REMark

Issue 6 • 1978

In this issue of REMark

A sneak preview of the H19 Video Terminal

CP/M

MORSE CODE ON THE ET-3400

RTTY PACKAGE FOR THE H-8

FOR THE H8/H17 SYSTEM

HT 11 INVENTORY PROGRAM

MORE BASIC IDEAS

NEW EDITOR

MODEM TALK

ANOTHER GREAT CONTEST

MORE INTERFACING HINTS

FEATURES OF MICROSOFT tm (MBASIC)

Official magazine for users of Heath computer equipment.

HUG MATERIALS AVAILABLE TO HUG MEMBERS

HERE IS A COMPLETE LIST OF MATERIALS AVAILABLE TO MEMBERS TO DATE.

HUG BINDER	885-4	$ 4.00
HUG TEE S	885-1100	$ 4.50
SHIRTS M	885-1101	$ 4.50
L	885-1102	$ 4.50
SOFTWARE TAPE I	885-1009	$ 7.00
SOFTWARE VOLUME I	885-1008	$ 9.00
VOLUME II		
TAPE II (BASIC)	See Page 34	
TAPE II (ASSEMBLY)		
ADVENTURE (H8) (disk)	885-1010	$10.00
HDOS PROGRAMMING GUIDE	885-1018	$ 5.00
HDOS DEVICE DRIVER	885-1019	$10.00
SOFTWARE (disk) See Page 34	885-1024	$18.00
RTTY COMMUNICATIONS PACKAGE	885-1023	$22.00
HDOS EDITOR	885-1022	$15.00

NOTE: Always place your orders on the green order form and include payment plus shipping and handling.

on the stack

>CAT

COVER PHOTO — H19 New video terminal coming next month described on page 31

"REMark" is a HUG membership magazine published quarterly. A subscription cannot be purchased separately without membership. The following membership rates apply.

	U.S. Domestic	Canada & Mexico	Internat'l
Initial	$14	$16	$24
Renewal	$11	$13	$18

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

Send payment to: Heath User's Group, Hilltop Road, St. Joseph, MI 49085. Back issues that are available cost $2.50 postpaid to U.S. destinations. Request for magazines mailed to foreign countries should specify mailing method and add the appropriate cost.

Although it is a policy to check material placed in REMark for accuracy, HUG offers no warranty, either expressed or implied, and is not responsible for any losses due to the use of any material in this magazine.

Articles submitted by users and published in REMark, which describe hardware modifications, are not supported by Heathkit Electronic Centers or technical consultants.

HUG Manager and Editor ..... Jim Blake
Graphics ..... Ron Hungerford

Copyright © 1979, Heath Users' Group

REMark

CPM FOR THE H8/H17 SYSTEM

Lifeboat Associates in New York is modifying CPM to run on the H8 system. I talked with the President of the firm, Tony Gold and learned what is in store for us. :JB:

HUG: What is CPM?®

TG: CPM is a floppy diskette operating system for 8080 family microprocessors which would include the 8085 and the Z80. It is not only an operating system, which controls the data structures on the disk and the directory housekeeping, but it also includes a family of utility programs to operate with it which includes a text editor, assembler, utilities to debug programs by tracing through them and with break points, examing registers and is a program to recreate systems — to use different amounts of memory, a utility to handle files and direct files from device to device. It's a family. There is one other program. It is a Batch processor so that you can run a sequence of programs regularly just by creating a kind of menu. CPM is a trade name of Digital Research and is an operating system and a family of utilities which are supplied at no extra charge with the system.

HUG: Are you familiar with HDOS?

TG: I haven't had the opportunity and unfortunately the luxury of time to learn all the various operating systems. I have heard good things about HDOS; the fact that the people want CPM really should not be considered a criticism of HDOS. HDOS is very much a captive system of the manufacturer just as for instance ISIS is the captive system of INTEL and PTDOS of Processor Technology. It has been found useful to consider CPM much more of a standard in that it is not owned by a manufacturer. And manufacturers in general don't feel threatened to have this operating system co-exist with their own. In fact, this operating system is used very happily on the Helios with the Sol and on the North Star and on the Micropolis systems, although each of those manufacturers themselves have their own operating system.

HUG: Is CPM just one of those currently popular buzz words or is it really that much better a system?

TG: No, it is not a better system. What I was trying to say, in fact, is that it is possibly cruder than others. I don't know the features of HDOS, it may be that HDOS has superior features. There are some architectural problems with HDOS and I believe some errors were made originally when the thing was architected in the terms of addressing various parts in the system memory. I believe in general that it is not a good system practice to have the operating system hog low memory with all of the interrupt structures. In general it is best to make that available to the user, but the CPM was, I guess, first distributed in 1975 and I am sure that many clever things have been thought up since then and in all likelihood, the person who wrote HDOS was fully aware of what CPM could do and he therefore did things differently because he thought he was doing it better. CPM has a couple advantages over and above it's commonality. One advantage is that it is quite small and another is that it is extremely powerful. The kernel of CPM is only 4 K and that manages all of the disk control functions in terms of maintaining integrity of the data structures and directing data to disk files, opening files and such. That is a very remarkable achievement and considering it was written in a high level language with all the inefficiencies and code explosions associated with that. It is also extremely fast, but it is possible to write bigger operating systems with more bells and whistles. In general, those would operate more slowly. You have on your staff someone who is very experienced with CPM and probably now very experienced with HDOS and this is Barry Watzman. He is one of the people in the company who really understands CPM and wants people to use it. I could count on the fingers of one set of hands and feet the people who really understand it and you've got one of them there and you can certainly go to him for a contrastive analysis.

HUG: About the Editor and Assembler, what bells and whistles does it have?

TG: The Editor is very much reminiscent of the TECO Editor on the DEC machine. It is a character, not a line editor and it allows full global searching and replacing. It offers the ability to find strings deep in files. It has the ability of moving stuff around from one part of the file to another. It has the ability to find out how full your buffer is and what line of the file you are on. It takes a little learning, all editors do. In general, people don't like new editors. Most people learn on an editor and at that point are spoiled. I was brought up with DEC software (TECO), so I kind of slipped into ED with no trouble at all. Other people are more used to the kinds of editors that come with BASIC interpreters and we do have editors available. There is one written by Microsoft that is very reminiscent of their disk Extended BASIC and it has the same attractive features. It is a character editor and it has all the features of that, so you can use it for creating letters and also create source programs or whatever. It provides back up features. Most editors do. When you alter a file, it will back it up.

HUG: The Assembler, does it create cross reference?

TG: The Assembler doesn't create a cross reference table, it creates a listing file, with the errors flagged on the listing file and also separately, any errors will come to console and it creates an output object file in INTEL hex format. It is not a macro-assembler and it does not have linkage and relocation. We have other assemblers that will do all of those things, but the one furnished by Digital Research does none of those.

HUG: Well, obviously, one of the reasons a person might want to buy CPM is because there might be some more software available, right?

TG: Okay, there are two sources of software on the CPM. First of all I must say that the CPM offered on the H8 is not standard CPM, it is modified because of the unique architectural characteristics (of the H8), but for instance, we could come up with a COBOL for the H8 in a week because it is just a matter of relinking it with different constants. Whereas, coming up with a COBOL for HDOS might take two or three months and many thousands of dollars. For instance, the FORTRAN which has been provided for HDOS, we could have had that available on the CPM in about one or two man days of work and a total elapsed time of a week. Microsoft would relink with different constants and run immediately with no debugging or anything. It is field tested and a proven system that has run in thousands of systems around the world. That is the advantage of CPM. And, it is very easy to acquire reconfigured proprietary software. The other source of software is that there is an extensive users' group library of software and Barry Watzman has expressed an interest in making that available to Heath Users and maybe it will go through HUG. Maybe we will put in a separate set of volumes that will be in the CPM library. Those programs are in source form for the most part and those would just be edited for different constants and reassembled for the Heath version of CPM. There is enormous software. It is something like 6 megabytes. It is enormous! It is between 30 and 40 8 inch diskettes.

HUG: Does CPM require two drives?

TG: CPM doesn't require two drives, but no operating system that I've come across could really work well with one drive. All the main problems with single drive systems is that the user doesn't do enough backing up. It takes him 20 minutes to back up his disk at the end of the day and he says, "well, I'll back up at the end of the week". And then one day, unfortunately, the cat bites his diskette half way through the week and at that point he realizes the advantage of a 30 second back up procedure with two drives. Just the redundancy and the ability to have a spare around.

HUG: When will it be available and how much?

TG: The cost is $145 including the utilities and at that time (June) we will also have a COBOL and FORTRAN similar to the one you have. We will have Microsoft Extended BASIC also. And some other languages if they are available. We will be showing CPM at NCC (in May). We are planning to be distributing it by June, but that is conservative. We can generally be on the conservative because what happens is, we normally finish the software well ahead and then sit down and write the manuals. That takes longer, but I think that there should be no trouble in meeting June 1. It is about a three or four man week job.

HUG: Who will support this if the user runs into a question?

TG: We support it.

HUG: You have a staff that can cope with all this?

TG: Yes.

HUG: What kind of documentation can we expect?

TG: There will be 6 manuals from Digital Research plus the manual of special features for the Heath User.

HUG: Sounds good.

TG: The BASIC and all these other things will be available in June also, or is just down the road a little further.

I am going to get better informed on what's run under HDOS to know what I should be providing, but they all can be provided on very short notice. For instance, CBASIC. We had that put together for the TRS80 in about 2 days, just by saying hey, change this number to that and after that it was just relinked. And the guys at Microsoft are equally responsive; so it is a trivial job to come out with these versions for CPM. It is 3 constants or 4 constants and the whole thing is finished, so it is really a matter of deciding what it is that the Heath users want and make sure that it is prepared for him.

HUG: You mentioned BATCH capability. How about indirect files? Where a user can build an indirect file so that when he boots up, it automatically goes to some predetermined task.

TG: Yes, we have that built in the systems we supply. We put in a feature where you have the program called AUTO. When he starts out from cold, it will run this program, then that program itself can, if necessary, chain another program.

We have a thing called a 'mode byte' which has different options, by setting the bits of that byte high or low, he can enable read-after-write or he can enable initialization of this program and a number of different features that we built in based on the setting of this configuration byte and that cold start program is one of the center features.

Armed with this information, we called Barry Watzman over at the 'Software Den'.

HUG: What are the major differences between HDOS and CPM?

BW: There are two major differences in philosophy between CPM and HDOS. When HDOS was written, it was written with the idea that the environment was known. It was a Heathkit system with Heathkit I/O devices and I/O boards.

Also, HDOS was written with the idea that the user was probably not an expert on computers and didn't know much about them, and to a certain extent that the user had to be protected from accidental destruction of the system or his programs and diskettes by himself or by his own unfortunate actions.

CPM, on the other hand, was designed with the philosophy (since Digital Research doesn't even make hardware) that they wanted to make the system as general as they possibly could; totally adaptable to any reasonable hardware environment.

Also, it was designed with the idea that the user or at least the system implementer was somewhat knowledgeable of computers and programming. The consequence is that HDOS does provide the user with somewhat more protection, the documentation quality is better, considerably in some areas, and it is probably

the easier system for the novice to learn. On the other hand, the capabilities that one has with HDOS, (and this is particularly true for the experienced or sophisticated user), are somewhat less than with CPM. HDOS actually does more for the user than CPM does; but sometimes what it does is not what is desired. CPM is a more versatile system than HDOS. It restricts the user a lost less, though it also protects him a lot less. What we found in adapting outside software to HDOS, such as the forthcoming business software, is that some of the things which were done in HDOS to protect the novice user make it difficult for us to adapt software not originally written for HDOS to run uder HDOS. Another difference in the two systems is that because no one microcomputer manufacturer, whether it is Heath or IMSAI or Processor Technology or whomever, has a dominant share of the market, and because software written for any one operating system will in general not operate under any other operating system, one finds that the only operating systems which really attract a very large quantity of available software are those which are not the proprietary property of a single manufacturer. In the case of the 8080 systems that really means CPM. There is really no competitor for it.

HUG: Compare the operating systems directly.

BW: One really does find a lot of similiarities. Maybe more similiarities than differences. Both systems are about the same size. They both need approximately 4 to 6 K — that's a generalization. Both systems have esstentially the same utilities; they both come with a program called PIP which does pretty much the same thing in both systems. In fact, an HDOS user could use CPM PIP without even reading the manual and almost everything would work with the same syntax. They both come with an Assembler, Text Editor and a debug package. Both systems use very very similar, in fact, almost identical disk file allocation techniques; and both of them feature non-physically contiguous dynamic allocation of disk space to files. So, to that extent, they are the similar. CPM's utilities, I think, are generally better than HDOS's. PIP is pretty much the same, but it has the same wild card structure and has pretty much the same command format, but it has about 20 or 25 additional functions that are not present in the HDOS PIP and some of them are very useful. However you can't do a directory or delete files from within CPM's PIP.

The debug package is much better. It allows you to trace execution. And it does not require that you reassemble the program just to run debug. It works with the program at its normal address. It has several other capabilities which HDOS's DBUG lacks, including a built-in desassembler.

The Editor is much better. The Editor is actually somewhat similar to the Editor which I gave you. (which we called BWEDIT for HDOS). It is a little bit slower than BWEDIT but it has many additional features including a Q buffer. And it also has the ability to read another disk file into the one that you are editing. Command syntax is much more concise than the HDOS Text Editor.

The Heathkit assembler is perhaps better than the standard CPM assembler. It has features that the standard CPM assembler does not, but one of the things about CPM is the wealth of software available and so there are least 6 additional assemblers available from other sources. Two of them are free through the Users' Group and in addition there are assemblers by Microsoft™, TDL, Digital Research and TSC which are not free, but which are very nice. The Microsoft Assembler has cross-references and it generates relocatable object modules which can then be linked together with the linkage Editor. The TSC assembler comes with source code. There is also a free assembler in the CPM Users Group, on Volume 16, I believe, which does do cross-references. And is very nice.

HUG: Since the user has the AUTO routine and BATCH, does this allow him to approach turn key operation?

BW: Yes it does. These are really two separate capabilities; the first is the BATCH capability, what CPM calls the SUBMIT FILE. The SUBMIT FILE facilitates the ability to create, using the Text Editor, a file of CPM console commands. Suppose we gave that file the name of SORT. If we then Submit SORT INFILE OUTFILE, it will execute all those commands, but more than that, we could put parameters in the Submit file and the command line values INFILE and OUTFILE will be substituted for the parameters at Run time. You can create a file to run a whole stream of programs and not specify which programs until you Submit the command line. The other facility in CPM to run continuously is the AUTO thing that Tony talked about. But, the distinction in those really is that BATCH runs a file of CPM console commands and AUTO runs an assembly language program. I say assembly language, but it could be Microsoft BASIC. The point is that by combining the two, you really have the ability to do virtually anything you want to including set up pass word protection of the entire system or come up running a given BASIC program

HUG: Speaking of Microsoft, is that going to be transportable between the systems?

BW: No. CPM and HDOS are two completely different operating systems. CPM disks do not have compatible directories, they do not have volume labels, they do not have the same disk allocation structures. The same thing is true with HDOS diskettes. So, anything that is written for CPM will not run under HDOS and anything that is written for HDOS will not run under CPM. Furthermore, the diskettes themselves will be incompatible. It will not be possible to take a CPM diskette and Mount it while you are running HDOS. The reverse is not quite true. CPM provides the assembly language programmer with absolute track sector accessiblity and it will therefore be possible for someone who has CPM to insert an HDOS disk in one of the drives and access it. By the way, one thing I mentioned, CPM will support, in theory, 4 drives; whereas HDOS will only support 2. Our hardware won't support four of course, but we are making some provisions to simulate the third and fourth drive under HDOS — CPM. You will have to change the diskettes, but logically you will be able to run a program that accesses four drives simultaneously. CPM will issue the appropriate messages at the appropriate times. Because CPM provides track sector access, it will be possible for the assembly language programmer to put an HDOS diskette in one of the drives and to access it while running CPM. He will not be able to access it at the HDOS file level, however, he will only be able to access it at the track sector level. With regard to Microsoft

BASIC and Microsoft FORTRAN, the languages supported are the same, but the compilers themselves are different. Therefore, if a user wants to run BASIC and/or FORTRAN under both CPM and HDOS, unfortunately he is going to have to buy the HDOS version and the CPM version. With regard to the program itself, a FORTRAN program will run under either Microsoft FORTRAN for HDOS or Microsoft FORTRAN for CPM with little or no changes, but the user is going to have a media conversion problem which is not inconsiderable. The sophisticated user will be able to figure out a way to do it, I feel quite certain, but most users are going to have to make a decision as to whether they want to run FORTRAN under HDOS or under CPM and for the most part, stick with that. The HDOS versions of BASIC and FORTRAN are much less expensive than the CPM versions, by the way, because Heath's marketing department wants to encourge the maximum possible use of our customers systems, and that means making software readily available; so we have made it (relatively) inexpensive. We have no control over the cost of the CPM versions, since we won't be selling them ourselves: Remember CPM for the H8 will **not** be a Heath Company product, though we are assisting in and encourging its development.

HUG: So, the applications programs are not transportable.

BW: Yes, in general that's correct. Again, the sophisticated user may be able to transport them. I have, as you know, already done some transporting of CPM stuff to HDOS here, but we have more knowledge of the system and better facilities than the average user has, including multiple complete systems. You know, obviously, if you have RS-232 serial ports and you've got two complete computer systems, you can set up one system running HDOS and one system running CPM and ship it down over the RS-232 port which is in essence what we are doing here. In general though, users will not be able to transport programs between the two systems.

HUG: What kind of I/O access does CPM provide?

BW: CPM's I/O capabilities are a lot more versatile than they are under HDOS. One of the things that happens with CPM is that the user has source code for the I/O modules. In a normal CPM system, this would include the disk I/O modules, but Tony, because he is going this as a proprietory product, may choose not to include that. Even so, the user will still have access to the disk at a track-selector level, and the user will still have the source code for his terminal and console device I/O drivers, which means a lot. It means that the user can add features or terminals or additional devices that under HDOS he would not be able to add. More to the point, CPM supports multiple physical devices. For example, let's take a typical top of the line Heath system, the guy has a brand new H19 video terminal (which I don't think is in the catalog yet) and he has a DECwriter and he has an H14 line printer. Now, he comes up and he is running on the CRT terminal and he decides he wants a hard copy lisiting of something he is going to do at the console. All he has to do is say STAT CON:=TTY: and the next prompt comes up on the DECwriter and the DECwriter is, from then until he reassigns it, his console. What's nice too is that this can be done under program control, so your I/O devices are not bound to a physical device at the time you assemble them, they don't bind until run time. This is true of the line printer and other devices as well as the console.

