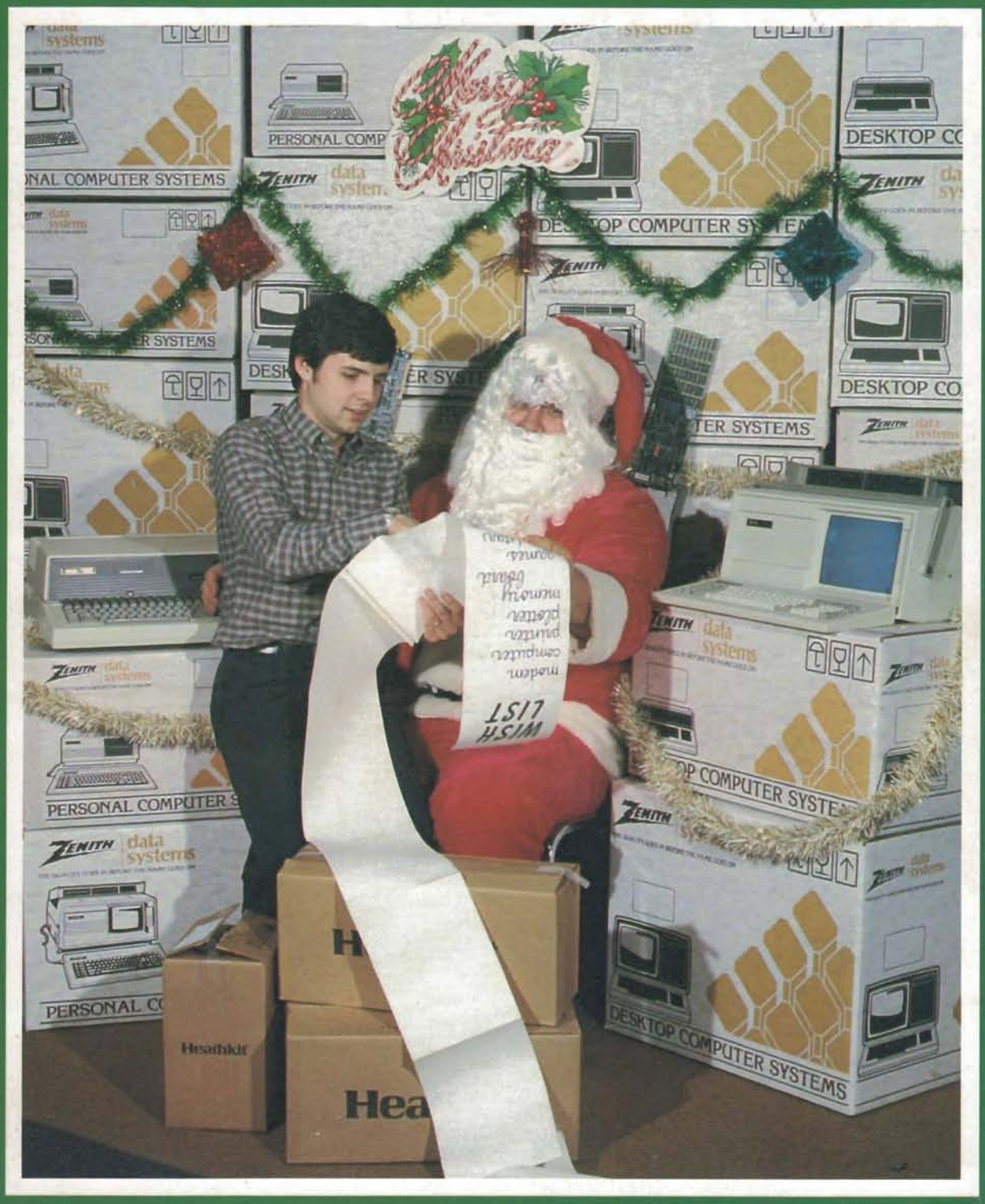
$2.50

REMark®

Volume 5, Issue 12 • December 1984

P/N 885-2059



Official magazine for users of HEATH ZENITH computer equipment.

