driver (see New HUG Software), it is now possible to increase your disk capacity under HDOS 2.0 by using double sided drives. However, as the description of the HUG SY: device driver states, the H8 requires the extended configuration option to use double sided drives. If you don't have (or can't get!) an extended configuration option (HA-8-8) for your H8, here is a hardware modification for the H17 that will allow you to run double sided drives without the EC option.

Figure 1 is a schematic of the modification. This circuit decodes port 362 octal when the IOW line is active to produce a pulse when the computer writes to the port. This pulse is used to latch the current value of data line D6, and this latched value becomes the side select signal for the disk drives. This duplicates the side select capability that is already built into an H89 computer. The RESET line is used to ensure that side 0 of the disk is selected at power-up or when the computer is reset. With this arrangement, software that cannot handle two-sided disks can operate the drive as a single sided drive, maintaining compatibility.

The circuit was built by mounting 3 wire wrap sockets on a small piece of perf board. I wired the internal connections first, then attached wires about six inches long to each of the external connections (that go to the H17). I plugged in the IC's and mounted the whole thing upside down with double stick tape near the lower right corner of the board under U19. I drilled a small hole in the board to the right of U19 where it would not cut any traces, and ran the wires coming from the mod board through it, and connected them to the appropriate places. I cut the wire wrap pins as short as possible so the board would fit in my H8 next to another board. I tested the circuit by attaching a logic probe to pin 32 of S103. It should show a logic high when you first turn the computer on. If you send 100 octal to port 362 (with the front pannel), it should go low, and if you send 000, it should go high again. If it passes this test, you should be able to run double sided drives.

PS:



Double Side Mod for the H17 Controller Board

```
    IF CHR1=506 THEN          {WE HAVE READ 506 BYTES FROM 512}
       BEGIN
          FOR X:=1 TO 6 DO    {READ NEXT 6 BYTES TO FINISH THIS BLOCK}

             GET(CHR);
       CHR1:=0                {RESET COUNTER TO 0}
       END                          {BACK AGAIN}
    UNTIL CHR2=1                    {STOP IF BYTE=376}
END;  {REPLACE THE SEMICOLON BY A PERIOD TO RUN IT AS A PROGRAM}
```

Local HUG News

CHUG - The Capital Heath Users' Group has 190 active members. This particular group is one of the largest in the country according to CHUG President, Bill Johnson. This group supports classes for the beginner at Fairfax High School in Fairfax, VA. The classes are lead by members of the club and include introduction to many of the computer languages, HDOS, and the system itself. CHUG meets on the third Monday of each month at Fairfax High School. For additional details on becoming a member of this growing group, contact Denny Smith (703) 978-1260 or write to CHUG; Box 341; Fairfax, VA 22030. Thanks CHUG for your continued support of the Heath Users' Group and the Heath User.

OOPS!.....

Hans Korn of HUG, Germany wrote that their group (approx. 200 strong) is now starting with plans for formal meetings. We had published in Issue 15 of REMark that a new group was just getting together in Frankfurt. Hans let me know in quick order that his club had been active for two years and invited those individuals to join them in their efforts at HUG, Germany. For further details, please contact Hans at the HUG premises in Robert-Bosch-Strasse 32-38, Dreieich/Frankfurt.

MUG....HEC BB on the SOURCE

The Mission Kansas Heath Electronics Center supports the local users' group known as MUG. Dave Kobets, the store Manager, has produced a small bulletin board on the Source for the area users' and any other individual that is interested in following the activities of MUG. Several of Dave's group have already contacted HUG via the new POST section for Heath users' found on the SOURCE. Dave encourages activity on the MUG BB and recent announcements on HUG POST should increase the input. For further information on MUG, contact Dave Kobets at the Mission, Kansas Heath Center

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

		NEW MEMBERSHIP FEE IS:
RENEWAL RATES		
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.

located at 5960 Lamar Avenue or phone (913) 362-4486. Thanks Dave for bringing the MUG BB to our attention.

H-8 NOW RUNNING A BULLETIN BOARD.....

Paul Mayer and Dave Leonard have recently completed many hours of construction to produce a "home built" BB service. They inform us that the entire system is comprised of Heath hardware. The system is a fully loaded H-8 operating with an H-47 and the H-17-3 for a total of five drives for storage. Should you wish to take a look at what can be done when the motivation is there, try your modem on (312) 671-4992. Paul believes this is the first "ALL-HEATH" system ONLINE as a mini-MNET. What a project guys!!!! (NEAT BOARD TOO!!!!)

VECTORED FROM PAGE 23

..... Is SOURCE slower than MicroNET? Do you have to print out all those instructions each time you want to do something e.g. MAIL or POST? What did I mean by KEYWORD in the previous paragraph? Well, these and other questions Bob and I will attempt to answer in the following issues of REMark where we will have a place for the HUGMB via SOURCE.

For this issue I will just say a few words about these questions. To a beginner or someone who does not understand the SOURCE, MicroNET appears to be more user oriented. This is really not the case. The SOURCE uses KEYWORDS, which simply means this allows you to go any place on the SOURCE just by entering the proper KEYWORDS. The proper KEYWORDS can take you "around" all the instructions and time comsuming "jargon" that appears on the screen each time you use some DATA command. This will greatly increase the speed and efficiency of SOURCE and as you "play" with the system and learn what it can do, you will see how very powerful it really is. (That is for future issues.)

One final note about SOURCE: SOURCE application forms are available through all the Heath stores. For reference on comparison of cost of MicroNET versus SOURCE, see Bob's article in issue 17 of REMark, "A RE-VISIT WITH THE SOURCE".

If you are wondering why we are spending so much time with SOURCE and MicroNET, be sure to read the article "HUGBB via MicroNET".

<TLJ>

Heath
Users'
Group
Hilltop Road
St. Joseph MI 49085

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