If the user has a line printer, he can say STAT LST: = LPT: and the line printer output goes to the H14. On the other hand, he can say STAT LST:=CRT: and it comes out on his H19 video terminal. One additional feature they provide for is the ability to run really large system type BATCH and they provided BATCH capability, which is different by the way, from the BATCH we talked earlier where you are running from a disk file. Here, BATCH refers to the ability to assign the console to the BATCH device and you can run from an 80 column card reader and a line printer, just leaving the system to run unattended for a long period of time. I don't know if any of our users will ever take advantage of that capability, but it is there. The device independence under CPM is very great, much more so than it is under HDOS, and in large systems with multiple terminals, that can become a significant feature.

One thing I might mention with regard to I/O though; I wrote a skeletal BIOS for Tony, and as I wrote it for him initially, our console I/O is set up without interrupts and that will be one difference between HDOS and CPM.The user will not have the type-ahead buffer. There is nothing in CPM that prevents you from using interrupts, it would be fully possible to enable them and use the front panel clock or an interupt driven console.We certainly could have, but we just felt it would be quicker and easier not to and it is more common in CPM systems to run the console without interrupts. Also, it will make the source code much easier for most of our users to deal with, since interrupt service routines are somewhat complex.

HUG: Earlier you mentioned that HDOS was very protective with the flags and so on, is that true with CPM?

BW: Well, yes and no. CPM is in it's 4th major release which is about a year old now; they corrected a major deficency which sometimes caused people to lose disks when changing them without rebooting. You can change disks readily under CPM now with no danger of losing one. If you have 10 disks and you want to know what is on them, you can just slap the disks in and type DIR, which by the way is a resident command, so it is quick. You can type DIR, get a directory, take the disk out, put another one in. There is no mounting or dismounting. Although there is no mounting or dismounting, if you don't tell CPM to change the diskette, (which is done simply by typing control-C) he assumes that the directory information he has in his tables in memory is correct. HDOS gets around this with mount and dismount. Volume number has its limitations as you only get 255 possible numbers. Some people have the tendency to number all their disks 1 or 255 (me for example). Also, the physical act of having to type Mount and Dismount is a pain. 1.4 CPM fixed that problem in what I consider to be a very satisfactory manner. They checksum the directory on every access. Then, you can change disks freely and do anything you want to as long as you don't try to write; CPM has noted the change and made the entire diskette "read-only" in software. But he really dosen't care. Until you try to write. Then he gives you a read-only error message, and you have to control-C.It is generally impossible now to mess up a diskette. To that extent, it is protected. Under

1.4, I have never heard of a single instance of anyone losing a diskette, but as far as the system itself is concerned, it doesn't really try to protect the user from himself. The user under CPM has complete access to the operating system and all the modules. If he wants to make a patch in assembly language or in HEX (by the way, CPM is HEX oriented as opposed to OCTAL), he has absolute access to CPM's object code. There is no write protect. There is no lock as there is in HDOS so all of the modules of the operating system are directly available to the programmer. If he wants to change the operating system itself, he may regret it, but he can do it. In fact, there is a special utility called SYSGEN which is provided explicitly to facilitate the user's doing just exactly that. It creates in memory at a known location an image of the operating system which can then be modified and written back onto the disk. The user can do anything he wants to do to the operating system for better or for worse, and in some cases, no doubt, it will end up being for worse. One other thing I would like to mention. CPM does not use overlays. The entire system basically is resident. It's quite small as Tony mentioned. It's not position independent code, like HDOS.

HDOS loads into low memory and relocates itself every time you boot. CPM has to go through a SYSGEN process. This is very explicit and I might add it is somewhat demanding. Some of our users may have difficulty getting through this process. You have to go through a SYSGEN process; there is an entire manual devoted to it, but it really kind of assumes a knowledge of assembly language in which you create your CPM system and the SYSGEN process is where you put your own I/O modules in if you want to do that. You have the source code and at that point you can change it and reassemble it. Also, at that point, you define where you want it to reside. From then on until you do another SYSGEN, CPM never relocates itself, it loads directly into its final operating address. This has some advantages. One of them is that some people are doing spooling or simple multi-programming under CPM where your printer may print a listing even while you are using the editor or the assembler for something else and they do this by creating a partition above the operating system for the spooling software. You cannot do that under HDOS because he relocates to the top of the memory every time you boot. Tony will distribute CPM already SYSGENED for some small size system, 16K or 24K, most likely. If the user has more memory than that, he will have to do a partial SYSGEN to relocate the system, which isn't too bad. But if he wants to make major I/O changes, he will have to go through the complete SYSGEN process on his own. The Digital Research manuals that are provided, frankly, are not extremely good, especially for the novice. In fact, I would say the novice is probably going to have a good deal of difficulty with this. It is fairly demanding. However, it won't be necessary unless major changes are required, and we wrote the BIOS to support up to two teminals plus line printer on H8-4 and H8-5 cards, so most users won't have to fool with it. Also, Tony, on the North Star system, I know, and this will probably be the case on the Heath system, provides his own manual in addition to the standard Digital Research manual, describing how to do that. That will hopefully ease the process, but the user should appreciate that this is not likely to be a Heathkit quality manual. One consideration in choosing betweem CPM or HDOS is the ability level of the user and also the user's access to help.

Both Tony and Digital Research have their own Technical Consultant staffs available by phone to assist their users. However Heath will **not** be in a position to support CPM at this time.

HUG: So, which operating system should a person buy?

BW: It depends on his level of knowledge and his ability to do what he wants to do. Probably, the more sophisticated users, writing in several languages and more fimiliar with the system, would tend to gravitate to CPM, while the beginning user, or those who use mostly BASIC, would be happier with HDOS. In general, I would probably recommend most of our users have HDOS, if they've got the Heathkit system and that if they acquires CPM, it should probably be as a second system in addition to HDOS. There may be some sophisticated users who are already familiar with CPM and non-Heath systems that may choose to purchase CPM and an H17 disk system and never acquire HDOS at all and that is fine. But, for most of our users, probably they will either want just HDOS or both systems. Also, CPM does not come with any diagnostic software, so that is another reason for buying HDOS, especially for kit builders.

HUG: Tony mentioned he had about 30 or 40 eight inch diskettes full of stuff in the library. Briefly what kind of things could we look forward to?

BW: You name it. One of the things I can't really convey is just the incredibly vast quantity of software available for CPM. It is massive and it is mostly source code and the scope is just enormous. It runs all the way from games to assemblers and editors to a complete system-wide master catalog system that keeps a catalog of every file on every disk in your entire system even if you have dozens of diskettes. If you want to know for example what's on a given diskette, you can find out without getting out that diskette. If you want to know a list of every diskette on which a given file is located, you can find out. The software availablity for CPM is one of the strongest reasons for going to CPM. The other being the increased versatility it gives you, but the software availability is just fantastic. It is the equivalent roughly of 100 to 150 of our mini floppies. Mostly in source codes. That is the major argument for CPM and there is a lot of that stuff in BASIC, a lot of stuff in assembly language; and even some stuff in FORTRAN and other languages. There are also other languages in the users group, including a complete ALGOL Complier and complete disk BASIC Interpreter, source code, not object code. Several of them, not just one. There is a language called ML80 which is a macro-assembler with structured programming concepts, DO WHILE and DO UNTIL and this type of thing. The volume of software is incredible, mostly system software I guess and games, but there are also applications items in there, such as a complete general ledger system. And a word processing package among others. I just can't remember off hand all of the stuff available.

EDITORS NOTE: If you call one of the Heath Technical Consultants for any help, he will say, "huh?" In other words, they will not be in a position to support CPM.

EOF

COPYING DISC FILES FROM H17 TO H27

Doc Campbell

Note: To carry out this procedure, you will need 2 terminals (an H9 and a Decwriter, or 2 H9s, or equivalent), the H17 Floppy and the H27 Floppy, an H8 and a H11. Each computer will need 2 serial IO boards configured as follows. Note that these are all serial IO boards. Note that the following example uses an H9 connected to the H11 computer, and a Decwriter connected to the H8 computer.

1. The primary IO board in the H11 is assumed to be connected to the H9 as its primary output, ie to TT: with address 177560 and interrupt vector 060. The primary IO board in the H8 is configured to address its TT:, (in our case, it is jumpered to the DEC as a TT:, rather than as an AT:.)

2. The second IO card in the H8 must be jumpered to include the following parameters:

   a. Jumpered for AT: (AT: = PORT 374 OCTAL).
   b. RS 232.
   c. Jumpered for 9600 Baud.

3. The second IO card in the H11 must be jumpered to include the following parameters:

   a. Jumpered for H10, ie with address 177550 and interrupt vector 070.
   b. RS 232.
   c. Jumpered for 9600 Baud.
   d. C12 should be removed (if it is installed).

4. Now, a 2 wire cable is prepared to connect these second serial IO boards. The cable can be "tack-soldered" or appropriate plugs can be obtained.

   a. One wire goes from ground on one IO board to ground of other IO board. On the H11 rear panel socket from this IO board, pin 24 is ground. On the H8 rear panel socket from the 2nd board pin 4 is ground.
   b. The other wire goes from Serial Output of the second IO board in H8 to Serial Input on the H11 IO board. On the H11 rear panel socket, pin 2 is the input. On rear H8 panel socket from the 2nd board pin 8 is data output.
   c. Also, connect a jumper from pin 3 to pin 23 of rear panel socket on H11.
      Note that this rear socket on the H11 is the one coming from the second modified IO board.

5. The H8 is connected to the H17, and the H11 is connected to the H27 as usual.

6. The following procedure transfers the files from H17 disc to H27 disc.

   a. Bootup both systems and put both in PIP. If a new disc in H27, enter DK:/Z and CR on H9 to initialize disc.
   b. Enter the following on H9 (connected to H11) DK:filename.ext=PR:/A and CR.
   c. Enter the following on DEC (connected to H8) AT:=filename.ext and CR
   d. Wait until HDOS returns "FILE COPIED" on the Dec.
   e. Enter CNTRL D on the H9. (Nothing seems to happen here).
   f. Enter AT:=TT: on the DEC and CR.
   g. Type anything (a space is good), then CR on the DEC.
   h. Enter CNTRL D on the DEC.

AT THIS POINT THE FILE WILL BE WRITTEN FROM THE H17 DISC TO THE H27 DISC!!!!

NOTE: The size of the file to be transferred depends on computer memory space. With 32 K of memory in the H8 and 16 K in the H11, we found that a maximum of about 30 sectors in H8 (translates to 15 blocks in H11) could be transferred as one file. For larger data files, we simply made smaller files, transferred them and then concatenated them using the H27 disc operating system.

EDITORS NOTE:

Some of our Users have converted from the H8 System to the H11 System to gain more storage in business applications — Doc Campbell being one, the first, as soon as the H27 became available — So, Doc explains a little bit about the process —

:JB:

H8-17 DISC BASIC VERSUS HT11 DISC BASIC SYNTAX

There is no standardization of syntax among various Basic interpreters. A summary of syntax differences between H8-17 and HT-11 may prove helpful to someone who is revising H8-17 to HT-11 disc Basic programs. The MACRO (and other) features of HT-11 EDIT make for easy program revision.

These features are **not** supported in the current version of HT11 - BUILD, LOCK, UNLOCK, FREEZE, FREE, UNFREEZE, SEG (H8 panel function), CNTRL options, PEEK, POKE, STEP, UNSAVE, LNO function, PAD function, and PIN function. Any statements or functions just mentioned will have to be removed from programs. CIN is also not supported but if used for EOF marker is replaced by IF END...THEN; and SPC (not supported) may usually be replaced by TAB; CLEAR (in a program) and BOOLEAN operator functions are not supported, but it is relatively easy to alter programs in these instances to carry out the desired manipulations. (See below for examples.) The CHAIN function is supported, but sets all variables to zero or null. If "carryover" of variables is necessary, they must be placed in a temporary disc file and then accessed from the "chained" program.

(The following are **not** found in H8-17, but are present in HT11 - BIN function, OCT function, TRM$ function, RANDOMIZE, IF END...THEN, OVERLAY, VIRTUAL MEMORY FILE functions, DAT$, RENAME, and RESTORE number n file function.)

In the next list, the items on the left are the H8-17 statements - functions, and on the right are the equivalent HT-11 statements - functions. Any number in parentheses after the HT-11 version refers to a more detailed evaluation given in the numbered paragraphs that follow.

Multistatement Line Separator	
:	\
String Concatenation	
+	&
MID$(X$,Y,Z)	SEG$(X$,Y,Z+Y-1) (1.)
LEFT$(X$,N)	SEG$(X$,1,N) (1.)
RIGHT$(X$,N)	SEG$(X$,LEN(X$)-N+1,LEN(X$)) (1.)
MATCH(X$,Y$,N)	POS(X$,Y$,N)
OPEN fname FOR READ etc	OPEN fname FOR INPUT etc
OPEN fname FOR WRITE etc	OPEN fname FOR OUTPUT etc
To Reference Line Printer	
"AT:"	"LP:"
PRINT #X,	PRINT #X:
LINE INPUT #X,	INPUT #X: (2.)
INPUT #X,	INPUT #X: (2.)

(1.) The MID$ function of H8-17 is called the SEG$ function in HT-11. Also, there is a subtle difference. Whereas Z in the H8-17 function represents the total number of characters to be sampled, 'Z' in HT-11 = the position of the last character to be sampled. The position of the last character to be sampled is equal to the position of the first character to be sampled (Y) + the number of characters to be sampled (Z) less 1. Hence, Z in H8-17 syntax = Z + Y - 1 in HT11 syntax.

There is no LEFT$ function in HT-11. Use the equivalent SEG$ function listed.

There is no RIGHT$ function in HT-11. Use the equivalent SEG$ function listed.

If there is ever the possibility that X$ would be a null when used in the SEG$ function, in a program, then be sure to bypass the line with a line just preceding:

```
IF X$ = "" GOTO line number
```

Otherwise, your program will 'trap to 4' in the event of a null.

(2.) There is no LINE INPUT in HT-11 - INPUT covers both numerical values and strings.

HT-11 does not support a 'string-prompt' in INPUT statements. If the H8-17 program line reads:

```
30 LINE INPUT "ENTER YOUR NAME";N$
```

then substitute:

```
30 PRINT "ENTER YOUR NAME";\INPUT N$
```

MORE SYNTAX DIFFERENCES:

HT-11 does not contain a PAUSE function. You can accomplish an infinite delay by using INPUT #0:Z9$. (Z9$ is any variable that you do not use elsewhere in your program.) Entering a Return terminates this 'pause'. If you had used the PAUSE statement just to delay the program for a finite amount of time, simply substitute a 'do nothing' FOR...NEXT loop, such as -

```
50 FOR I = 1 TO 500\NEXT I
```

Here, vary the '500' to any number that supplies the desired delay.

There is a very subtle and insignificant difference in the way H8-17 and HT-11 handle FOR...NEXT loops. Following the loop, the variable is the number following the 'TO' plus 1, in H8-17, whereas it is the number in HT-11. For example, following completion of the loop given in the previous paragraph, I would = 501 in H8-17 but would = 500 in HT-11.

It is important to remember that IF...THEN, IF...GOTO, or IF...GOSUB statements, if false, go to the next sequential line in H8-17, BUT in HT-11 they go to the NEXT STATEMENT on a multistatement line (assuming there are multiple statements on the line)!

(continued next page)

In HT-11 IF...THEN is **only** allowed to reference a line number. An IF...THEN **statement** such as:
IF X = 5 THEN PRINT "X = 5"
is **NOT** allowed. This can be solved with:

```
30 IF X=5 GOTO 500
40 STOP
500 PRINT "X=5"\GOTO 40
```

'IF...THEN GOTO' and 'IF...THEN GOSUB' are **NOT** allowed. Use either THEN or GOTO (or GOSUB), but not both at the same time.

Since there are no BOOLEAN functions in HT-11, you must revise these portions of any programs. Consider the H8-17 statements:

```
30 IF X=5 AND Y=10 THEN 50
40 PRINT "X &/OR Y DO NOT = THE REFERENCED NUMBERS"
45 STOP
50 PRINT "X = 5 AND Y = 10":STOP
```

IN HT-11 this could be written:

```
30 IF X=5 THEN 800
40 PRINT "X &/OR Y DO NOT = THE REFERENCED NUMBERS"
45 STOP
50 PRINT "X = 5 AND Y = 10"\STOP
800 IF Y=10 GOTO 50\GOTO 40
```

Both of the above programs accomplish the same thing.

There are a few COMMANDS that are used differently. In H8-17, to continue a program after a STOP statement, one uses GOTO line number, Return, then CONTINUE. In HT-11 just use GOTO line number and Return. To exit a program in H8-17 one enters BYE and Return. In HT-11 one or two CNTRL C's takes you to the monitor.

If you wish to CLEAR variables in an HT-11 program you must set the variables to zero or null with statements such as:

```
X=0\Y$=""\FOR I=1 TO 50\D$(I)=""\NEXT I
```

Although H8-17 allows you to CLOSE a file which is not open, without an error message, HT-11 gives an error message if you attempt to CLOSE a file which is not open.

Last, although there is no UNSAVE statement/command in HT-11 you can recover disc file space (although you can not delete the file name) from HT-11 Basic by using the following statement/command:

```
OPEN "fname" FOR OUTPUT AS FILE #1\CLOSE #1
```

EOF

PROGRAMMING COURSE COMING. . .FINALLY! :JB:

ASSEMBLY LANGUAGE PROGRAMMING? WHAT'S IN IT FOR ME?

The most powerful language to use on any computer is ASSEMBLY LANGUAGE. Wait! Hear this out! It's not long. Consider the fact that ANYTHING that can be done in any other language on a particular computer can be done in Assembly Language. And, it is a true programming language, not just a convenient notation for machine code. The use of labels, computed operands, comments, and assembler directives allows the assembly language programmer to create a source code that is adaptable to the system and easily read. One of the most useful features of a programming language is that it is flexible, and not dependant upon specific addresses and hardware of the computer. In an assembly language program, hardware specifications can be made at the beginning and referred to symbolically through the rest of the program. In this way, any changes that may be needed are located together and are easily located. By using "computed" origin statements, and other assembler directives, you can make an assembly language program operate where routines along with as many remarks or comments as you desire, makes the program listing quite readable.

Perhaps the most fascinating fact concerning assembly language is that anything your computer is capable of can be done with assembly language. To run a program written in BASIC, the computer interprets the instruction and performs all the machine operations necessary to do that instruction. The programmer can do everything with assembly language that could be done in BASIC, because there is a mnemonic for every machine operation. Further, to run a program written in BASIC, the BASIC program must be in memory. After it has been assembled, an assembly language program performs the operations without that support, and because the instructions are not to be translated, the program runs much faster.

After reading this, you are probably interested in learning Assembly Language. HEATH CONTINUING EDUCATION has an assembly language programming course for 8080/8085 MPU based computers such as the H8. The course number is EC-1108 and it's described in your June 1979 HEATHKIT catalog.