YOU'RE IN GOOD COMPANY WITH AN H-1000

Amdahl, Nestles, Bell Labs, Jet Propulsion Laboratories, the U.S. Army and Navy, Allied, Bendix, CDR – these are just a few of the many satisfied H-1000 users. They know that the H-1000 gives them performance and compatibility unmatched by the giants. And many distinguished hobbyists have discovered that the H-1000 is available at a very attractive price – typically half the cost of a new IBM-compatible. Just look at the benefits:

- emulates IBM PC with monochrome board; runs most PC software at 4 times the speed
- 100% H89 compatible; runs your programs twice as fast with NO changes
- on-board memory from 128K to a full megabyte; holds twice the data and programs of any comparable PC!
- built-in Ghost disk for all operating systems; adds an additional RAM disk for instantaneous disk accesses.
- from $995 to upgrade your H89, or $1995 for a complete computer.

A STATE-OF-THE-ART UPGRADE

TMSI has taken the best features of Heath's 8-bit H89 and combined them with the best of the 16-bit PC's to give you the best of both worlds . . . and the ONLY package that builds on your current investment.

The 8-bit side gives you access to the world of low-cost 8-bit software for the CP/M and HDOS operating systems. The H-1000 runs all Heath/Zenith software and hardware for the H89 family without change. And with its 4MHz Z80, expanded memory, and ghost disk, your existing programs perform like never before.

The 16-bit side of the H-1000 features an 8086 running at over 8MHz, offering you the fastest screen response and program execution of any PC compatible on the market today. The standard keyboard and screen emulate an IBM PC with monochrome display; accessory color and graphics boards can extend the emulation even further. The "Ghost disk" is an added bonus; entire disks can be loaded into memory, freeing your "real" drives for another disk. You get instantaneous access to your data that cuts out all that waiting while your computer talks to your disk drive.

FULL SERVICE BACKUP DIRECT FROM THE FACTORY

The H-1000 comes fully assembled and tested, with Heath's "we won't let you fail" quality manuals. A full 90-day warranty covers each unit, with an optional service contract available. And we work closely with our customers for special applications assistance. Get the benefits of high-performance computing today! Call or write:

TMSI, Dept. RN
366 Cloverdale
Ann Arbor, MI 48105
(313) 994-0784

TMSI

Heath Users' Group

Staff

Manager ..... Bob Ellerton
(616) 982-3867

Software Engineer ..... Pat Swayne
(616) 982-3463

Bulletin Board and
Software Developer ..... Jim Buszkiewicz
(616) 982-3463

Software Coordinator ..... Nancy Strunk
(616) 982-3838

Secretary ..... Margaret Bacon
(616) 982-3463

REMark

Editor ..... Walt Gillespie
(616) 982-3789

Editorial and Advertising
Assistant ..... Lori Latham
(616) 982-3794

Printers ..... Imperial Printing
St. Joseph, MI

	U.S. Domestic	Canada & Mexico	International
Initial	$20	$22*	$30*
Renewal	$17	$19*	$24*

*U.S. Funds.

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

Limited back issues are available at $2.50, plus 10% shipping and handling. Check HUG Product List for availability of bound volumes of past issues. Requests for magazines mailed to foreign countries should specify mailing method and appropriate added cost.

Send Payment to: Heath/Zenith Users' Group
Hilltop Road
St. Joseph, MI 49085

(616) 982-3463

REMark®

Volume 5, Issue 12 • December 1984

on the stack

ON THE COVER: A typical HUGgie (Scott Bennett, Heath Graphic Designer) asks our great HUG Santa (Jim Buszkiewicz, HUG Software Developer) for a new computer system and accessories.

Inside Information

Heath Zenith Computers & Electronics

We're Changing Our Store Name

Heathkit Electronic Centers are becoming Heath/Zenith Computers and Electronics stores. Some have already...the rest will soon. You've seen our growing commitment to a wide range of computers and accessories. Now we're ready to become one of the most important computer retailers in North America.

You're Among The First To Know

HUGgies have been key to building success into the Heath/Zenith line of computer products. And you can be assured of our continued commitment to you. Our stores will now provide you with even more innovative computer technology in both assembled and kit form.

The Celebration Includes A Special Deal

We're excited about our new store name. And we want you to share in the excitement by taking advantage of the offer below. It's one of the best deals we've ever made...so hurry before it's over!

H-89 System

VERSATILE H-89 SYSTEM INCLUDES:

• HS-89 all-in-one computer with 48K RAM, internal disk drive and choice of green or white screen • Z-87-89 assembled dual disk drive • HOS-8917-2 CP/M operating system • HMS-817-2 FORTRAN compiler for scientific and engineering programs.

In addition, your purchase of the H-89 System entitles you to 50% off* the list price of the following products:

• SuperCalc HSC-817-1 or HSC-837-1 • SuperSort HMP-817-1 or HMP-837-1 • WordStar HMP-817-2 or HMP-837-2 • C BASIC HCM-817-1 or HCM-837-1 • MS FORTRAN HMS-837-2 • MS COBOL HMS-837-3 • MS BASIC compiler HMS-837-4 • PASCAL HOS-8937-3 • Z-89-11 multi-mode interface.

OFFERS END JANUARY 31, 1985.

Come and see it all at your local Heathkit Electronic Center... or your Heath/Zenith Computers and Electronics store!

*Prices and discounts include HUG membership discount. Offer subject to product availability.

For HUGgies only **$1259*** $1,623.97 current advertised price.

Your TOTAL SERVICE computer center

• Service • Support • Software • Accessories • User Training • Competitive Prices

VEC-861

Heath Zenith®

*Units of Ventechnology Electronics Corporation

Computers and Electronics

BUGGIN' HUG

Software Coverage Only?

Dear Walter,

I have been an owner of an H-100 computer and a subscriber to your magazine for almost a year now. I have found your magazine both useful and informative. However, it seems that your coverage is exclusively oriented toward software. I believe that a true users' group should address hardware aspects, as well as software aspects of the computer. For example, I would be interested in seeing an article which shows how to add a hard disk drive to the H-100.

Sincerely,

Ken Goto
1306 Bottle Brush Lane
San Jose, CA 95118

Editor's Note: Your comment is well taken, however, we print what we receive from HUG members. Little or no articles have come in on hardware modifications to the H/Z-100. Perhaps you would care to put something together for us.

This Is Ridiculous!

Dear HUG:

This is ridiculous! Reference my letters dated August 15 (Nov. Buggin' HUG) and August 16 (Dec. Buggin' HUG), concerning the CROSSREF program listing in the August issue of REMark, I discovered that the first variable reference in the target program will not appear in the printout.

The following replacement lines for the listing I enclosed with my 16 August '84 letter (the listing that begins with line 64999) will not only correct that problem but will also eliminate the need for the program to show a hyphen as the first entry in the printout:

```
65008 IF CH>3 THEN LV=1:PRINT:INPUT
      "Enter the Var/Line # to search for; ";@
      V$(1):V$(1)=V$(1)+" -":J=1:GOTO 65010
65009 V$(0)=V$(0)+" -":J=0
65032 F=0:FOR J=0TO LV:GOSUB 65012:NEXT:IF F=1
      THEN RETURN ELSE @ V$(J)=AV$+" -":LV=LV+1:GOTO 65014
```

You will note that line 65008 is the same as that part of the author's line 65015 after the ELSE, except for his unnecessary V$(2)="ZZZ". When searching for only one variable or line # reference (CH>3), it is necessary to identify V$(1) at that point and to add the " -" to separate the variable or line # from the list of line numbers where it appears.

However, the hyphen must also be added to the first item in the array V$(n) when all variables/line # references are being searched for (CH = 1, 2 or 3). The reason is that line 65012 (line 65150 in the author's listing) would result in a syntax error if the first item in the array had no hyphen. (X would equal -2.)