NUMBERS —

By: Chuck Dattolo
METRO DETROIT AREA
HEATHKIT H8 COMPUTER
USERS' GROUP

Computers, contrary to common belief, are very simple minded devices. If you could look into the very heart of your system you would notice that it only contains a large array of switches. These electronic switches may be either on or off. A mechanism is provided to turn these on or off and to determine if they are on or off. About the only thing that a computer has going for it is the speed at which it can throw and 'read' the switches. The switches we have been referring to are more commonly called memory.

To simplify the design of the computer, it was decided to organize this memory into neat packages of eight switches per memory cell. A switch represents a 'bit'; a cell represents a 'byte'. If a computer is to compute it, obviously, must work with numbers. To a human, the digits 132 mean the numeric value one hundred thirty-two! That is obvious to everyone except our computer who must jam that figure into a bunch of little switches. A noble task even for a human.

If you look carefully at our example (132) you will note that it is made up of digits in the range of 0-9 and that the position of each digit in the number apparently means something. Specifically, the position to the right is the 'ones' column; the second to the right is the 'tens' column and the third to the right is the 'hundreds' column. In our system of numbering we find that $1 = 10^0$ and $10 = 10^1$ and $100 = 10^2$ etc. This magic number 10 is referred to as the 'base' or radix of the numbering system.

Since our computer only has a switch that can be either on or off, say a 1 or a zero we could acknowledge that it has only a two digit possibility. If a numbering system with 10 possibilities is called a 'decimal' system then one with two possibilities is a 'binary' system. So, we say that each digit is a binary digit or 'bit' for short. To the computer then, our example of 132 would look like 10 000 100. I'll leave it to you to verify that!

It wasn't very long before it became obvious to humans that binary numbers were cumbersome to use. To simplify this, man — not machine — grouped these bits into other bits to create a new numbering system. . . . .OCTAL — base 8 and HEXIDECIMAL — base 16. To see how this works, let's look at three bits. . .the possibilities of which are: 000,001,010,011,100,101,110,111. The right most digit is the 'ones' column, second to the right is the 'twos' column and the third to the right is the 'fours' column. Hence, digits of 0-7 can be represented. This is called the base eight or Octal. Similarly, a group of four bits giving a combination of 0-15 is called hexidecimal. Did you follow that? . . .Obviously not since you would have noticed that a digit can only be a single character, yet we have 10,11,12, 13, 14 and 15. To get around that, designers substituted A, B, C, D, E and F giving 16 single digits for the number system in the base 16 we abbreviatingly call 'HEX'.

Since a lot of people found it difficult to relate 'letters' to the numeric values of 10, 11, etc., many manufacturers adopted the octal format as a standard — including Heath, D.E.C. and others. Simple, huh? Well, even the editor of this DUMP can see that 8 isn't evenly divisible by three and what you end up with is 2 & 2/3 octal digits per word. Well, since the left digit can only be 0 to 3 that is no big problem. . .we simply note that the largest octal number that can be stored in the base 8 is 377. That relates to 255 in the base 10.

Still with me? Before you go running back to the Heath Store saying that your thousand dollar computer can't add higher than 255, let me give you the rest of the story.

In order for the computer to handle higher numbers it is necessary for it to combine 'words' back to back and treat them as bigger words. For example, two octal words back to back would be 16 bits with a maximum octal value of 177777 or a decimal value of 65535. If you noticed that our example was wrong; you are catching on. We did say that the maximum third digit value could only be 3, yet the example lists it as seven — 177(7)77. What happened is the extra bit came from the adjacent byte we added to make two words. Remember — 2 & 2/3 digits per octal word plus the second word gives us a combined word of 5 & 1/3 digit or 177 777. If you wanted to lessen the confusion and keep each octal word sacred you would do what Heath and others have done — create a modified octal known as split or offset octal. In the modified form, 177 777 would be represented as 377 377. Hope this makes sense to you. For a short lesson, try writing the 'pc' address of 040 100 that you use to execute a program in systems :true octal; decimal; and binary.

So far we have discussed how numbers could be represented in binary. We were assuming that the numbers were whole numbers (1,2,3 etc.). In computerese we refer to such (whole) numbers as 'INTEGERS'. In practice we often need numbers that contain fractional parts ($1.23, 123.456, 1.4375 etc.). Such fractions are referred to as 'REAL' numbers. The first attempt at handling real numbers utilitzed a technique called 'SCALING'. If you wished to represent dollars and cents a scaling factor of 100 was applied. . .so, $12.34 became 1234 (or 1,234) in integer. Once the input information was scaled then the calculations were made and the resulting product was rescaled back. That worked really nice since you were always dealing with two decimal places. Well, obviously not all applications neatly used two decimal places so it became a problem of trying to decide which scaling factor to apply.

Next comes the 'FIXED POINT' plan which attempted to correct the problems of scaling. In this system the 'point' was permanently placed at a specific location and kept there. In the example of a four byte number, we might say that the first

two bytes would represent the whole part of the number and the last two bytes identified the fractional portion with the 'point' being fixed between the second and third places. Fixed point had two major flaws. First, the size allowed for the whole part was often not large enough to accommodate the piece of datum that was unusually larger than normal sized data and secondly, the space allotted for the largest normally handled number was always using up memory space even though the actual entry was small. If a number is unusually large it actually will have a small number of significant digits. For instance, we might say that Lake Michigan had a capacity of 523,496,738,162,275.4375 gallons of water at 25°C. The significance of .4375 (7 cups) is fairly insignificant when 523 trillion gallons are being considered. Hence only three digits are probably significant. . .523!

To take advantage of this phenomenon, designers created a system that allowed the 'decimal point' to float and ingeniously called it 'FLOATING POINT'. It was quite similar to exponents in scientific notation where numbers like 0.234E4 are read as 2,340. In this example 0.234 is called the MANTISSA and (E) 4 is the exponent. Just how to represent the floating point in the computer turned out to be the 64.E3$ question. Everyone had their own idea, vanity prevailed and no standard arose. Most manufacturers chose some form of binary exponents and mantissas while others, like I.B.M., went with a hexidecimal exponent and a binary mantissa. Floating point structure is really a trade off between 1. storage requirement, 2. precision of the number, and 3. the range (size) of the number being considered. The old question of how much to rob Peter to pay Paul. In common practice 4 bytes are used to represent a number with 5 or 6 digits of precision. Special double precision is attained using 8 bytes to achieve 12 or 13 digits of precision (depending on the trade offs used). Since floating point became popular, manufacturers quickly designed hardware to manipulate the floating point. . .a box which often costs more than the original processor. Will this technology trickle down to the lowly 8080?. . .we're not sure. But, you can bet that some of the capabilities of floating point will be included in newer software releases.

We discussed how modern day computers translate decimal numbers into binary and how man decided to further simplify that representation into different number bases such as octal and hexidecimal. I don't want to give the impression that it was always done this way. . .in fact, the whole scheme was a result of trial and error. Octal and Hex were found to be the most efficient after exhaustive tries of other combination.

A while back, man had the brilliant (no pun intended) idea that if he took a bunch of light bulbs and arranged them in a block. . .say five across and seven high, that he could create the image of a number by selectiviely turning bulbs on and off. The idea is still used on billboards, time/temperature signs and to some degree in nixie tubes, 1.e.d.s etc. It didn't take those early pioneers too long to discover that the 35 bits (one for each bulb) it took to control each display was terribly inefficient.

Man, being very inventive, said that since there were only 10 numbers to represent the scheme could be encoded into four bits of binary code. Subsequent geniuses designed hardware items to recognize the binary 0000 and display it as a zero '0', by illuminating the appropriate lamps in the block. The name man gave to this encoding scheme was binary coded decimal or BCD for short. Later, letters were added to the capability as well as a full assortment of graphic symbols. The IBM company developed an "Extended Binary Decimal Interface Code" or EBCDIC. It required eight bits of information but could represent almost anything. Then, thinking that they (not Ford) had a better idea, the Teletype Corporation introduced 'BAUDOT' named after its inventor. They used five information bits to represent 58 characters. Not wanting to be outdone, a man called Holerith introduced another system of numbering for representing characters on punch cards.

It soon became obvious to the User that a standard was in order before things went totally out of control and so a committee was formed (sound familiar) to consider the task. The parameters which subsequently developed were called the 'American Standard Code for Information Interchange' and was quickly acronym'd ASCII. At last, everyone would use the same thing and peace would come to. . .what????. . . .IBM refuses to go along with this effort? Oh well, two standards are better than six! But, the same company that refused to join the American Standard also refused to join their own standard and created a third plan called correspondence code for use with their new Selectric type system. Misgivings plague everyone and certainly IBM is no exception; soon they had to give in to a re-design of the typing element so that epcdic and other codes could be accommodated. So, the new typing wonder isn't even standard in itself. Fortunately, most everyone uses ASCII and it is fairly safe to assert that if you have a program that outputs a '1' it will be recognized as a '1' on everything but an IBM product.

Both numbers in binary and printable characters in ASCII can be represented by a computer, even though the computer only sees them as a 'bunch of switches turned on or off'.

In future articles we will be looking at how negative numbers and one with fractional values may be represented as well as the ways of converting from one type to another. When the series ends we will hopefully understand the marvelous things our computer does when we type.

```
10  Let B=90
20  Print "See you in";B;"Days"
30  End
    Run
```

(Editors note: Chuck has begun writing a new language for the H-8 and 8080 based systems which will combine the benefits of basic and the advanced capabilities of Fortran — an extensive undertaking for one person. We wish you well with Forbasic Chuck.)

EOF

Reprinted from 'DUMP' with their permission — :JB:

TIDBITS ON MBASIC

By: Kathy Borden

EDITORS NOTE:

Our thanks to the documentation writer on the MBASIC project for preparing this article for us.

The MICROSOFT BASIC (MBASIC) has several features that are of interest to you. Obviously, they will be no more than touched on here.

PRINT USING

PRINT USING allows output, either to screen or line printer, of a specific format. The general format of the statement is PRINT USING <string>;<value list>, where <string> is a string expression, variable, or constant; and <value list> are the items to print.

In the following example, a set of backslashes () means a field of 2 spaces; any blanks enclosed by them enlarges the field. **$ inserts a $ immediately to the left of the last digit; any unused spaces will be filled with astericks. The original size of the field was determined by the ###'s. The comma forces commas to be printed in the resultant output where required.

```
10 F$="          **$##,###.##"
20 INPUT "WHATS THE NAME";N$
30 INPUT "WHATS THE AMOUNT";A
40 PRINT USING F$,N$,A
50 GOTO 20
OK
RUN
WHATS THE NAME? BORDEN
WHATS THE AMOUNT? 26
BORDEN      ******$26.00
WHATS THE NAME? BROWN
WHATS THE AMOUNT? 1.238
BROWN       *******$1.24
WHATS THE NAME? SCHAEFFER
WHATS THE AMOUNT? 123456789
SCHAEFFE   %$123,457,000.00
```

Note that the % indicates too large an entry; however, the entire value is rounded and displayed. A name respones that is too large is simply truncated; no error is generated.

RANDOM I/O

MBASIC also has RANDOM I/O abilities. This feature allows any record in a file to be accessed in minumum time. (A record is a sector, or 256 bytes.)

Some special characters to know (which are used in the following example) are:

```
I     Single precision variable (default)
I$    String variable
I%    Integer variable (see line 30 in the example)
I!    Single precision variable
I#    Double precision variable (see line 50)
```

(I and I! are the same, unless another definition is sent to I.) An @ symbol means a continuation line; that is, one BASIC statement is written on more than one physical line for programming legibility.

```
10 OPEN "R",1,"SY1:RFILE.EXT"
20 FIELD #1,32 AS A$, 8 AS B$
30 FOR I%=1 TO 10
40 LSET A$="RECORD NUMBER "+STR$(I%)
50 I#=I%+2D-09
60 LSET B$=MKD$(I#)
70 PUT #1,I%
80 NEXT I%
90 INPUT 'WHATS THE RECORD";I%
100 IF I%<=0 THEN 130
110 IF I%<=LOF(1) @
      THEN GET #1,I% : PRINT A$,CVD(B$) @
      ELSE PRINT "RECORD DOES NOT EXIST"
120 GOTO 90
130 CLOSE
140 END
OK
RUN
WHATS THE RECORD? 5
RECORD NUMBER  5                    5.000000002
WHATS THE RECORD? 10
RECORD NUMBER  10                   10.000000002
WHATS THE RECORD? 11
RECORD DOES NOT EXIST
WHATS THE RECORD? 1
RECORD NUMBER  1                    1.000000002
```

LINE	EXPLANATION
10	Open first file and name it.
20	FIELD reserves first 32 characters for A$ and the next 8 characters for B$.
40	Left justify and SET A$ to the string RECORD NUMBER concatenated with the integer value of I.
60	Left justify and SET B$ to the double precision value of I. (MKD$ turns I# into a string for storage.)
70	PUT (store) I in the buffer.
110	Analyze the input response. If the response exceeded the buffer end, LOF(1), tell the operator. If the response was in range, print the record number and the value. (CVD translates the stored string I back into its double precision numeric value.)

ERROR TRAPPING

If you want to analyze the cause of program errors, the capability to write an easy, effective error trapping routine is available. The ON ERROR GOTO statement forces the program to branch to your error routine when something incorrect happens. Your routine could, for example, print the failing step and error message, evaluate the probable cause, and re-enter the program at a different point to rectify the problem. Example:

```
10 ON ERROR GOTO 100
20 X = 1.2/0
30 PRINT X
40 NEXT I
50 STOP
100 IF ERR <> 11 THEN ON ERROR GOTO 0
110 PRINT "YOU CANT DIVIDE BY ZERO, DUMMY!"
120 PRINT "SEE LINE ";ERL
130 X=0
140 RESUME NEXT
150 END
OK
RUN
YOU CANT DIVIDE BY ZERO, DUMMY!
SEE LINE  20
 0
NEXT without FOR in 40
OK
```

Using an ON ERROR GOTO XXXX statement at the start of the program establishes that any error will cause a program branch to the routine. ERR contains the error code number and ERL has the line number that caused the failure.

The error code value for a divide by zero error is 11; a check is made to confirm that that is the error. The user's error message is output, followed by the step number, and the program resumes execution at step 30. (The line after the failing one.)

The ON ERROR GOTO 0 ensures that an unexpected error will print its own message; that is, your error routine will be ignored and program execution will stop as usual.

Continued on page 24.

BUGGIN'

HUG

Just a quicky this time. . .in case someone asks:

I discovered that issue 41, Space War, doesn't want to run with Ext.B.H.BASIC 10.05.00 (The new one for the H8-4, etc.) because the $Inbuf location is different and the program uses the buffer to get input instead of $Rchar.

Lines 3250, 3270, 3280, 3290, and 3300 need to have their PEEK and POKE statements changes from 8301 to 8271. Simple!

I've only played the game a few times at level 1, since it came only last Wednesday. It seems to accept input with the above changes. Looks like I'll need an H8-4 pretty soon for an H-14!

Ray Klatt
2538 Easy Street
Ann Arbor, MI 48104

I would like to bring to your attention a misprint I found in the number 4 issue of the HUG REMark Magazine. On page 16, in the subroutine for the money for matter, line 150 will not execute the HDOS basic the way it is printed. It should be changed to read:

```
150 IF L>L1 THEN A$=LEFT$(A$,L-
    L1)+","RIGHT$(A$,L1):GOTO 140
```

With these corrections the subroutine works very well.

— — — — —

I'm an avid Heathkit equipment user. My system consists of the H8, H9, H17, H36. I have 32K of memory and plan to expand it to 64K within the next couple of months.

I'm also interested in starting a computer club in the Joliet area. I would appreciate it if you could put something in your "meetings and Club Notices" column. The name of the Club is:

Joliet Computer Club
2359 F Plainfield Rd.
Crest Hill, Ill. 60435
Phone (815) 725-0749

Thank you for your attention to this matter.

James R. Sossong
CO-owner,
Business Computer Systems Inc. of Joliet

Dear HUG:

For the benefit of those who may have an unused parallel port or two on their H8-2, the following synopsis of programming instructions is offered. It should be noted that as I have both a parallel and a serial board installed in my H8, it was necessary to re-jumper the H8-2 Channel 0 to octal address 200-201 (128-129 decimal) so that both would work at the same time. Also, for those who, like me, have the H17 disk, you should note that the port jumpered to octal address 374-375 (252-253 decimal) is the address for outputting AT: as in the case of a printing terminal.

Since most of my work is in BASIC, the following is a conversion of the assembly instructions that would otherwise be entered via the front panel or in an assembly instruction program.

OUT 129,78	Initialize the UART at port 129
OUT 129,5	Enables data in/out, drive pin 12 high
OUT 129,39	Enables data in/out, drive pin 12 low
OUT 129,64	Resets the UART at port 129

The above instructions allow the device control pin on the CH 0 terminal strip to be set high (device off) or low (device on). It then becomes a simple exercise to use that signal to operate a relay circuit for external control of some device, which is, in fact, just what the signal does for a printer, etc. These same instructions would be used to program a parallel port to operate a guage device driver. The inclusion of a short BASIC routine can take care of it all, and at low speeds you'll never notice the delay. Of course, the H14 gets around all this.

The following is an example of a short BASIC demonstration program that will alternate the signal on pin 12 of the H8-2 terminal strip from logic high to logic low (best observed with a logic probe):

```
10 OUT 129,78: REM Program the
   UART;
20 OUT 129,5: REM Turn device control
   off (logic high)
30 PRINT "Device CONTROL OFF"
   :REM So you can see what happened.
40 PAUSE
50 OUT 129,39: REM: Turn device con-
   trol on (logic low)
60 PRINT "DEVICE CONTROL ON"
   :REM As in LN 30
70 PAUSE
80 GOTO 20: REM Loop for observation
   purposes
```

I hope this removes some of the mystery in the use of the H8-2 for things other than driving a paper punch or a teletype-like device. I'll be happy to correspond with anyone who has moved into the area of device control, either via BASIC or assembly language.

D. C. SHOEMAKER
720 Ottawa #100
Leavenworth, KS 66048

Dear HUG:

If you are (like me) the proud owner of a minimum configuration H-11 computer (8K), you have probably realized by now that the BASIC interpreter takes up almost all available memory, barely leaving enough space for trivial programs. You therefore have the choice between Assembly Language (provided you have LOTS of patience) or FOCAL (which you may be seeing for the first time). This article will outline a few "neat tricks" which will allow you to develop fairly sophisticated programs using FOCAL-4K, which is an interpreter (like BASIC), but which unfortunately runs much slower.

The first (and perhaps most striking) feature of FOCAL is its unique way of handl-

ing arrays: unlike BASIC (and FORTRAN and. . .) you do NOT need to DIMension arrays! This means that you do NOT reserve space for cells in the array you will never use. This is very convenient when handling "sparce matrices", that is, a large matrix (100×100, say) where only 5 to 10% of the cells are nonzero. Any FOCAL variable (including elements of arrays) whose value is set to zero (or which undefined) does not take up ANY space!