The apparent solution was to place the V$(1)=V$(1)+" -" and J=1 statements on a separate line that would apply regardless of the value of CH. However, that not only caused V$(1) to always be nothing but a hyphen, a useless first entry in the sorted printout, but also caused the problem of losing the first variable reference at line 65032 because J and LV were not tracking together properly.

Therefore, by leaving line 65008 as it was, adding line 65009 [i.e. starting the counter J at 0, instead of 1 and adding the hyphen to V$(0) instead of V$(1)], as well as starting the loop in line 65032 with 0 instead of 1, two birds are killed with one stone. The final sorted list is not only accurate and complete, but also there is no hyphen as the first item in the printout.

Sincerely,

C.F. Mowery, Jr.
406 Van Reed Manor Drive
Brandon, FL 33511

Microsoft MACRO-80

Dear HUG,

Several months ago, I purchased the Microsoft MACRO-80 assembler package from Heath. It is, of course, much different than the Heath ASM assembler. There are many different assembler directives and the method of operation is more complex. And unfortunately, the reference manual does not explain fully how to use the full power of this program, nor do there appear to be any books available to assist the novice assembly language programmer.

It would be extremely beneficial if someone knowledgeable in the use of the Microsoft MACRO-80 assembler package would write a tutorial for publication in REMark Magazine, perhaps as part of the current series on assembly language programming.

If such articles are not feasible for whatever reason, can you provide me with any information concerning source materials for learning how to use the MACRO-80 assembler. Any assistance you may be able to provide would be greatly appreciated.

Sincerely,

James R. Scheip
6487 Silver Ridge Circle
Alexandria, VA 22310

An Annoying Problem With MS-DOS V2

Dear HUG:

As you know, the PRINT utility in MS-DOS V2 works differently than the PRINT utility in ZDOS with regards to skipping over form perforations of fan-fold paper. This is tied into the number of lines per page, or page length as it is sometimes referred to.

In ZDOS, the PRINT utility provided a default page length if one was not supplied with the "P" option. However, that is not the case with the PRINT utility under MS-DOS V2. There is no default page length. Unless there are page control characters embedded into the listable file, as is the case with the Macro Assembler and Compilers, the file is printed right over the form perforations. This can be very annoying if the paper is not positioned quite right and the printing occurs right over the form perforations.

The solution to this is to use the page length parameter when calling the PRINT utility. However, this brings up another annoying problem. Once given a page length parameter, the PRINT utility uses this

from file to file without ever resetting the page length counter when finished with a file unless told to do so explicitly with the "F" parameter (form feed after print).

Let me give you an example. Suppose your 1st file was small, say 40 lines and you called the PRINT utility with a page length of 50 lines per page. When the file is finished printing, the page counter has been decremented down to 10. You externally issue a page eject (form feed) to get your listing. A short while later, you decide to print another file which is say 200 lines long. You call PRINT with no form feed or page length parameters. The PRINT utility will continue to decrement the original page length down to 0, then issue a page eject, and finally reset the page length counter. What happened is that the first page contains just 10 lines, then 50 lines per page till the last page.

To get around this annoying problem, I always specify the /F and /Pn parameters, form feed and page length parameters respectively, for every file that I print. This works for me. Did anyone else experience this situation?

Richard L. Mueller, Ph.D. (Rich)
11890-65th Avenue N.
Maple Grove, MN 55369

Problems Using RANDOMIZE TIME/DATE In ZBASIC

Dear HUG,

In response to David Harvey's inquiry (Buggin' HUG June '84) concerning the use of the ZBASIC RANDOMIZE function, I submit my solution to the problem of automatically reseeding the random number generator. The following was printed in the FT HUG Newsletter.

I started using the RANDOMIZE statement the first day I owned my computer. By looking at a ZBASIC Demo program in volume 1, page 8.30 of the documentation, I noticed the use of RANDOMIZE TIME/DATE (TIME being the numerical value of TIME$ and DATE being the numerical value of DATE$) to automatically "seed" the random number generator at the beginning of a program. This must be the way to go, I thought. Wrong!