Another interesting feature is the Real Time Clock (FCLK) function which is enabled by removing a jumper on the H-11 power supply. Unfortunately, it can only time up to about 9 minutes before resetting to zero. This is due to a bug (oversight?) in the way FOCAL handles the EXTERNAL INTERRUPT (vector 100_8). In the 4K version, the internal code for the Interrupt Handling Routine is:

ADDRESS	CONTENTS	MNEMONIC
10646_8	Hi-order word of timer	HI
10650_8	Lo-order word of timer	LO
10652_8	005267	INC LO
10654_8	177772	
10656_8	005567	ADC HI
10660_8	177764	
10662_8	000002	RTI

The only problem is that . . .the INC instruction does NOT set the Carry flag! I have tried using an ADD instruction instead, but some other problem (?) seems to be mapping all values of LO between 100000_8 and 177777_8 onto values 0_8 to 77777_8 (ignoring the most significant bit), still causing the time to act funny. I would very much appreciate hearing from someone that knows of a solution to this problem.

The point that should be made to conclude this article is that it is worthwhile learning FOCAL if you are (like me) an individual who likes writing sophisticated programs in the minimum amount of time, but lack the $ required to invest in this additional 16K RAM board (you should see the price of these up here in Canada!).

Francois ROY
44 Bedard #310
HULL, Que., CANADA
J8Y 5Z7

--EDIT

H8 REGULATOR SOCKETS

HEATH ENGINEERING HAS SPECIFIED A NEW MOLEX CONNECTOR SHELL WHICH FORCES BETTER CONTACT BETWEEN THE 5v REGULATOR LEAD AND THE SPRING CONNECTOR — SEND HEATH YOUR PURCHASE DATE. . .AND QUANTITY NEEDED. ORDER HEATH PART NUMBER 432-1080 — THEY WILL BE SHIPPED N/C (SEE "H8 CRASH ARRESTER" ISSUE #5)

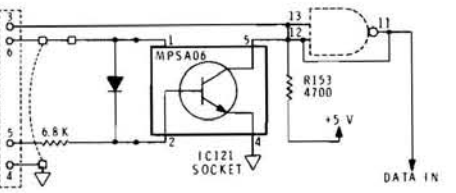
TRUE RS-232 FOR THE H8-5

Dear HUG:

Having begun my computer kit apprenticeship some fifteen years ago on a Heath EC-1 Analog Computer, I now feel qualified for a card as journeyman H8 assembler as a result of constructing several systems for use in our schools. Including my personal system, I have assembled to date: five H8's, twelve H8-1's (each with H8-3), ten H8-5's, and two H9's. As I gained experience troubleshooting the various circuits, several minor hardware problems were overcome with decreasing difficulty. These included such things as bad memory chips, faulty keyboard switches, a 7805 regulator which failed, and an incompletely etched control circuit board which left two data lines shorted.

I have recently solved an additional hardware problem which I believe is a design fault on the H8-5 serial board. A variety of terminals are being used with the five H8's. Console terminals consist of two H9's, 2 Microterm ACT IVa's, and a Digilog 33. Alternate terminals or printers connected to port 374Q are two Teletype ASR33's with DEC H313 interfaces and two Integral Data IP-255's. All of the nine I/O devices are connected via the RS 232 interface and therein lies the problem. Originally, only the two H9's and the Digilog 33 would transfer data to the H8-5 while the two ASR33's and the two ACT IVa's would not. After digging through all of my magazine back issues, I found sufficient information on the EIA RS-232C standard to indicate that the interface input impedance should be greater than 3000 ohms; the H8-5's input impedance is approximately 800 ohms. Most RS 232C terminals (excepting the H9) have an output impedance on the order of 2000 ohms which makes it impossible to light the LED in IC 121 on a "mark" input. I have solved this problem by modifying the H8-5's in the following manner:

1. Remove R152, 680 ohms
2. Change R151, 680 ohms to 6.8 K
3. Reverse D108, 1N4149
4. Remove IC121, 4N26
5. Vend out pins 11, 12, and 13 of IC122, 74LS00
6. Make solder bridge on back side of board between IC122-11 and 12
7. Install MPSA06 transistor in IC121 socket (form leads to fit): emitter to pin 4, base to pin 2, and collector to pin 5



The H8-5's being used with the IP-225 printers have been additionally modified so that the RS 232 input can be used with the EIA level printer CTS signal by removing the existing connections to IC 124-3 and 17 and connecting a jumper from IC 122-11 to IC-124-17. The port 374 Q boards also have all cassette I/O components omitted.

Several months ago I encountered a problem with Ext. BASIC 10.01.02 which manifested itself in a non-working TAN function.

The problem:

Bit 0 of address 040 073 must be clear in order for TAN to operate. If the bit happens to be set at power on, the software does not clear it on load and go which results in failure of the TAN function. (Problem doesn't exist in BASIC 10.02.01.)

The fix:

(1) Load configuration tape of BASIC 10.01.02.
(2) Key in changes —

```
040 101  060
040 102  104
104 060  076 000      MVI A
104 062  062 073 040  STA
104 065  041 156 041  LXI H
104 070  042 101 040  SHLD
104 073  303 156 041  JMP
```

(3) Set memory address to 104 075.
(4) Dump new configuration tape.

Another problem encountered which I have yet to find a solution involves the HDOS BASIC. When executing:

```
10  OPEN "AT:"FOR WRITE AS
    FILE #1
20  PRINT #1, CHR$ (12)
30  CLOSE #1
```

Eleven line feeds occur on the IP-225 printer rather than a form feed.

The line: 40 OUT 252, 12 produces a true form feed The same occurs on the ACT IVa console terminals:

PRINT CHR$ (12) produces eleven line feeds while
OUT 250, 12 produces form feed (clear screen).

I'll close with a suggestion: There should be a way to mount and dismount SY1: diskettes from BASIC without returning to HDOS. Microsoft can do it — why not Heath?

B. G. Chambers,
Science Supervisor

H-9 T.V. INTERFACE

Leon G. Biesiadecki
P.O. Box 2092
Ridgecrest, Cal 93555

There are many cases when a remote T.V. type display can be helpful in reducing a crowd around your H-9.

In preparation for an upcoming news display at a local county fair, much thought was given to leaving my H-9 out in the open so the fairgoers could read the news from a local radio station. I decided to install the following circuit in my H-9 allowing the use of a T.V. monitor (with a video input) as a large screen display. An R.F. adapter can be added to convert this signal to a standard T.V. channel.

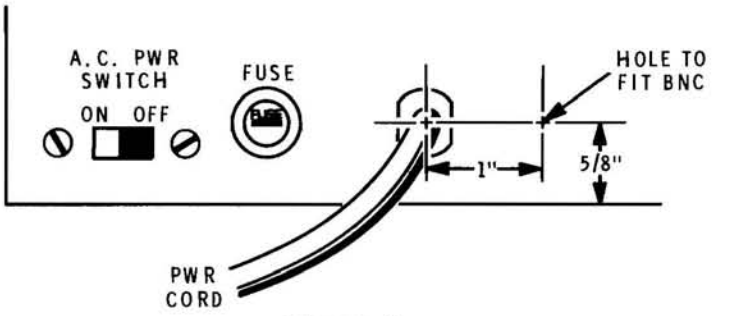


Figure 1

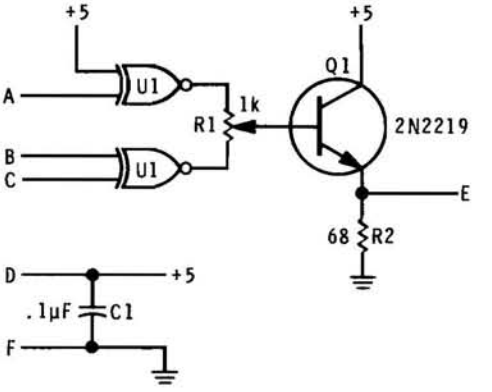


Figure 2

The installation of this circuit can be accomplished without modification of existing circuit boards. A hole is drilled in the rear panel of the H-9 as shown in figure 1. The size of this hole will vary, depending on the type of BNC connector you install at this location. Mount the P.C. board behind this connector next to the Character Generator board. Remove the Character Generator board and solder the wires from the T.V. Interface to P201 and P202. Take care not to short the pins together, and wrap the wire around the pin. Press the wire down as far as possible, and solder it in place.

Connect as follows:

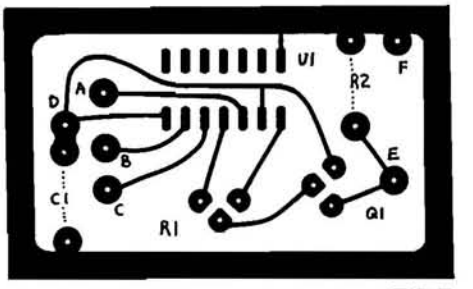
A (Video)	to P202 pin 1
B (V Sync)	to P202 pin 4
C (H Sync)	to P202 pin 2
D (+5)	to P201 pin 3
E (Vid Out)	to BNC Jack
F (Gnd)	to P201 pin 5

If a scope is available, connect to the BNC on the rear panel and adjust R1 for about 60% video and 40% sync. Otherwise, adjust R1 to midrange. Do not adjust to either limit of rotation.

This modification will now allow viewing the data on your H-9 from a remote location.

PARTS LIST

C1 — .1 uf 35v disc
R1 — 1K PC mount Trimpot
R2 — 68 Ω 1/4W
Q1 — 2n2219
U1 — 7486
Misc — Panel mount BNC,
P.C. board



EOF

BASIC IDEAS

Thanks to Bob Meister and his "H-11 BASIC PATCHER."

A BASIC bug that I found particularly troublesome was one which occurred during an input routine using "IF SEG$(A$,1,1)= THEN X" statement. If the return key is inadvertently pressed without having entered a character, then a reversion to a "COLD START" would occur.

My solution was to first test the input string using "LEN(A$) = 0". If the test is true, then print "INVALID ENTRY" and return to the input request statement. An example of the routine I now use for testing YES or NO inputs is listed on lines 10 through 60 with the additional lines ending in 5 shows for clarity.

```
05 PRINT "DO YOU WANT IN-
   STRUCTION"
10 INPUT A$
20 IF LEN(A$)=0 THEN 50
30 IF SEG$(A$,1,1)="Y" THEN
   105
40 IF SEG$(A$,1,1)="N" THEN
   205
50 PRINT "INVALID ENTRY,
   ANSWER YES OR NO"
60 GO TO 10
105 REM INSTRUCTIONS
205 REM PROGRAM
```

George Roth
11030 Richfield
Livonia MI 48150

In disk basic, there is the CIN() command. But, I have discovered a method of reproducing this command for tape basic (either extended issue). This procedure uses the input buffer POKE 8301. In normal operation, this address, 8301, is 0. But when something is typed on the console, the PEEK(8301) becomes equal to the length of the string typed. Thus, it is possible to bypass the need for the carriage return. For example, the question "DO YOU WANT INSTRUCTIONS?" can be answered this way by using the following program:

```
10 PRINT "DO YOU WANT IN-
   STRUCTIONS?";
20 POKE 8301,0
30 IF PEEK(8301)=0 THEN GO
   TO 30
40 B$=CHR$(PEEK(8302))
50 PRINT B$ : POKE 8301,0
```

If this program were run, the print-out would look like this:

```
DO YOU WANT INSTRUC-
TIONS? Y
```

where the portion underlined was inputted from the console. Note, though, that no carriage return was needed. The operation of the program is very simple. Line 10 prints the prompt, line 30 sets up a loop that does not end until something is typed on the console. As soon as something is typed on the console, the computer looks at the PEEK(8302), which is the first memory location which contains the actual ASCII code for what is typed in. But, since the computer only took 1 character from the console, all memory locations after 8302 would be meaningless. Line 40 puts the character taken into the string character B$. If you wanted a number typed in, then line 40 would look like this:

```
40 B=VAL(CHR$(PEEK(8302)))
```

This would put the character (numeric only) into the variable B. Line 50 prints the variable; that way the question looks as though it were done with one of the INPUT statements. Because you never really clear the input buffer, it is necessary to clear it by using the POKE command. By varying these lines a little, you can make interesting modifications, such as timed entries, and many other things.

Andrew Dessler
001401780308

VARIABLES SCRATCH PAD

It is a good idea to keep track of the variables in a program, especially in a large and complicated one. It is easy to lose track of which variables have been assigned for what purpose and which ones are still unassigned. When submitting a program for publication it is helpful if you can list the variables and their use. This program will print a scratch pad which will make this information easy to recall. It will increase the chances that a publisher will appreciate your program and accept it for publication. It will also become a permanent part of your records, enabling you to recall exactly what you did if that becomes necessary at a later date.

The program was written for use on my Baudot printer with a line length of only 72 characters. If your printer allows a longer line you may adjust the length by increasing the argument of the SPC(iexp) instructions in subroutines 1000 and 3000. A similar instruction can be added in subroutines 2000 and 4000 and the argument set for the desired spacing between variables.

Statement 130 is an indefinite PAUSE. This was put in to allow getting ready for the next page at my own speed and has no prompt statement because I did not want any superfluous printing at the end of the page. When you have the paper aligned, just hit the space bar and printing will continue. If you have a tractor feed printer, you will want to replace the PAUSE with the proper instructions to provide form feed onto the following page.

It is amazing how much easier it is to write a complicated program using this scratch pad. When you think you are getting cramped for new variables, a quick glance at the scratch pad reveals that there are actually bushels of possiblities remaining!! Imagine writing a program

which uses all of the available variables? There are 11 × 26 × 4 or 1144 possiblities! I hope this program will be of some help when you write one which uses them all.

G. L. "Jerry" Hale K0JH
6334 Edward Street
Norfolk, VA 23513

```
 LIST
5 REM //"VARIABLES SCRATCH PAD"  FOR HEATH H-8 COMPUTER
10 REM //ADAPTED BY GERALD L. HALE KØJH, FROM THE BASIC IDEA
15 REM //CONTAINED IN AN ARTICLE IN 'KILOBAUD' MAGAZINE MARCH 79
2Ø REM //WRITTEN BY ROBERT GOFF WHO GOT THE IDEA FROM 'ADVANCED
25 REM //BASIC', BY J.S. COAN, HAYDEN BOOK COMPANY, ROCHELLE PARK
3Ø Y$=" Ø123456789":Z$=" ABCDEFGHIJKLMNOPQRSTUVWXYZ"
4Ø PRINT  P=1
45 PRINT TAB(45);"PAGE";P:PRINT
5Ø PRINT "PROGRAM:";TAB(3Ø);"DATE:";TAB(45);"PROGRAMMER:"
6Ø PRINT :PRINT
7Ø FOR I+1 to 26
8Ø FOR J=1 to 11
9Ø ON P GOSUB 1ØØØ,2ØØØ, 3ØØØ,4ØØØ
1ØØ NEXT J
11Ø PRINT :PRINT
12Ø NEXT I
13Ø PAUSE
14Ø IF P=4 GOTO 9999
15Ø P=P+1:GOTO 45
1ØØØ PRINT MID$(Z$,I,1):MID$(Y$,J,1);SPC(3);:RETURN
2ØØØ PRINT MID$(Z$,I,1),MID$(Y$,J,1);"(";" ";")";" ";:RETURN
3ØØØ PRINT MID$(Z$,I,1);MID$(Y$,J,1);"S";SPC(2);:RETURN
4ØØØ PRINT MID$(Z$,I,1);MID$(Y$,J,1);"S";"(";" ";")";:RETURN
9999 END
```

Here are two programs both of which will solve the problem of reading in data that contains commas. (HDOS)

Program 1 line 60 makes use of the fact that line input inputs a complete line into a string allowing quotation marks and commas to be part of that string.

Program 2 line 30 and line 60 make use of the fact that if a comma is needed to be input in a string, then quotations marks must be used around the string that is being input. Line 30 uses the CHR$ function to place a quotation mark on both sides of the string C$ as it is written to the disk. Therefore, when C$ is read in by the input statement on line 60, the quotation marks are seen by the 10 routines and the comma between the quotation marks is not taken as a separator but as part of the string.