Try this program:

```
10 TIME$="12:00:00"
20 DATE$="01/01/83"
30 RANDOMIZE TIME/DATE
```

It produces the error, "overflow in 30."

Overflow will occur any time TIME/DATE is greater that 32767 or less than -32768. On January 1st, this will happen after about 9:00 a.m., on January 2nd, it won't happen until about 6:00 p.m., and on or after January 3rd, your OK, sort of.

Try this program:

```
10 TIME$="12:00:00"
20 DATE$="12/31/83"
30 RANDOMIZE TIME/DATE
40 PRINT RND
```

RUN the above program several times. RUN it a few more times. That's right! You keep getting the same number each time. What kind of randomness is this? It will take 365 seconds (about 6 minutes) before you can be sure of getting a new "seed" for your RANDOMIZE statement using TIME/DATE on the 31st of December. If you're playing a game, and you don't reRUN it for 6 minutes, everything is fine. Otherwise, you keep repeating the same game over and over.

The idea of RANDOMIZE TIME/DATE is great. Every TIME we run a program we get a different "seed" for the random number generator, and this "seed" is different for every DATE of the year. However, because of the overflow problem and the time delay problem, we need to modify it a bit.

A solution.

The variable TIME; 0 to 86399
for TIME$; 00:00:00 to 23:59:59

The variable DATE; 1 to 365
for DATE$; 01/01/?? to 12/31/??

First of all, the overflow problem can be overcome with some scale factors:

```
RANDOMIZE (TIME)*(65535)/(86399)-32768
```

will produce "seeds" from -32768 to +32767 as TIME goes from 0 to 86399.

```
RANDOMIZE (DATE)*(65535)/(365)-32768
```

will produce "seeds" from -32768 to +32767 as DATE goes from 0 to 365.

RANDOMIZE TIME/DATE is a nonlinear function of time. i.e., on January 1st, the "seed" can be updated every second if required, but on December 31st it takes 365 seconds. TIME and DATE need to be combined in a linear fashion.

```
RANDOMIZE
[(TIME)*(65535)/(86399)-32768]+
[(DATE)*(65535)/(365)-32768]/2
```

will produce "seeds" from -32768, for TIME=0, DATE=0, to +32767 for TIME=86399, DATE=365. "Reseed" time is only 2.64 seconds for any DATE.

The algebra can be reduced to:

```
RANDOMIZE .379*TIME+89.77*DATE-32768
```

Use this statement at the beginning of all your game programs to insure that you will indeed get a new and different game each and every time you RUN it.

Also, please note that the local club listing (June '84) for FT HUG was in error. The correct contact is:

FT HUG (Fort HUG)
c/o Charles McJilton
3317 Buckskin Trail
Laporte, CO 80535

Del Tapparo
2512 Bradbury Ct.
Ft. Collins, CO 80521

An INCONSISTENCY In MS-DOS V2 on Z100-PC Series

Dear HUG:

I'm not sure if the general Heath/Zenith user community is aware of an inconsistency in the way the TAB (ASCII code 09H) character is handled by MS-DOS V2 on the Z100-PC when outputting that character to the system console (CRT) using the different "DOS" function calls.

The TAB character functions properly when it is sent to the console using either the DOSF_CONOUT or the DOSF_OUTSTR function.

Vectored to 64 ☞

INTERROGATOR™

Drive Diagnostic Program for the ZENITH Z/H-100 Series

Diagnosing disc drive problems has just become a great deal easier.

Interrogator,™ Dysan's economical new drive diagnostics package, can run a drive through its pace in minutes. To tell you exactly what's wrong. Or right.

Just like that.

Test results come right up on the screen of the machine you're testing. Or you can print out hard copy records.

The Interrogator package includes a Digital Diagnostic Diskette,™ complete test instructions and software programmed to perform a whole battery of sophisticated tests.

Things like head radial alignment. Head positioner linearity. Head azimuth alignment. Index timing. Spindle speed. Read/write verification. And over a dozen others.