PROGRAM 1

```
10 LINE INPUT "CITY STATE, AND ZIP?";C$
20 OPEN "FILE.DAT" FOR WRITE AS FILE #1
30 PRINT #1, C$
40 CLOSE #1
50 OPEN "FILE.DAT" FOR READ AS FILE #1
60 LINE INPUT #1,; C$
70 CLOSE #1
80 PRINT C$
```

PROGRAM 2

```
10 LINE INPUT "CITY, STATE, ZIP?",C$
20 OPEN "FILE.DAT" FOR WRITE AS FILE #1
30 PRINT #1, CHR$(34); C$; CHR$(34)
40 CLOSE #1
50 OPEN "FILE.DAT" FOR READ AS FILE #1
60 IPUT #1,; C$
70 CLOSE #1
80 PRINT C$
```

```
10 GOSUB 10000
20 PRINT "FREE BYTES ="#I
30 END
10000 REM THIS SUBROUTINE RETURNS THE NUMBER OF FREE BYTES LEFT.
10010 REM SUBROUTINE IS ONLY VALID FOR EXTENDED BASIC 10.02.01
10020 REM
10030 REM NUMBER OF FREE BYTES LEFT IS RETURNED IN VARIABLE 'I'.
10040 REM VARIABLE 'J' IS USED AS A SCRATCH PAD.
10050 LET I=PEEK(17861)*256+PEEK(17860)-18049
10060 FOR J=17832 TO 17857 STEP 5
10070 LET I=I-PEEK(J+1)*256-PEEK(J)
10080 NEXT J
10090 RETURN
```

```
10 REM USING USER DEFINED FUNCTIONS FOR RIGHT$ LEFT$ AND MID$
20 \
30 OPEN "LP:" FOR OUTPUT AS FILE #1
40 DEF FNR(X$,A)=SEG$(X$,LEN(X$)-A+1,LEN(X$))\REM RIGHT$ FUNCTION
50 DEF FNL(X$,A)=SEG$(X$,1,A)\REM LEFT$ FUNCTION
60 DEF FNM(X$,A,B)=SEG$(X$,A,B+A-1)\REM MID$ FUNCTION
70 A$="JIM BLAKE"
80 PRINT #1:FNR(A$,4)
90 PRINT #1:FNL(A$,3)
100 PRINT #1:FNM(A$,5,3)
110 CLOSE #1
```

EOF

```
00010 REM  SAMPLE PROGRAM TO SHOW USE OF SUBROUTINES SUBMITTED
00012 REM  RUNS ON H8 WITH H9 TERMINAL ANY VERSION OF B.H. BASIC
00100 PRINT "CHECKBOOK PROGRAM":PRINT
00101 B$=" 0.00 "
00102 PRINT "ENTER CHECK AS NEGATIVE NUMBER OR DEPOSIT AS POSITIVE NUMBER";
00104 INPUT " :  ";X0
00106 GOSUB 1105
00107 I$=Y$
00108 W8$=B$:W9$=Y$                                                    BY:  GARY P. YEARGAIN
00112 GOSUB 1200                                                            3030 W. NEOSHA  RT 13
00120 PRINT                                                                 TUCSON, AZ  85705
00122 PRINT "INPUT WAS ";I$;" BALANCE IS ";Y$
00123 B$=Y$
00124 PRINT :PRINT :GOTO 102
01000 REM  ***** ROUTINES TO ADD OR SUBTRACT LARGE-NUMBER STRINGS *****
01008 REM
01009 REM  THIS ROUTINE WAS DEVELOPED TO OVERCOME THE LIMITATION OF
01010 REM  BASIC WHICH RESTRICTS NUMBERS TO SIX SIGNIFICANT FIGURES.
01011 REM  WHILE THIS LIMITATION DOES NOT AFFECT MOST PROGRAMS THERE
01012 REM  ARE TIMES WHEN IT IS A DRAWBACK, PARTICULARLY IN PROGRAMS
01013 REM  DEALING WITH MONEY WHEN NUMBERS MUST BE EXPRESSED TO EXACTLY
01014 REM  TWO DECIMAL PLACES AND MAY RUN TO MORE THAN SIX FIGURES.
01015 REM  IT IS INTENDED THAT THESE PROGRAMS BE CALLED AS SUBROUTINES
01016 REM  TO HANDLE THE ADDITION AND SUBTRACTION OF NUMBERS IN STRING
01017 REM  FORM.
01018 REM
01019 REM  RESTRICTIONS AND LIMITATIONS:
01020 REM  THE NUMBERS HANDLED BY THESE ROUTINES MUST BE IN STRING FORM.
01021 REM  EXAMPLE: A$=" 12345.67 "
01022 REM  NUMBERS MAY BE POSITIVE, NEGATIVE OR ZERO.
01023 REM  NUMBERS MUST HAVE A SPACE AFTER THE SECOND DECIMAL PLACE.
01024 REM  NUMBERS MUST BE OF MINIMUM LENGTH OF FIVE PLACES.
01025 REM  EXAMPLE: "X.XX "  SO THAT ALL VALUES LESS 1.00 WILL BE EXPRESSED
01026 REM  AS A ZERO, DECIMAL POINT, TWO ADDITIONAL DIGITS AND A SPACE.
01027 REM  TO CONVERT NUMBER VARIABLES TO STRING VARIABLES WHICH SATISFY
01028 REM  THESE CONDITIONS CALL SUBROUTINE 1100, WHICH IS A MODIFICATION
01029 REM  OF A ROUTINE TAKEN FROM A HEATH CO. PROGRAM.
01030 REM
01099 REM  ******************************************************
01100 REM  GENERATE A STRING FOR PRINTING A NUMBER THAT WILL GUARANTEE
01102 REM  TWO DIGITS AND A SPACE TO THE RIGHT OF THE DECIMAL POINT.
01103 REM  ENTRY: X0=NUMBER       EXIT: Y$=STRING
01105 Y$=STR$(INT(X0*100+SGN(X0)*0.5))
01106 IF X0>=0 GOTO 1112
01107 Z1=LEN(Y$)-3:Z1$=MID$(Y$,3,Z1)
01108 IF Z1>=3 GOTO 1110
01109 Z1$="0"+Z1$:Z1=Z1+1:GOTO 1108
01110 Y$=LEFT$(Y$,2)+MID$(Z1$,1,Z1-2)+"."+MID$(Z1$,Z1-1,2)+" "
01111 RETURN
01112 Z1=LEN(Y$)-2:Z1$=MID$(Y$,2,Z1)
01113 IF Z1>=3 GOTO 1117
01116 Z1$="0"+Z1$:Z1=Z1+1:GOTO 1113
01117 Y$=LEFT$(Y$,1)+MID$(Z1$,1,Z1-2)+"."+MID$(Z1$,Z1-1,2)+" "
01118 RETURN
01120 REM  ******************************************************
01200 REM ROUTINE TO ADD LARGE-NUMBER STRINGS
01201 REM ENTRY: W8$, W9$        EXIT: Y$
01202 REM STRING NUMBERS MUST BE MINIMUM LENGTH OF FIVE ("XX.X ")
01203 REM YOUR PROGRAM MUST SET W8$=FIRST NUMBER TO BE ADDED AND
01204 REM W9$=SECOND NUMBER TO BE ADDED THEN GOSUB 1200.  UPON RETURN
01205 REM Y$=STRING WHICH IS SUM OF VAL(W8$)+VAL(W9$) IN FORM "XXXX.XX "
01210 IF VAL(W8$)>=0 AND VAL(W9$)>=0 GOTO 1220
01212 IF VAL(W8$)<=0 AND VAL(W9$)<=0 GOTO 1240
01214 IF ABS(VAL(W8$))>=ABS(VAL(W9$)) GOTO 1260
01217 GOTO 1280
01220 W7=VAL(RIGHT$(W8$,5)):W8=VAL(LEFT$(W8$,LEN(W8$)-5))
01222 W5=VAL(RIGHT$(W9$,5)):W6=VAL(LEFT$(W9$,LEN(W9$)-5))
01224 W4=W8+W6:X0=W7+W5:IF X0>=10 THEN X0=X0-10:W4=W4+1
01226 GOSUB 1105:W7$=RIGHT$(Y$,LEN(Y$)-1)
01228 IF W4=0 THEN W6$=" ":GOTO 1234
01230 W6$=STR$(W4):W6$=LEFT$(W6$,LEN(W6$)-1)
01232 IF LEN(W7$)>5 THEN W7$=RIGHT$(W7$,LEN(W7$)-1)
01234 Y$=W6$+W7$
01236 RETURN
01240 W7=-VAL(RIGHT$(W8$,5)):W8=VAL(LEFT$(W8$,LEN(W8$)-5))
01242 W5=-VAL(RIGHT$(W9$,5)):W6=VAL(LEFT$(W9$,LEN(W9$)-5))
01244 W4=W8+W6:X0=W7+W5:IF X0<=-10 THEN X0=X0+10:W4=W4-1
01246 GOTO 1226
```

VECTORED TO PAGE 35

DECIMAL I/O WITH THE H8 MAINFRAME

Anyone who has built the Heathkit H8 will be aware that this is a highly versatile machine, even without a terminal. What may be less obvious is the fact that with a little extra effort it can be made to function as a sophisticated programmable calculator, complete with decimal input and output. (Do you find it hard to explain to your friends why you invested so much in a machine that can only count to seven?) This is accomplished by means of two subroutines — one for input via the front panel keyboard, the other for output on the display.

INPUT

At any point in a program, you can enter data via the keyboard by calling the subroutine already programmed in the Front Panel Monitor. 315(CALL) 260003 causes the program to halt execution and wait for you to push a key, causing the corresponding binary value to be placed in the Accumulator and processed in whatever ingenious manner you wish. Keys 0 through 9 put 0 through 9 in the accumulator; the remaining keys provide 10 through 15, and can serve as function keys to indicate end of input, type of data, thing to be done next, etc.

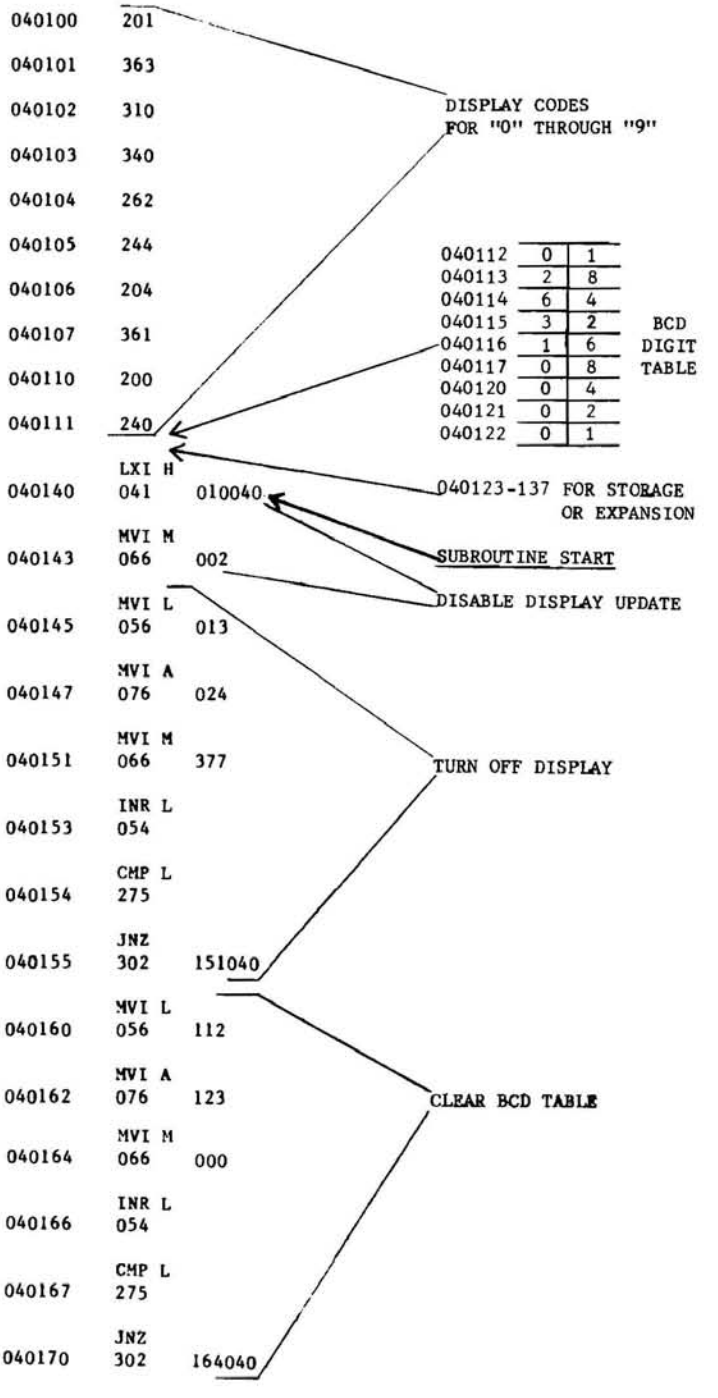
Since only one digit is processed at a time, you must call 260003 once for each digit in a given number. If you were entering a series of numbers 0 through 999, for each number your routine could do the following; take the first digit entered and multiply it by 100, take the second digit and multiply it by 10, take the third digit and add it to the first and second, and store the sum for future use.

Simultaneous display of the digits being entered can be provided by using parts of the Output subroutine, described next.

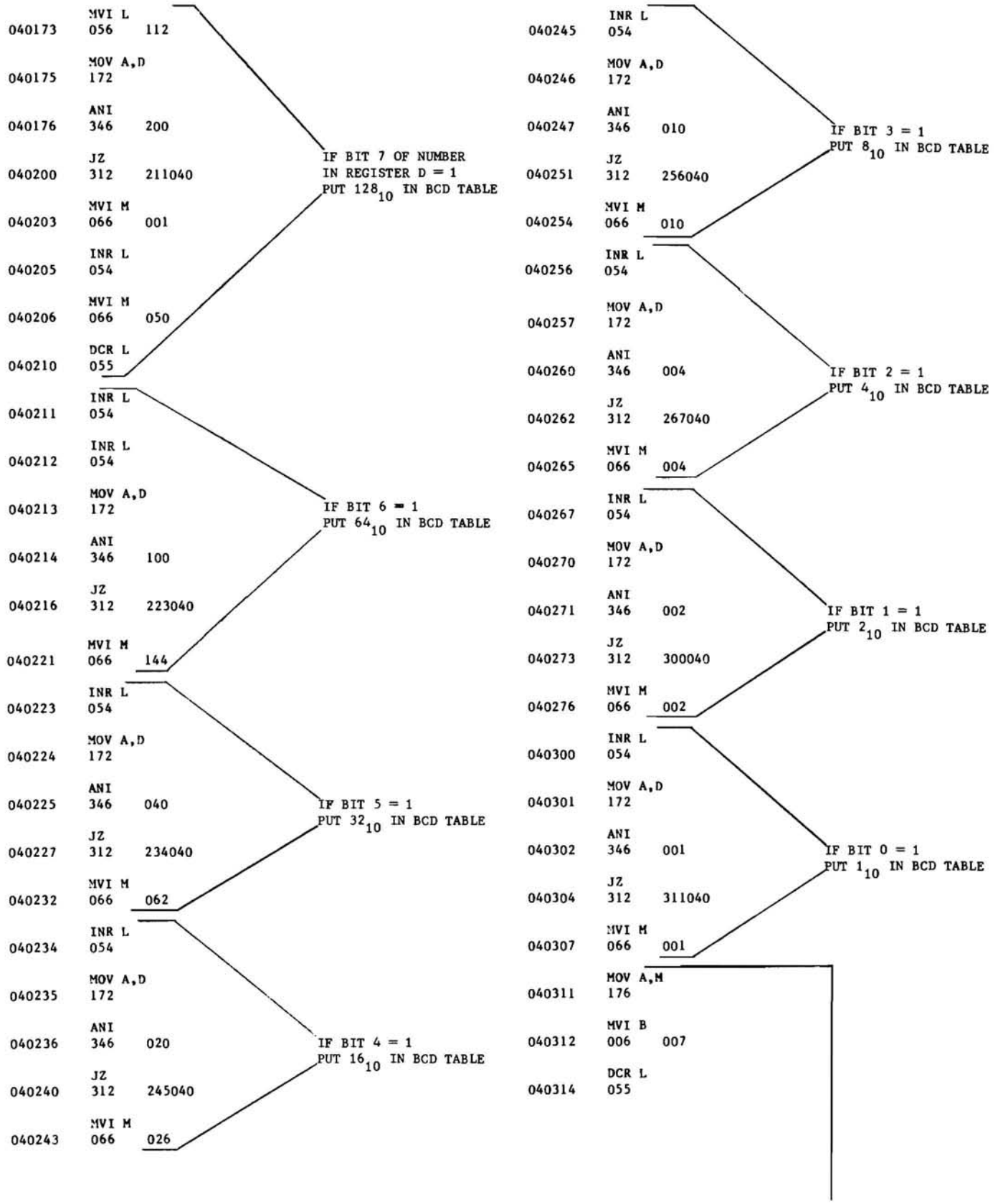
OUTPUT

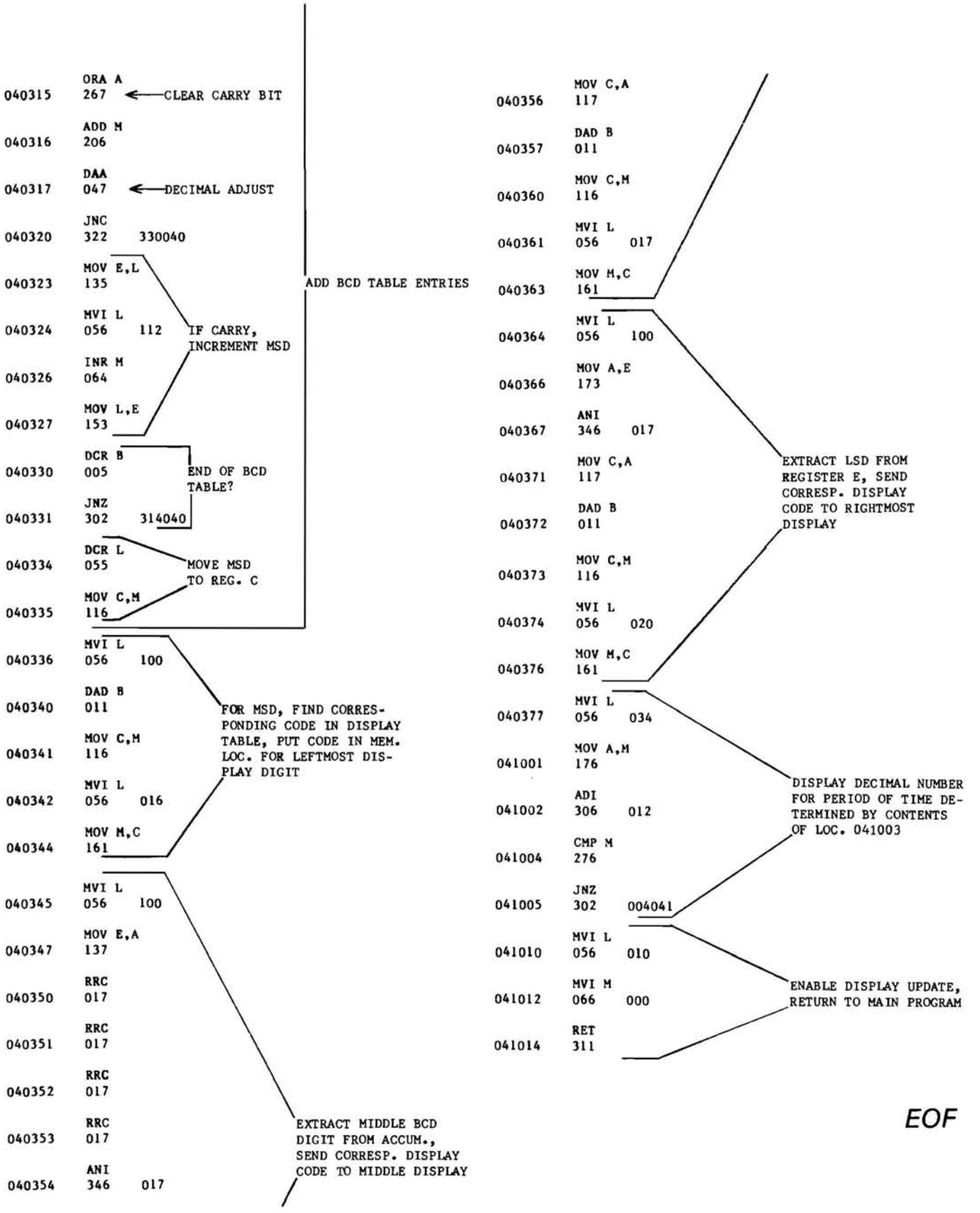
To illustrate the basic principles of this subroutine, a relatively simple version is described and listed here. It takes whatever number is present in Register D (i.e., up to 377_8), and displays it in decimal on the three middle display digits.

When the subroutine is called, it first disables display updating by the Front Panel Monitor, turns off the display and clears the BCD Digit Table. It then checks each bit in Register D; if a bit = 1, its BCD value is placed in the table. The values in the table are then added, and the three BCD digits comprising the sum are separately added to the bottom address of the Display Code Table, giving the address of the corresponding display code. The display code is then placed in the memory location for one of the three middle display digits. The decimal number is displayed for the period of time determined by the value in location 041003. The display time equals approximately this value times 0.5 second. When this time elapses, display control is returned to the Front Panel Monitor, and the subroutine is exited. The contents of Register D have not been altered.



By: ALAN DAY
907 BROWN AVENUE
CAMBRIDGE, OHIO 43725


MVI L
040173 056 112
MOV A,D
040175 172
ANI
040176 346 200
JZ
040200 312 211040
MVI M
040203 066 001
INR L
040205 054
MVI M
040206 066 050
DCR L
040210 055
IF BIT 7 OF NUMBER
IN REGISTER D = 1
PUT 128 10 IN BCD TABLE
INR L
040211 054
INR L
040212 054
MOV A,D
040213 172
ANI
040214 346 100
JZ
040216 312 223040
MVI M
040221 066 144
IF BIT 6 = 1
PUT 64 10 IN BCD TABLE
INR L
040223 054
MOV A,D
040224 172
ANI
040225 346 040
JZ
040227 312 234040
MVI M
040232 066 062
IF BIT 5 = 1
PUT 32 10 IN BCD TABLE
INR L
040234 054
MOV A,D
040235 172
ANI
040236 346 020
JZ
040240 312 245040
MVI M
040243 066 026
IF BIT 4 = 1
PUT 16 10 IN BCD TABLE
INR L
040245 054
MOV A,D
040246 172
ANI
040247 346 010
JZ
040251 312 256040
MVI M
040254 066 010
IF BIT 3 = 1
PUT 8 10 IN BCD TABLE
INR L
040256 054
MOV A,D
040257 172
ANI
040260 346 004
JZ
040262 312 267040
MVI M
040265 066 004
IF BIT 2 = 1
PUT 4 10 IN BCD TABLE
INR L
040267 054
MOV A,D
040270 172
ANI
040271 346 002
JZ
040273 312 300040
MVI M
040276 066 002
IF BIT 1 = 1
PUT 2 10 IN BCD TABLE
INR L
040300 054
MOV A,D
040301 172
ANI
040302 346 001
JZ
040304 312 311040
MVI M
040307 066 001
IF BIT 0 = 1
PUT 1 10 IN BCD TABLE
MOV A,M
040311 176
MVI B
040312 006 007
DCR L
040314 055


ORA A
040315 267 ←CLEAR CARRY BIT
ADD M
040316 206
DAA
040317 047 ←DECIMAL ADJUST
JNC
040320 322 330040
MOV E,L
040323 135
MVI L
040324 056 112
INR M
040326 064
MOV L,E
040327 153
IF CARRY, INCREMENT MSD
DCR B
040330 005
JNZ
040331 302 314040
END OF BCD TABLE?
DCR L
040334 055
MOV C,M
040335 116
MOVE MSD TO REG. C
ADD BCD TABLE ENTRIES
MVI L
040336 056 100
DAD B
040340 011
MOV C,M
040341 116
MVI L
040342 056 016
MOV M,C
040344 161
FOR MSD, FIND CORRESPONDING CODE IN DISPLAY TABLE, PUT CODE IN MEM. LOC. FOR LEFTMOST DISPLAY DIGIT
MVI L
040345 056 100
MOV E,A
040347 137
RRC
040350 017
RRC
040351 017
RRC
040352 017
RRC
040353 017
ANI
040354 346 017
EXTRACT MIDDLE BCD DIGIT FROM ACCUM., SEND CORRESP. DISPLAY CODE TO MIDDLE DISPLAY
MOV C,A
040356 117
DAD B
040357 011
MOV C,M
040360 116
MVI L
040361 056 017
MOV M,C
040363 161
MVI L
040364 056 100
MOV A,E
040366 173
ANI
040367 346 017
MOV C,A
040371 117
DAD B
040372 011
MOV C,M
040373 116
MVI L
040374 056 020
MOV M,C
040376 161
EXTRACT LSD FROM REGISTER E, SEND CORRESP. DISPLAY CODE TO RIGHTMOST DISPLAY
MVI L
040377 056 034
MOV A,M
041001 176
ADI
041002 306 012
CMP M
041004 276
JNZ
041005 302 004041
DISPLAY DECIMAL NUMBER FOR PERIOD OF TIME DETERMINED BY CONTENTS OF LOC. 041003
MVI L
041010 056 010
MVI M
041012 066 000
RET
041014 311
ENABLE DISPLAY UPDATE, RETURN TO MAIN PROGRAM

EOF

FAST STRING SORTING WITH HT-11

By: Tom Nelson

It is often useful, especially in business applications, to be able to sort records by customer name or account number. The HT-11 system has some sophisticated features which enable it to sort strings fast. Records can be alpha or numeric or both. They can be of variable length. And the sort can be in ascending or descending order.

The program described here makes use of the random access (virtual file) feature of HT-11. Instead of bringing the entire record into an array in core, only the key fields are read in, along with the "tag" or record location in the virtual file. This means that a large number of fields can be sorted in core. Then the tag is used to access the records from the virtual file in proper sequence and write them to an output file.

Input File Program

The following program, PAYMNT.BAS, is a simple version of a payment receipt input file. Notice that the data is written out to a string virtual file, VFI$. Numbers are converted to strings, and a comma is inserted for future use as a delimiter. Then the whole record is concatenated into a single string variable in statement 180.

As shown in the example, the file would accept up to 80 records, each of which could have up to 64 characters. (Remember that is you print this file using R PIP it will look strange because it's a virtual file.)

PAYMNT.BAS

```
100 A$="PAY1"
110 OPEN A$ FOR OUTPUT AS FILE
    VF1$(80)=64
120 C$="'"
130 I = 1
135 PRINT
140 PRINT I;"NAME";\INPUT N1$
150 PRINT "AMOUNT";\INPUT A
160 IF N1$="XXX" GO TO 300
170 A1$=STR$(A)
180 N$=N1$&C$&A1$
190 LET VF1(I)=N$
200 I=I+1
210 GO TO 135
300 PRINT "RECORDS MADE= ";I-1
310 STOP
320 END
```

Sort Program

Now that a data file has been created, we are ready to illustrate SORT.BAS. Suppose five records were in PAY1.DAT:

1 SMITH, 100
2 JONES, 75
3 MURPHY, 30
4 ANDERSON, 90
5 JACOBSON, 85

The sort program asks the name of the input file, PAY1; the name of the output file, for example SORT1; the number of the first sort column, for example 1; the number of records, five in this example; and the number of columns to be sorted, for example 4. It is usually not necessary to sort on a whole name. Punctuation and numbers could be in the sort field also.

The program first reads the sort field into array A$(I,1). It also stores the tag or index number of the record in word 0 of the array, A$(I,0). This is done in statements 210 to 240.

Sort Method

The program now starts to sort (statements 250-380). It uses a simple algorithm called the "exchange" method. It starts by storing the key field of the first record, A$(I,1), in S1$. The location of the record in the virtual file, A$(I,0) is stored in S2$. Going through the loop, if any smaller key field is encountered, its location in the loop is stored in P. After the loop has been completed, the value of A$ stored in the lowest position is exchanged with the values in position P (statement 330 to 350). Then the lower bound of the loop is incremented.

Thus with each pass, the lowest value is moved into the first position in the array and the loop is decreased in size by one value.

Once the array A$ has been sorted, its "tag", the record location in the virtual file, is used to randomly select the records in sorted order. The records are written out to an ordinary sequential file (statement 500 to 560).

In our example, after the sort is completed, the array A$ would contain:

Word 0	Word 1
4	ANDE
5	JACO
2	JONE
3	MURP
1	SMIT

Therefore, record VF1(4) is printed into file #2 first, followed by VF1(5) and so on. If it is necessary to have the file in descending order, simply reverse the index of the output to go from A1 to 1 in steps of −1.

Partitions and Merging

Most files that I sort are only 50 to 100 records long. However, if a really big file must be sorted, a combination of sort and merge can be used. Merging is a process of combining several smaller sorted files into one large one. This is done by reading one record from each of two input files, comparing their key fields and writing out the lowest. HT-11 makes a merge program easier with two features. First, the file names can be represented by a symbol such as F1$. Thus, a set of file names can be made by the program by concatenation: F1$="SORT"&"1", etc. Then a series of files such as SORT1, SORT2, etc. can be opened as they are needed by sort to accomodate an array of sorted records.

The second important feature is the test for end-of-file. HT-11 does this with an IF END statement. In merging it is essential that all files read to completion and that the output file closes normally after all records of all files are merged.

Conclusion

Contrary to the rather negative discussion of virtual files contained in the INTRODUCTORY OPERATIONS manual (p. 5-14 ff), I found the virtual file operation easy to work with and capable of making a fast and efficient sorting program.

SORT.BAS

```
100 DIM A$(80,1)          \REM CREATES A 25X2 MATRIX
110 PRINT TAB(15); "S T R I N G   S O R T"
120 PRINT "THE PROGRAM IS CURRENTLY DIMENSIONED AT 80 RECORDS"
130 PRINT "INPUT FILE NAME IS: ";\INPUT F1$
140 PRINT "OUTPUT FILE NAME IS:";\INPUT F2$
150 PRINT "KEY FIELD: WHICH COLUMN IS FIRST?";\INPUT C
160 PRINT "HOW MANY RECORDS?";\INPUT A1
180 PRINT "NUMBER OF COLUMNS IN KEY?";\INPUT C2
190 C1=C+C2-1                          Segment field
200 OPEN F1$ AS FILE VF1$(80)=64
210 FOR I=1 TO A1                      LOAD ARRAY
220 LET A2$=VF1(I)                     Read file
230 A$(I,1)=SEG$(A2$,C,C1)             Set keyfield = A$ 1
235 A$(I,0)=STR$(I)                    Set location = A$ 0
240 NEXT I
250 L=1                                Initialize lower loop bound to 1
260 P=L                                SORT
270 S1$=A$(L,1)                        Initialize position of small
272 S2$=A$(L,0)                        Set S = first value in loop
280 FOR I=L TO A1                      Also store location in virtual file
290 IF A$(I,1) > S1$ THEN 320          SEARCH LOOP  Test for smaller key field
300 P=I                                if smaller remember position in loop
310 S1$=A$(I,1)                        set new small string
312 S2$=A$(I,0)
320 NEXT I
330 A$(P,1)=A$(L,1)                    Exchange first in P
335 A$(P,0)=A$(L,0)
340 A$(L,1)=S1$                        Smallest into first
350 A$(L,0)=S2$
360 L=L+1                              Increase lower loop bound
370 IF L=A1 GO TO 490                  Test for end of sort
380 GO TO 260
490 OPEN F2$ FOR OUTPUT(10) AS FILE #2
500 FOR I=1 TO A1                      PRINT SORTED FILE
510 T1$=A$(I,0)
520 T1=VAL(T1$)                        Transform file location to numeric
530 PRINT T1,A$(I,1)                   Terminal printout
535 PRINT #2:VF1(T1)                   File print of original record
540 NEXT I
560 CLOSE #2
580 END
```

EOF

TIDBITS ON MBASIC

Continued from page 13.

LINE EDIT

Line edit can be used to clean up all the "crossed finger" typing inside a line. That is, you do not have to retype the entire line. It is useful to remember that the editing commands you enter will not be echoed by the terminal. They are printed in the following example so you'll know what to type.

```
   10 PRINT A,V,C,D     YOU ENTER
   EDIT 10              YOU TYPE (Edit line 10)
   10                   TERMINAL SCREEN SHOWS
      sv                YOU TYPE (Search for first V)
      PRINT A,          TERMINAL SCREEN SHOWS
              cb        YOU TYPE (Change character V to B)
              B         TERMINAL SCREEN SHOWS
               l        YOU TYPE (Print rest of line)
               ,C,D     TERMINAL SCREEN SHOWS
   10                   TERMINAL SCREEN SHOWS
      <cr>              YOU TYPE (terminates editing of line)
      PRINT A,B,C,D     TERMINAL SCREEN SHOWS
   LIST                 YOU REQUEST
   10 PRINT A,B,C,D     TERMINAL SCREEN SHOWS
   OK
```

Remember that Edit will NOT echo your commands, but act on them imemdiately. In the real world (your own program), this will be fine. You will be able to easily change a character or group of characters within a line. Try it when you get your copy of MBASIC.

SOME EXTRA TREATS:

Variable names can be any length, so long as the first character is alphabetic. However, only the first two characters determine the variable name, so SLIP=SLID.

POKE and PEEK let you pry around in memory.

DOUBLE PRECISION numbers allow up to 17 digits of accuracy.

RENUM will renumber and reincrement any part of the program. (It cannot reorganize, alas.)

EOF

HT11 INVENTORY PROGARM

```
10 REM INVENTORY
20 PRINT  \ PRINT 'INVENTORY VER #1.0'
30 PRINT '* * * * * * *  INVENTORY MENU  * * * * * * * * * *'
40 PRINT 'THESE ARE THE PROGRAM OPTIONS:'
50 PRINT TAB(10);''ADD'';TAB(50);''LOOKUP''
60 PRINT TAB(10);''LIST'';TAB(50);''DELETE''
70 PRINT TAB(10);''REORDER'';TAB(50);''ADJUST''
80 PRINT TAB(10);''INIT' DISKETTE';TAB(50);''DONE''
90 Z1=0
100 N=7 \ DIM R1$(7) \ FOR I=0 TO N \ READ R1$(I) \ NEXT I
110 PRINT  \ PRINT 'ENTER REQUEST ';
120 INPUT R$
130 FOR I=0 TO N
140 IF R$=R1$(I) THEN 180
150 NEXT I
160 PRINT 'INVALID REQUEST'
170 GO TO 110
180 ON I+1 GO TO 190,300,590,660,740,810,1070,1280
190 REM 'INIT'.
200 IF Z9=1 THEN 160
210 OPEN 'INVPN' FOR OUTPUT AS FILE VF1$(1000) \ CLOSE VF1
220 OPEN 'INVDSC' FOR OUTPUT AS FILE VF2$(1000)=32 \ CLOSE VF2
230 OPEN 'INVQ' FOR OUTPUT AS FILE VF3%(1000) \ CLOSE VF3
240 OPEN 'INVRP' FOR OUTPUT AS FILE VF4%(1000) \ CLOSE VF4
250 OPEN 'INVRQ' FOR OUTPUT AS FILE VF5%(1000) \ CLOSE VF5
260 OPEN 'INVCST' FOR OUTPUT AS FILE VF6(1000) \ CLOSE VF6
270 OPEN 'INVDAT' FOR OUTPUT AS FILE VF7%(1)
280 VF7(0)=-1 \ CLOSE VF7
290 GO TO 110
300 REM 'ADD.'
310 Z1=1 \ GOSUB 1380
320 IF J=0 THEN 420
330 IF J=2 THEN 360
340 PRINT 'ENTRY FOR PART ';P$;' ALREADY EXISTS'
350 GO TO 110
360 PRINT 'A DELETED ENTRY FOR ';P$;' EXISTS'
370 PRINT 'DO YOU WISH TO REINSTATE IT ';
380 INPUT A$
390 IF A$<>'YES' THEN 110
400 VF1(I)=SEG$(VF1(I),2,LEN(VF1(I)))
410 GO TO 110
420 IF VF7(0)<>1000 THEN 450
430 PRINT 'INVENTORY FILE IS FULL'
440 GO TO 110
450 A=VF7(0)+1
460 VF1(A)=P$
470 PRINT 'DESCRIPTION '; \ INPUT D$ \ VF2(A)=D$
480 PRINT 'QUANTITY '; \ INPUT Q \ VF3(A)=Q
490 IF VF3(A)>=0 THEN 510
500 PRINT 'INVALID ENTRY' \ GO TO 110
510 PRINT 'REORDER POINT '; \ INPUT Q \ VF4(A)=Q
520 IF VF4(A)<0 THEN 500
530 PRINT 'REORDER QUANTITY '; \ INPUT Q \ VF5(A)=Q
540 IF VF5(A)<0 THEN 500
550 PRINT 'COST '; \ INPUT Q \ VF6(A)=Q
560 IF VF6(A)<0 THEN 500
570 VF7(0)=A
580 GO TO 110
590 REM 'DELETE.'
600 Z1=1 \ GOSUB 1380
610 IF J=1 THEN 640
620 PRINT 'ENTRY FOR PART ';P$;' DOESN'T EXIST'
630 GO TO 110
640 VF1(I)='D'&VF1(I)
650 GO TO 110
660 REM 'ADJUSTMENT'
670 Z1=1 \ GOSUB 1380
680 IF J<>1 THEN 620
690 PRINT 'ENTER QUANTITY ADJUSTMENT '; \ INPUT Q
700 IF Q>=0 THEN 720
710 IF VF3(I)+Q<0 THEN 500
720 VF3(I)=VF3(I)+Q
730 GO TO 110
740 REM 'LOOKUP'.
750 Z1=1 \ GOSUB 1380
760 IF J<>1 THEN 620
770 PRINT VF2(I)
780 PRINT 'QUANTITY =';VF3(I)
790 PRINT 'COST =';VF6(I)
800 GO TO 110
810 REM 'LIST'.
820 Z1=0 \ GOSUB 1380
830 OPEN 'LP:' FOR OUTPUT AS FILE #1
840 PRINT #1:CHR$(27);CHR$(117);CHR$(32);
850 I=0 \ J=50 \ K=0
860 IF I>VF7(0) THEN 1040
870 IF SEG$(VF1(I),1,1)='D' THEN 1020
880 IF J<50 THEN 950
890 K=K+1 \ J=0
900 PRINT #1:DAT$;TAB(45);'INVENTORY LISTING';
910 PRINT #1:TAB(105);'PAGE ';K \ PRINT #1: \ PRINT #1:
920 PRINT #1:'PART #';TAB(20);'DESCRIPTION';TAB(55);'QUANTITY';
930 PRINT #1:TAB(65);'REORDER POINT';TAB(85);'REORDER QUANTITY'
940 PRINT #1:TAB(105);'COST' \ PRINT #1:
950 PRINT #1:VF1(I);
960 PRINT #1:TAB(20);VF2(I);
970 PRINT #1:TAB(58);VF3(I);
980 PRINT #1:TAB(70);VF4(I);
990 PRINT #1:TAB(95);VF5(I);
1000 PRINT #1:TAB(105);VF6(I)
1010 J=J+1
1020 I=I+1
1030 GO TO 860
1040 PRINT #1:CHR$(12)
1050 CLOSE #1
1060 GO TO 110
1070 REM 'REORDER'.
1080 Z1=0 \ GOSUB 1380
1090 OPEN 'LP:' FOR OUTPUT AS FILE #1
1100 PRINT #1:CHR$(27);CHR$(117);CHR$(1);
1110 I=0 \ J=50 \ K=0
1120 IF I>VF7(0) THEN 1250
1130 IF SEG$(VF1(I),1,1)='D' THEN 1230
1140 IF VF3(I)>VF4(I) THEN 1230
1150 IF J<50 THEN 1210
1160 K=K+1 \ J=0
1170 PRINT #1:DAT$;TAB(30);'REORDER REPORT';
1180 PRINT #1:TAB(75);'PAGE ';K \ PRINT #1: \ PRINT #1:
1190 PRINT #1:'PART #';TAB(20);'DESCRIPTION';TAB(55);
1200 PRINT #1:'REORDER QUANTITY' \ PRINT #1:
1210 PRINT #1:VF1(I);TAB(20);VF2(I);TAB(62);VF5(I)
1220 J=J+1
1230 I=I+1
1240 GO TO 1120
1250 PRINT #1:CHR$(12)
1260 CLOSE #1
1270 GO TO 110
1280 REM 'DONE'.
1290 IF Z9=0 THEN 1370
1300 CLOSE VF1
1310 CLOSE VF2
1320 CLOSE VF3
1330 CLOSE VF4
1340 CLOSE VF5
1350 CLOSE VF6
1360 CLOSE VF7
1370 STOP
1380 REM 'FIND ENTRY'.
1390 IF Z9<>0 THEN 1480
1400 OPEN 'INVPN' AS FILE VF1$(1000)
1410 OPEN 'INVDSC' AS FILE VF2$(1000)=32
1420 OPEN 'INVQ' AS FILE VF3%(1000)
1430 OPEN 'INVRP' AS FILE VF4%(1000)
1440 OPEN 'INVRQ' AS FILE VF5%(1000)
1450 OPEN 'INVCST' AS FILE VF6(1000)
1460 OPEN 'INVDAT' AS FILE VF7%(1)
1470 Z9=1
1480 IF Z1=0 THEN 1610
1490 PRINT 'ENTER PART NUMBER '; \ INPUT P$
1500 IF VF7(0)=-1 THEN 1550
1510 FOR I=0 TO VF7(0)
1520 IF P$=VF1(I) THEN 1570
1530 IF 'D'&P$=VF1(I) THEN 1590
1540 NEXT I
1550 J=0
1560 RETURN
1570 J=1
1580 RETURN
1590 J=2
1600 RETURN
1610 RETURN
1620 DATA 'INIT','ADD','DELETE','ADJUST','LOOKUP','LIST'
1630 DATA 'REORDER','DONE'
```

EOF

USING THE ET-3400

MORSE CODE INSTRUCTION

By: *Louis C. Graue, K8TT*
624 Campbell Hill Road
Bowling Green, Ohio 43402

Perhaps some ET-3400 owners would like to learn Morse code in order to become a ham. The program presented selects a character at random from a table, sends it in perfect morse code through a speaker, pauses for a while, and then displays the character on a LED readout. The student listens to the code, tries to write it down and checks the result on the readout.