Interrogator is here now for the Zenith Z/H 100 Series. Testing both 5¼" and 8" models. Versions for other computer models are on the way.

Whether you're going for one drive or need to assure the compatibility of a company full of drives, Interrogator is the perfect tool.

And it's just the kind of thing you'd expect from Dysan. The leader in quality, innovative media and diagnostic tools.

Interrogator. What could be easier?

Just this. A toll free phone call to get more information. Or place an order. (800) 551-9000.

Dysan®

5201 Patrick Henry Drive
P.O. Box 58053
Santa Clara, CA 95050
(408) 988-3472

ANNOUNCING THE MODIFIER

A disk utility that modifies the CP/M BIOS to be able to read and write to a number of 5.25" CP/M disk types.

There is a growing need for the everyday user of computer systems to be able to take data files home from the office to continue to work on them. The computers at home and at work may both run a version of CP/M, but the disk structures may be incompatible. This is especially a problem in the 5.25" world. MODIFY 89 was designed to address this problem. MODIFY 89 makes the CP/M operating system access a specified 5.25" drive as one of the below disk types.
Disks placed in that drive that are of the specified type can be used as if they were one of the standard disk types accepted by the H8, H/Z89 or H/Z90 computers. Thus PIP, STAT, DIR and others will work for that disk also. The price for MODIFY 89 is $49.95.

MODIFY 89 is set for the following disk types:

- Access S.S. D.D.
- Cromemco S.S. D.D.
- DEC VT180 S.S. D.D.
- IBM PC/Zenith 100 (CP/M) S.S. D.D.
- IBM PC/Zenith 100 (CP/M) D.S. D.D.*
- Kaypro II S.S. D.D.
- Morrow Micro Decisions S.S. D.D.
- NEC PC-8001A S.S. D.D.
- Osborne S.S. S.D.
- Osborne S.S. D.D.
- Otrona D.S. D.D.*
- Superbrain Jr. S.S. D.D.
- TI Professional S.S. D.D.
- TRS-80 Model I (Omnikron CP/M)
- TRS-80 Model III (MM CP/M)
- Xerox 820 S.S. S.D.
- Xerox 820-II S.S. D.D.
- Standard (Tests for H/Z37 and C.D.R. Disk types)

* = Double sided 5.25" drive required

S.S. = single sided, D.S. = double sided, S.D. = single density, D.D. = double density

Limitations: MODIFY 89 is not a disk duplicate program. It is currently available for use with an H/Z89 or H/Z90 computer that has an FDC-880H double density 8" and 5.25" controller, using C.D.R.'s BIOS V.2.9 or with an H8 computer using the FDC-H8 by C.D.R. Systems, Inc.

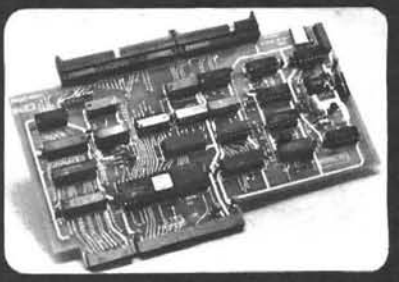
MODIFY 100 will soon be released for the Z100 line of computers at a price of $75.00.

Contact:

C.D.R. Systems, Inc.
7210 Clairemont Mesa Blvd., San Diego CA 92111
Telephone: (619) 560-1272
Or a C.D.R. Systems, Inc. dealer near you.

1 CONTROLLER

FOR 8" & 5.25" DRIVES



Now be able to run standard 8" Shugart compatible drives and 5.25" drives (including the H37 type) in double and single density, automatically with one controller.

Your hard sectored 5.25" disks can be reformatted and used as soft sectored double density disks. The FDC-880H operates with or without the Heath hard sectored controller.

PRICED AT $395

Includes controller board CP/M boot prom, I/O decoder prom, hardware/software manuals BIOS source listing.
HDOS driver now available for $50.00.

5-20 day delivery–pay by check, C.O.D., Visa, or M/C.

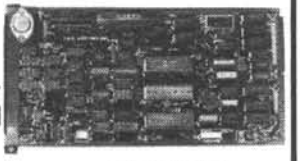
Contact:
C.D.R. Systems Inc.
7210 Clairemont Mesa Blvd.
San Diego, CA 92111
Tel. (619) 560-1272

Controlled Data Recording Systems Inc.

AVAILABLE NOW THE FDC-H8

DOUBLE DENSITY 8" AND 5.25" CONTROLLER FOR THE H8 COMPUTER

Has all of the capabilities of our popular FDC-880H controller, with the added features of;



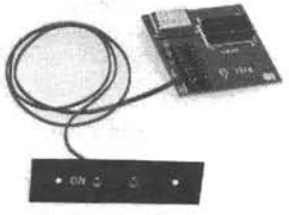
- Direct memory access (DMA) data transfer.
- Hard sectored controller (H17) incorporated on the board.
- Runs with the standard 8080 CPU card and with Z80 CPU upgrades.

PRICE $495

- Accesses both hard sectored disk formats and soft sectored disk formats through the same drives attached to the FDC-H8 without hardware additions.

Contact:
C.D.R. Systems Inc.
7210 Clairemont Mesa Blvd., San Diego CA 92111
Telephone: (619) 560-1272

THE ORIGINAL Z100 SPEED MODULE RUN YOUR Z100 PROGRAMS FASTER



The ZS100 runs the Z100 CPU 50% faster, (7.5 MHZ) in 8088 mode.

The ZS100 installs easily with no soldering.

The ZS100 is externally switchable between speed mode and normal.

The ZS100 improves the time performance of applications packages with no software modifications needed.

The ZS100 is for all Heath/Zenith 100, 110 and 120 series computers.

SPEND LESS TIME WAITING FOR YOUR COMPUTER

Contact your local Heath/Zenith dealer to buy this cost effective Z100 enhancement, or contact:

Controlled Data Recording Systems, Inc.
7210 Clairemont Mesa Blvd.
San Diego, Ca. 92111
Telephone (619) 560-1272

The completed unit. The door just to the left of the CRT slides open to expose the brightness control and keyboard cable storage area.

Heathkit Releases First Computer Kit After Seven Years!

. . . a review of the H-161

Jim Buszkiewicz
Software Developer

It was around August 1977 when Heath announced the first H-8 and H-11 computer kits. I wasn't with Heath at that time, but I still remember seeing the preliminary advertisements in Byte and Interface Age magazines. The ads went something like.. "Now build a computer the way you would want it, designed with the options you want...," or something like that. Both computers were totally disassembled except for the CPU boards. Not long afterwards the H-19 terminal and H-89 computer made their appearances. The H-89 was essentially an H-19 with a beefed up power supply and another pre-wired circuit board. Except for the video and power supply boards, most of the construction done on these units was mechanical in nature. Being a hardware consultant at the time, I found that most Heath computer customers had split opinions as to whether they wanted to do alot of construction, or very little. Having gone to several computer shows, I found that most new computer enthusiasts were attracted to the fact that they didn't have to do much soldering to complete the H-89. Veteran computer owners were quite upset when they found out that their soldering irons stayed cold through most of the build. Well, all good things must pass, and the H-89 made way for the H-100 systems. The low profile H-100 had exactly 2 solder connections. Even if you didn't know how to solder you could tie the wires to their respective connectors without any degradation in system performance. This was the straw that broke the you-know-what. The letters came pouring in. Most of them saying: "What happened to the old Heathkit we all knew and loved?." To these people I say: "Never fear, the H-161 is finally here!." Even though the CPU and Video boards are factory wired and tested, kit building addicts will soon tire of the three thousand three hundred and forty eight solder connections needing to be made to complete this kit.