If one is just beginning, it would be best to set the number in address 00a1 at 05_{16} instead of 24_{16} so that the characters will be randomly selected from just the first five letters. After these are learned increase the number to include a few new characters, and so on. The speed of the sending can be adjusted by changing the delay loop.

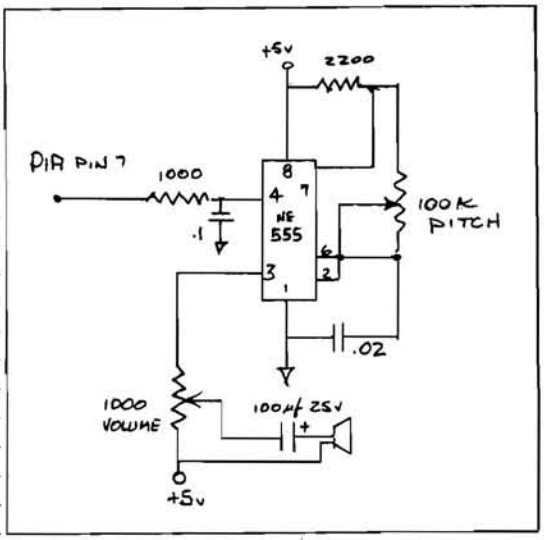
The speaker is operated by the processor through the circuit mounted on the breadboard. See page 28.

ADDRESS	CONTENTS	LABEL	OP CODE	OPERAND	COMMENTS
0000	86 , 01 ,		LDA	A #$01	Set up high port
0002	B7 , 01 , Fd		STA	A	of address of random number
0005	CE , FF , 04		LDX	#$FF04	Make P/A ports
0008	FF , 80 , 00		STX		$A_0,\ldots,A_7$ outputs
000B	Bd , 00 , 50	START	JSR	CLRDIS	Clear the displays
000E	C6 , 03 ,		LDA	B #$03	Set up pointer for
0010	F7 , 01 , FC		STA	B	Code groups
0013	Bd , 00 , 90		JSR	RANNUM	Generate a random number
0016	FE , 01 , Fd		LDX		Set up table pointer
0069	A6 , 00 ,	GROUP	LDA	A 0,X	Get code from table
006B	C6 , 08 ,		LDA	B #$08	Set up pointer for sending code
001d	B7 , 80 , 00	BIT	STA	A	Start code output
0020	48 , ,		ASL	A	continue sending
0021	5A , ,		DEC	B	Keep track of element sent
0023	* Bd , 00 , 80		JSR	Code speed	Delay to set code speed
0026	5d , ,		TST	B	Have all 8 bits been processed
0027	26 , F5 ,		BNE	BIT	If not, go get next bit
0029	08 , ,		INX		Otherwise, get set for next group
002A	F6 , 01 , FC		LDA	B	Set pointer to keep track of 3 code groups
002d	5A , ,		DEC	B	Counting
002E	F7 , 01 , FC		STA	B	Save pointer
0031	26 , E7 ,		BNE	GROUP	Have the 3 groups been sent
0033	A6 , 00 ,		LDA	A 0,X	If so, get ready for display of code character
0035	Bd , 00 , 70		JSR	DELAY	Wait a while
0038	Bd , FE , 3A		JSR	OUTCH	Then display character
003B	Bd , 00 , 70		JSR	DELAY	and hold it for a while
003E	20 , CB ,		BRA	START	Repeat
CLEAR DISPLAY AND DELAY					
CLRDIS	, ,				

*0022 should be a NO-OP

ADDRESS	CONTENTS	LABEL	OP CODE	OPERAND	COMMENTS
0050	FF , 01 , FA		STX		Save index pointer
0053	Bd , FC , BC		JSR	REGDIS	Reset to left-most display
0056	4F , ,		CLR	A	Clear
0057	C6 , 06 ,		LDA	B #6	
0059	Bd , FE , 3A	CLRD	JSR	OUTCH	o
005C	5A , ,		DEC	B	
005d	26 , FA ,		BNE	CLRD	Displays
005F	Bd , FC , BC		JSR	REGDIS	Reset to left-most display
0062	FE , 01 , FA		LDX		Restore index pointer
0065	39 , ,		RTS		Return to calling program
DELAY	, ,				
0070	FF , 01 , FA		STX		Save index pointer
0073	CE , FF , FF		LDX	#65535	Hold
0076	09 , ,	Wait	DEX		Display
0077	26 , Fd ,		BNE	Wait	For a while
0079	FE , 01 , FA		LDX		Restore index pointer
007C	39 , ,		RTS		Return to calling program
CODE SPEED	, ,				
0080	FF , 01 , FA		STX		Save index pointer
0083	CE , 0C , FF		LDX		Set
0086	09 , ,	Wait	DEX		Code
0087	26 , Fd ,		BNE	Wait	Speed
0089	FE , 01 , FA		LDX		Restore index pointer
008C	39 , ,		RTS		Return to calling program
RANDOM NUMBER GENERATOR					
RANNUM	, ,				
0090	CE , 01 , FE		LDX	#$01FE	Set pointer to random number
0093	A6 , 00 ,		LDA	A 0,X	Fetch random number
0095	49 , ,	NN	ROL	A	Make a
0096	A8 , 00 ,		EOR	A 0,X	New random number
0098	GC , 01 ,		INC	1, X	Increment the addend
009A	AB , 01 ,		ADD	A 1,X	Add to addend
009C	28 , 02 ,		BVC	✓	If V=0, increment addend once
009E	AB , 01 ,		INC	1,X	Otherwise, increment it twice
00A0	81 , 24 ,	✓	CMP	A #$24	Is number more than $24
00A2	22 , F1 ,		BHI	NN	If more, get another number
00A4	48 , ,		ASL	A	Make the number
00A5	48 , ,		ASL	A	A multiple of 4
00A6	A7 , 00 ,		STA	A 0, X	Store new random number
00A8	39 , ,		RTS		Return to calling program
CODE TABLE	, ,				
0100	B8 , 00 , 00				dit-dah
0103	77 , ,				7 segment code for letter A
0104	EA , 80 , 00				dah dit dit dit
0107	IF , ,				7 segment code for letter B
0108	EB , A0 , 00				dah dit dah dit
010B	4E , ,				C
010C	EA , 00 , 00				dah dit dit
010F	3d , ,				D
0110	80 , 00 , 00				dit
0113	4F , ,				E
0114	AE , 80 , 00				dit dah dit dit
0117	47 , ,				F
0118	EE , 80 , 00				dah dah dit
011B	7B , ,				G
011C	AA , 00 , 00				dit dit dit dit
011F	17 , ,				H

ADDRESS	CONTENTS	LABEL	OP CODE	OPERAND	COMMENTS
0120	A0 , 00 , 00				dit dit
0123	06 , ,				I
0124	BB , B8 , 00				dit dah dah dah
0127	38 , ,				J
0128	EB , 80 , 00				dah dit dah
012B	07 , ,				K
012C	BA , 80 , 00				dit dah dit dit
012F	0E , ,				L
0130	EE , 00 , 00				dah dah
0133	76 , ,				M
0134	E8 , 00 , 00				dah dit
0137	15 , ,				N
0138	EE , E0 , 00				dah dah dah
013B	1d , ,				O
013C	BB , A0 , 00				dit dah dah dit
013F	67 , ,				P
0140	EE , B8 , 00				dah dah dit dah
0143	63 , ,				Q
0144	BA , 00 , 00				dit dah dit
0147	05 , ,				R
0148	A8 , 00 , 00				dit dit dit
014B	5B , ,				S
014C	E0 , 00 , 00				dah
014F	0F , ,				T
0150	AE , 00 , 00				dit dit dah
0153	1C , ,				U
0154	AB , 80 , 00				dit dit dit dah
0157	3E , ,				V
0158	BB , 80 , 00				dit dah dah
015B	3F , ,				W
015C	EA , E0 , 00				dah dit dit dah
015F	37 , ,				X
0160	EB , B8 , 00				dah dit dah dah
0163	27 , ,				Y
0164	EE , A0 , 00				dah dah dit dit
0167	49 , ,				Z
0168	BB , BB , 80				dit dah dah dah dah
016B	30 , ,				1
016C	AE , EE , 00				dit dit dah dah dah
016F	6d , ,				2
0170	AB , B8 , 00				dit dit dit dah dah
0173	79 , ,				3
0174	AA , E0 , 00				dit dit dit dit dah
0177	33 , ,				4
0178	AA , 80 , 00				dit dit dit dit dit
017B	5B , ,				5
017C	EA , A0 , 00				dah dit dit dit dit
017F	5F , ,				6
0180	EE , A8 , 00				dah dah dit dit dit
0183	70 , ,				7
0184	EE , EA , 00				dah dah dah dit dit
0187	7F , ,				8
0188	EE , EE , 80				dah dah dah dah dit
018B	73 , ,				9
018C	EE , EE , E0				dah dah dah dah dah
018F	7E , ,				Ø
0190	AE , EA , 00				dit dit dah dah dit dit
0193	65 , ,				?



EOF

BITS AND NIBBLES :JB:

HDOS BASIC TIME SAVER —

When you load BASIC, about 2 k is left on disk to conserve memory — but it is slow. For instance, BASIC has to run out to the disk to grab an error message. . .irritating. However, typing control 4,1 before 'RUN'ing a program causes all of BASIC to be resident in memory but this cannot be done under program control. Happily there is a way around this. Load BASIC and type CNTRL 4,1. Type FREEZE. All of BASIC will be saved on disk with the extension, .BAF.Delete BASIC.ABS. Rename BASIC.BAF to BASIC.ABS. No longer will you have to type CONTRL 4,1 after loading BASIC. Neat!

RING-A-DING-DING

So you don't like the sound of the H9 bell? I didn't either, and found an extremely easy way to modify it's high pitch squeek to a low pitched beep. On the T.P.U. board there is a 22K resistor (the only one there). This resistor can be increased to 47K ohms. This mod' will increase the duration of the H9's bell. Now, to lower it's pitch, all you have to do is unsolder the white/red stripped wire on the 300/preset slide switch and solder the free end to the back of the I/O board IC601 pin 8. Happy beep-beep.

MORE SOFTWARE

Volume II, the latest release of HUG software is now available as announced in the new updated catalog which was recently sent to all members. It consists of two cassettes and the documentation and source listings. 885-1012 is a cassette with about 30 basic programs all written with version 10.02 (H8-13) 885-1014 is a cassette containing about 7 assembly language programs in Ted 8 source code. They must be assembled. 885-1013 is the necessary documentation to support these programs and the source listings. It is strongly recommended that you order this along with the tape since some of the programs require written instructions. 885-4: a HUG binder to hold the new release and; and 266-945: a cassette holder for two cassettes that is punched so your cassette masters can be kept in the binder for safe and handy storage.

HUG Binder is $4.00

Cassette Holder is $2.45

MODEM

To be announced in the next Heath Catalog is a neat little MODEM (WH-13). It is an answer/originate model which allows you to communicate with other computer owners via telephone. All you need is a terminal and one of these modems and you dial up any of the computer bulletin boards around the nation. Here is a list of known systems in operation at this time.

—BULLETIN BOARDS—
Chicago — 312-528-7141
Atlanta — 404-458-4886
New England — 617-963-8310
San Diego — 714-565-0961
Santa Clara — 408-246-2805
D.C. — 703-281-2125

Soon we hope to have our own system up and running and we would very much like to hear your thoughts on what the system should include. The modems are available now in your local Heath Electronic Center or by ordering from the Factory. The price is around $200 in the states. To save you time (and money) and allow more users on the system, here is a brief printout (next page) of what to expect when you call one of the systems. (they are very much alike.)

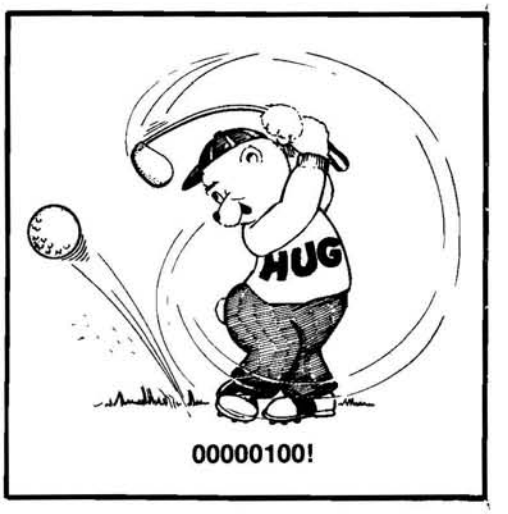
H14 Heat problem. . . What heat problem???

An independent newsletter recently published a piece suggesting that the H14 was delayed while engineering tried to figure out why the H14 print head would heat up and shut down the printer. Rubbish! The H14, under normal operation has an average initial through-put of 45 characters per second. The entire listings for Volume II were printed on an H14 without ever stopping. Yes, the head is protected with a sensor that will slow down or shut off the printer when the head temperature reaches a potentially dangerous level. This would occur when the printer was printing very dense text at 132 or possibly 96 characters per line. Under these conditions, the average through-put would be reduced to about 30 characters per second or about 300 baud. This is the way it was designed. I've heard of a user installing a blower which indicates an increase through-put by a bonus of 20%. Incidentally, life tests were terminated on the H14 after 50 million characters at 132 characters per line which means each head solenoid hit the paper about 150 million times.

Yes, the printer shut down during these tests which rendered an average through-put of 25 characters per second. Now that we have many local users' groups with regular newsletters, we often see misleading or totally wrong information published. It would be in everyones best interest if you would pick up the phone and contact Heath for the straight stuff before going to press.

```
    *** WELCOME TO WARD AND RANDY'S ***
*** COMPUTERIZED BULLETIN BOARD SYSTEM ***
       (IN OPERATION SINCE 2/16/78)

CONTROL CHARACTERS ACCEPTED BY THIS SYSTEM:

DEL/BS/_ BACKSPACE AND ECHO DELETED CHARS
CTL-C    CANCEL CURRENT PRINTING
CTL-K    FUNCTION ABORT
CTL-N    START SENDING 5 NULLS
CTL-R    RETYPES CURRENT INPUT LINE
CTL-S    SUSPENDS OUTPUT UNTIL NEXT CHAR RECEIVED
CTL-U    CANCELS CURRENT INPUT LINE

IF YOU GET STUCK, TRY:  CTL-K THEN C/R      UNTIL YOU
BAIL OUT BACK TO THE MAIN MENU.

WHENEVER WE REFER TO "C/R", WE MEAN YOUR "RETURN" KEY!!


Y/N: IS THIS YOUR FIRST TIME ON THE SYSTEM?Y
LOWER CASE CHARACTERS, Y/N: CAN YOUR TERMINAL HANDLE THEM?Y

What city/state are you calling from?St. Joseph mi.
File last updated 11/1/78

When you are asked a question, answer with Y for yes,
or N for no.
Are you using a video terminal (as opposed to a printer)?n

Want summary of system functions?y


The system has only three major functions:

    --> Entering messages
    --> Retrieving message summaries
    --> Retrieving messages


 Want to know purpose of this system?y
 It was conceived, financed, and is run by Ward Christensen
 and Randy Suess for message communication between computer
 hobbyists.  Typical message subjects include: 'something for
 sale', 'need help with something', 'club meeting notice',
 or just sharing ideas.  Feel free to leave a message on
 any hobbyist computer related subject.  Commercial 'ads' ARE
 permitted, providing they are hobbyist related.


 Want a quick summary of what to do your first time?y
 1.  S - Use the S function to retrieve message summaries
     (or Q for quick (subject only) summaries)
      *  Start with message 25 to 50 less than high # printed
         when you first signed on.
      *  Use control-S to stop/start output.
      *  Write down message numbers you want to see.
      *  Use control-K to kill the summary, return to menu
 2.  R - Use R function to retrieve messages of interest.
 3.  G - Use the G (Good bye) function when you are done.

 Remember to use the H (Help) function for details.

 NOTE:  When you receive the line:
 FUNCTION: (B,C,D,E,G.....) (OR ? IF NOT KNOWN)?
 --> Please reply ? the first time so you learn of all the
     functions available.
 What is your first name?Jim
 What is your last name?blake
 Hi, JIM BLAKE
 Y/N Did I get your name right?y
 (NOTE: This correcting is only done if this is your
 first time on the system.)

 Logging name to disk...
 Next msg # will be 419
 You are caller # 11915
```

```
FUNCTION: B,C,D,E,G,H,K,N,P,Q,R,S,T,W,X (OR ? IF NOT KNOWN)
??

----FUNCTIONS SUPPORTED----
B=Reprint bulletin
C=Case switch (upper/lower)
D=Duplex switch (echo/no echo)
E=Enter msg into system
G=Good bye (leave system)
H=Help with functions
K=Kill msg from system
N=Nulls: Set 0 to 9 as req'd
P=Prompt switch (Bell on/off)
Q=Quick summary (Msg #, subject)
R=Retrieve msgs by #
S=Summarize msgs
T=Print date, time
W=Reprint welcome
X=Expert user mode
FUNCTION: B,C,D,E,G,H,K,N,P,Q,R,S,T,W,X (OR ? IF NOT KNOWN)
?s
```



EOF

H19 SNEAK PREVIEW

The much rumored H19 Video Terminal is no longer a rumor... it should ship next month and will be described in the June catalog. For now, here are some of the more exciting features.

To begin with, it is a professional 24 line terminal with a 25th line to indicate status. Plus many other features not found on any other terminal. The H19/WH19 has a professional keyboard (similar to the DECwriter) numeric keypad and 8 user defined keys. The (P4) CRT displays 1920 characters 80 × 24, upper case in a 5 × 7 dot matrix and lower case in 5 × 9 matrix with descenders. In addition there are 33 special graphic characters that can be arranged and grouped to form many more symbols and special effects including reverse video by character.

Special local and software controllable excape sequences allow you to select and use thirty two special functions. . .automatic line or page scroll. . .cursor up, down, left, right and home plus direct cursor addressing. You can also insert and delete characters and lines and modify baud rates, all from the keyboard.

And here is a quick description of some of the other features. Also, you may want to copy the sub-routines at the end of this article so they can be called at will from any program thus saving the need to reinvent the wheel each time. We are very interested in the many ingenious new techniques that I'm sure you will come up with to show off the H19! :JB:

SPECIAL MODES

INSERT CHARACTER MODE

This mode lets you insert characters or words into the text already displayed on the screen. You must type ESC@ to enter the Insert Character Mode. You can then use the cursor controls to place the cursor at the point where you want to insert characters. As you type the desired characters, any existing text directly above and to the right of the cursor is shifted to the right. This feature lets you add letters or words to existing text without having to retype the whole text. When you finish inserting characters, type ESC O to exit the Insert Character Mode. You can also use the IC (Insert Character) key to enter and exit the Insert Character Mode.

DELETE CHARACTER

Type ESC N to delete the character directly above the cursor and shift any existing text on the right side of the cursor to the left one character position. You can also delete a character by typing the DC (Delete Character) key.

INSERT LINE

Type ESC L to add a new line of text before (above) the line the cursor is presently in. Make sure you position the cursor on the line **below** the line you want to add before you type ESC L. The cursor then moves to the start of the added line. You can also insert a line by typing the IL (Insert Line) key.

DELETE LINE

Type ESC M to delete all of the text from the line the cursor is presently in. You can also delete a line by typing the DL (Delete Line) key.

ERASE PAGE

Type ESC E to erase all text and fill the screen with spaces. The cursor is placed in the home position. You can also erase the page by typing SHIFT ERASE.

ERASE TO END OF PAGE

Type ESC J to erase all text from the present cursor position to the end of the page (screen). You can also erase to end of page by typing ERASE.

ERASE TO END OF LINE

Type ESC K to erase all text from the present cursor position to the end of the line the cursor is in.

GRAPHICS MODE

The Graphics Mode lets you display 33 special symbols. Type ESC F to enter the Graphics Mode. Then type any of the 26 lower case keys or the seven other symbol keys that correspond to the graphic symbols. Type ESC G to exit the Graphics Mode. You can place the Terminal in the Reverse Video Mode while it is in the Graphics Mode to increase the number of graphic symbols.

DIRECT CURSOR ADDRESSING

Direct cursor addressing allows the computer to control the position of the cursor on the screen. The top line is designated as octal 40 (32 decimal). The second line is 41 and so forth. The left column is also designated as octal 40. The number 40 is used because it is the smallest value of the printing characters. All values less than 40 are control codes.

The form of the direct cursor addressing escape code is:

ESC Y line# column#

Since the lines are normally thought to be numbered from 0 to 23 (from top to bottom) and the columns from 0 to 79 (from left to right), you must add 40 octal (32 decimal) to obtain the proper line # and column #.

REVERSE LINE FEED

Normally, a line feed moves the cursor down one line toward the bottom of the screen. A reverse line feed (ESCI) causes the cursor to move upward one line. If the cursor is at the top line it will remain there however, any text on the screen will be scrolled downward one line.

:JB:

EOF

```
5000 REM ERASE PAGE
5100 PRINT CHR$(27);CHR$(69);
5200 RETURN
5300 \
5400 REM ERASE BOTTOM LINE
5500 PRINT CHR$(27);CHR$(89);CHR$(53);CHR$(33);CHR$(27);CHR$(75)
5600 RETURN
5700 \
5800 REM ENTER REVERSE VIDEO MODE
5900 PRINT CHR$(27);CHR$(112);
6000 RETURN
6100 \
6200 REM EXIT REVERSE VIDEO MODE
6300 PRINT CHR$(27);CHR$(113);
6400 RETURN
6500 \
6600 REM ENTER GRAPHICS MODE
6700 PRINT CHR$(27);CHR$(70);
6800 RETURN
6900 \
7000 REM EXIT GRAPHICS MODE
7100 PRINT CHR$(27);CHR$(71);
7200 RETURN
7300 \
7400 REM MOVE CURSOR TO BOTTOM LINE
7500 PRINT CHR$(27);CHR$(89);CHR$(89);CHR$(33);
7600 RETURN
7800 \
7900 REM PRINT A$ ON BOTTOM LINE
8000 PRINT CHR$(27);CHR$(89);CHR$(53);CHR$(43);TAB(10);A$;
8100 RETURN
```


MANAGEMENT TRAINEE

Heathkit Electronic Centers, a Schlumberger Products Corp. Unit presently has career opportunities in the retail store program.

A retail and computer background is required. Excellent starting salary and benefits package. Send resume to:

PERSONNEL DEPARTMENT
SCHLUMBERGER PRODUCTS CORPORATION
P.O. BOX 167
ST. JOSEPH, MI 49085

An equal opportunity employer M/F

ED.HUG

In the last issue we mentioned we have a new editor for the H8/H17 system. However, there was not sufficient time to explain some of the features. Here is a list of commands and a few examples of their use. This editor is supplied on diskette as HUG P/N 885-1022 and includes full documentation and source code and HDOS common deck. Price is $15.00 plus $1.50 handling and postage.

INVOKING THE EDITOR

The editor is invoked with the following command (assume TEST.ASM is the file to be edited):

```
>ED TEST.ASM
```

This results in the editor being invoked and TEST.ASM opened for input; it also results in the creation of an output file, TEST,TMP, which is opened for write. Note that the file name is specified on the same line as the editor command, and cannot be specified after the editor is entered. If the file to be edited is on the SY1: drive, the format is: >ED SY1:TEST.ASM if the file is on the SY0: drive, but the output is to be placed on the SY1: drive (as would be necessary for files larger than 1/2 the size of the disk), the format is:

```
>ED TEST.ASM SY1:
```

The trailing colon on the output drive specification is optional. The above may be combined in any reasonable fashion, E.G.

```
>ED SY1: TEST.ASM SY0
```

Is valid, with input on SY1 and output on SY0.

COMMAND LINE FORMAT

The editor prompts for single letter upper or lower case commands with a '–' (minus sign). Any number of commands may be put on a command line, one after the other without any type of delimiter. Command lines are terminated (and executed) by typing two escapes (1B hex) in a row; escapes are echoed to the console as '$' (dollar signs); a single escape is used to terminate (delimit) strings in those commands requiring strings (search, replace and insert). In general, commands may be preceeded by a numeric repetition factor, which in some cases may be either positive or negative. The special symbol '#' (shift three) is interpreted as infinity (actually FFFF hex) for use when one wants to repeat a command until end of file or some similar automatic termination event. The following are examples of valid command lines (the commands used will be discussed below):

```
–B23T23L$$
```

This executes the B command, then executes the T command 23 times, then executes the L command 23 times.

```
–#A$$
```

```
–#AB23T$$
```

Bring in all the text which will fit, go to the beginning of the text and type out the first 23 lines (a full CRT screen, since the 24th line is occupied by the prompt for the next command line).

```
–0LT$$
```

Go to the beginning of the current line and type it out. In most cases the pointer will have already been at the beginning of the line, but this will move it there in any case.

–LT$$ move down 1 line and type it. Note: as a convienence borrowed from CP/M, a null command line (just a carriage return) is equivalent to an LT command. SKIP2

A string of commands may be executed repeatedly as a single command by enclosing them in <>'s. Consider the following examples:

```
–B5000<FJPO$0LTL>$$
```

This example goes to the beginning of the file and then finds the first 5000 occurrences of the string JPO, and in each case goes to the beginning of the line containing the string, types the line, and goes to the next line. (Quiz time: Why would the above command not work if the final L command was left out?) Another example:

The P (page) command is the same as typing 23L23T — it "pages" a 24 line CRT screen.

NOTE: The (P) page command can be changed at an EQU statement at the very beginning of the source code. The label is 'LPS'. This release is set for 12 lines.

COMMAND SUMMARY

COMMAND	FUNCTION
A	Append test into buffer from disk
B	Move pointer to beg. or end of text
C	Pointer movement by characters
D	Deletion of characters
E	End the edit
F	String search
I	Insert
K	Kill (delete) lines
L	Move pointer by lines
M	Displays text and buffer size
N	Auto string search
O	Return to original file
P	CRT screen page
Q	Quit the edit
S	String search and change
T	Type lines
U	Displays text and buffer size
W	Write lines to disk from buffer
Z	Pointer to end of file

EOF :JB:

SPECIAL H8/H17 SOFTWARE RELEASES!

VOLUME II

Tape II (BASIC Programs)

TAPE II (Assembly Language Programs)

A new release of HUG cassette software is ready — Volume II (885-1013) is the actual listings of all the programs and documentation for this new release. Over 30 very useful programs are included. ($12.00)

There are two cassettes that accompany this release. One is all BASIC Programs (885-1012); most are utility programs, (no games). The other cassette contains the complete Assembly Language Source Code for 7 general purpose programs (885-1014). The cassettes are $9.00 each. The details of this release are further described in your new Software Catalog.

Volume III which is all financial and amateur applications programs should be ready by the first of June.

W6LLO RTTY COMMUNICATIONS PROCESSOR

By: Howard Nurse — HUG P/N 885-1023

The W6LLO RTTY Communications Processor converts your H8/H17 system into an advanced amatuer radio teletype control center. When connected to a radio transmitter/receiver through a modem your computer will come alive with digital voices from around the world.

This package performs the function of the communications operating procedures required by the FCC, or in common use through convention. Some of these duties include conversion of data back and forth from BAUDOT to ASCII; CW Id; timekeeping; text editing; data buffering; operating and system status indication; and remote control functions.

Hardware requirements are an H8 with 24K, H17, H9 terminal, H8-2 with external UART, (schematic included) and RTTY modem such as RM-300 (as described in September 78 Ham Radio Magazine).

Complete written documentation and users guide included with the diskette and full source code. Available now — order HUG P/N 885-1023 price — $22.00 + 10% postage and handling.

Smaller type cassette version available in June.

DDDVD.ASM/BPDVD.ASM

DEVICE DRIVER — Source code and HDOS common decks which allows the sophisticated assembly language programmer to adapt this program to communicate with various periphial devices — (BPDVD.ASM is a BAUDOT device driver). Minimum documentation included and sans Heath technical support — P/N 885-1019 — price $10.

DISK SOFTWARE (H8/H17)

REN.ABS *By: Mark Duttwiler*

Assembly language program that renumbers basic programs.

DSKCAS.ABS *By: David Lovett*

Allows the HDOS user to transfer files to and from cassette.

PFS.BAS *By: Jack Coursey*

Complete personal finance program

FINANCE.BAS *By: R. Reale*

Complete home finance management program

INVENTORY *By: A. T. Snucker*

Suitable inventory program for small businesses.

Order HUG P/N 885-1024 — price $18. Each program is self documentated.

Vectored from Page 19

```
01260 W9=SGN(VAL(W8$))
01262 W7=VAL(RIGHT$(W8$,5)):W8=ABS(VAL(LEFT$(W8$,LEN(W8$)-5)))
01264 W5=VAL(RIGHT$(W9$,5)):W6=ABS(VAL(LEFT$(W9$,LEN(W9$)-5)))
01266 W4=W8-W6:IF W5>W7 THEN W7=W7+10:W4=W4-1
01268 X0=W7-W5:GOSUB 1105:W7$=RIGHT$(Y$,LEN(Y$)-1)
01270 W4=W4*W9
01272 IF W4=0 THEN W6$=STR$(W9):W6$=LEFT$(W6$,LEN(W6$)-2):GOTO 1276
01274 W6$=STR$(W4):W6$=LEFT$(W6$,LEN(W6$)-1)
01276 Y$=W6$+W7$
01278 RETURN
01280 W9=SGN(VAL(W9$))
01282 W7=VAL(RIGHT$(W8$,5)):W8=ABS(VAL(LEFT$(W8$,LEN(W8$)-5)))
01284 W5=VAL(RIGHT$(W9$,5)):W6=ABS(VAL(LEFT$(W9$,LEN(W9$)-5)))
01286 W4=W6-W8:IF W7>W5 THEN W5=W5+10:W4=W4-1
01288 X0=W5-W7:GOSUB 1105:W7$=RIGHT$(Y$,LEN(Y$)-1)
01290 GOTO 1270
01297 REM VARIABLES USED:W9$,W8$,W7$,W6$,W9,W8,W7,W6,W5,W4,X0,Y$
01299 REM  ************************************************************
01300 REM ROUTINE TO SUBTRACT LARGE-NUMBER STRINGS
01301 REM ENTRY: W8$, W9$        EXIT: TO LINE 1210
01302 REM STRING NUMBERS MUST SATISFY CONDITIONS FOR ROUTINE 1200
01303 REM AS THIS SUBROUTINE CHANGES THE SIGN OF THE SUBTRAHEND, W9$,
01304 REM AND THEN EXITS TO SUBROUTINE 1200 FOR THE ADDITION.
01305 REM YOUR PROGRAM MUST SET W8$=STRING NUMBER FROM WHICH YOU WANT
01306 REM TO SUBTRACT W9$ WHICH YOUR PROGRAM MUST SET EQUAL TO YOUR
01307 REM SUBTRAHEND.
01310 IF LEFT$(W9$,1)=" " THEN W9$=RIGHT$(W9$,LEN(W9$)-1):GOTO 1310
01312 IF LEFT$(W9$,1)="-" THEN W9$=RIGHT$(W9$,LEN(W9$)-1):GOTO 1210
01314 W9$="-"+W9$:GOTO 1210
01399 REM  ************************************************************
```

EOF

HUG MEMBERSHIP RENEWAL PROGRAM UNDERWAY

As you know, we have been using the form below to solicate membership renewals. This method saves HUG some bucks and has been working extremely well. However, if you do not renew within about thirty days of your expiration date, you will receive a general reminder in the form of an E Z mailer. Since there is some interim period between your sending in your renewal and it being posted, you may receive the notice even if you have renewed — of course, ignore the notice in that event — very shortly, all renewals will receive a new membership card. This will be the only acknowledgement. Also, anyone with a program in the library will automatically be renewed and sent a new ID card.

CUT ALONG THIS LINE

HUG MEMBERSHIP RENEWAL FORM

You can determine your expiration date by examining the last six digits of your ID number – example: 780202 indicates your membership began 02/02/78 and expires one year from then.

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 APPROPIATE BOX AND RETURN TO HUG

			NEW MEMBERSHIP? FEE IS:
RENEWAL RATES			
US DOMESTIC	$11 ☐		$14 ☐
CANADA	$13 ☐	US FUNDS	$16 ☐
INTERNAT'L*	$18 ☐	US FUNDS	$24 ☐

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

THE BACK PAGE —

CONTEST #4 WINNER

Congratulations to Dave Lovett of Wichita, Kansas as winner of contest #4! Dave's entry was an assembly language program that allows the HDOS user to transfer files to and from cassette, thus providing for long term, inexpensive off-line storage.

Dave's routine, with complete source code is included in the first release of disk software described on page 34.

Contest #6

The prize for the best H8 program submitted to the HUG software Library before June 18 is a piece of paper worth $250! Submit full printed listing and documentation. Submit documentation and a cassette or diskette. The program will be judged on the overall usefulness to other users, completeness of work, ease of use, creativity and quality of documentation. The winner will receive a gift certificate worth $250 which may be applied toward any purchase through the mail order catalog or through any Heath Electronic Center in the US or Canada.

H11 owners, same for you! A $250 gift certificate is yours for your program written in any language which in the opinion of the judges will help increase the usefullness of the H11 equipment to other users. If written in paper tape, submit a copy of your program on paper tape with adequate documentation which could allow a disk user to possibly understand the functions of your program so he may convert it if possible. If written under HT11, submit a diskette (whichwill be returned) with full documentation.

It is perfectly acceptable to include documentation, etc. on the diskette or cassette if you do not have a printer or if you're a rotten typist like me.

Deadline for all entries is June 18, 1979. Clearly mark all materials with your name, program, title and contest #6.

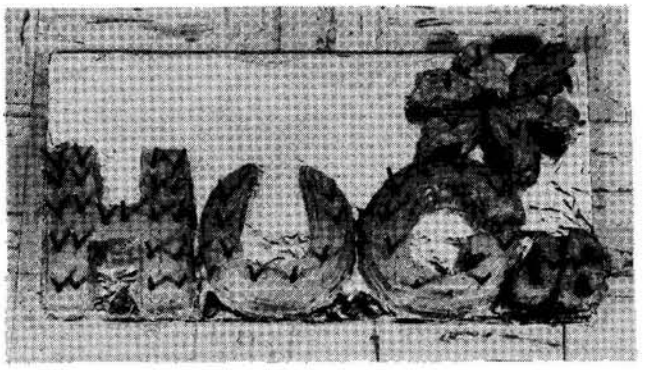
ET-3400 USERS — YOUR TURN IS COMING!

BACK ISSUES AVAILABLE

Back issues of REMark, beginning with issue one are now available. You may order them on the green order form at $2.50 each.

Issue one	885-2001
Issue two	885-2002
Issue three	885-2003
Issue four	885-2004
Issue five	885-2005

Since many local users groups have formed in the past few months, would you drop a line and tell us the details of your organization so we may include an updated list in the next issue?



Your HUG secretary is also a good cook — your Editor reached 29_{16}.

Heath Users' Group
Hilltop Road
St. Joseph MI 49085

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

POSTMASTER: If undeliverable, please do not return.

885-2006