Upon opening the very large (too big for UPS) shipping carton, the first thing I reached for was the Construction Manual. This manual has only 75 pages (less than the H-89A), and for those of you that haven't built a Heathkit for some time, that's parts list and construction steps only! If you recall, the H-8 had pictures of the circuit boards on each page with little arrows showing where each component went. Well, those pictures have all been moved to the Illustration Booklet, and schematic reference numbers are now used to indicate where each circuit board component should go.

Even though computer manuals are written to make the writer look intelligent and the reader stupid, the documentation included with this computer appears to be better than average. Along with the Assembly Manual and Illustration Booklet comes a booklet entitled, "Service Support Information." This book contains pinouts for all the IC's used in the kit, as well as component layout sheets, circuit board foil patterns, and a complete set of schematic diagrams. An Operations Manual is also included. This manual consists of the new mini-size three ring binder which holds a Users Manual, and the Operations Manual. The Operations Manual, if you recall, always had a circuit description section which gave all you hardware types an idea of where the electrons should and shouldn't be flowing. Well, that's changed here too. Now the information in the Operations Manual is geared toward configuring the system and supplying information a programmer might need to efficiently use the computer. This type of information always seemed to be lacking in earlier Heath computer manuals.

If you've ever built a Heath product, you usually have two choices when you begin assembly. You can either meticulously account for every part by checking it off of the parts list, or you can hope Heath didn't make a mistake and put all of the parts in one big pile until called for. If a part does come up missing, you can always claim you never got it in the first place. I, of course, chose the latter. This was probably a mistake. The memory board alone has over 200 parts to install on it. Remember Zymurgy's First Law of Evolving System Dynamics: "Once you open a can of worms, the only way to recan them is to use a larger can."

Before you actually start construction on the computer, you're given a short, but effective course in soldering. Included with this course is a separate Audible VOM kit and a practice circuit board onto which five IC sockets are soldered and then tested for opens and shorts with that VOM. I personally felt that I didn't need to take this course in soldering, having been born with a soldering iron in one hand.

Heath, having included eight of these soldering test circuit boards in my kit, didn't seem to share my confidence.

Having passed the soldering course part of the assembly manual, I then proceeded with the construction of the backplane. I'd recommend making sure that when you empty parts out of the paper bags, nothing gets caught under the paper flap at the bottom of the sack (like the tiny LEDs that go on the backplane). The backplane circuit board consists mainly of eight (8) sixty-two pin circuit board connectors, a few capacitors, resistors, and LEDs. One of the resistors, which was supposed to be 5 ohms at 10 watts, was not marked. Judging by its large physical size, I measured the value of what I thought to be the right resistor. Even though my Audible VOM indicated it was on the high side of 5 ohms, I installed it anyway.

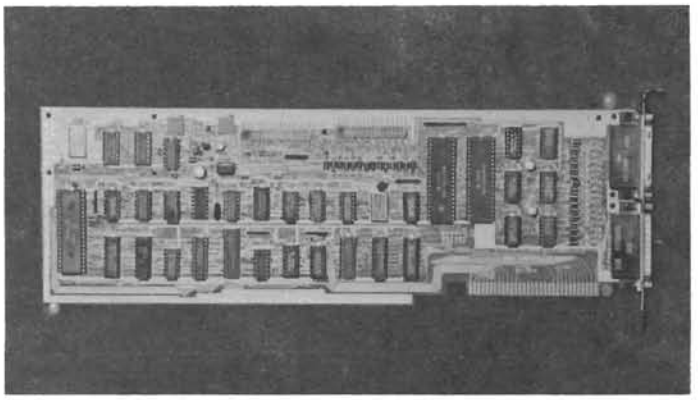
The next board you construct, the memory board, is a whopper. This board contains roughly 1370 solder connections, most of them IC sockets. Several hours are needed to complete this board, so take your time, and take several coffee breaks. The last step in constructing this board indicates that if you have purchased the Z-205-1 RAM Upgrade Accessory, install it at this time. I would advise waiting until the board is first tested in the machine. You may just have to remove all those extra chips to correct any problems that you may find. If you didn't buy extra chips for that board, you may want to purchase your extra memory from someplace other than Heath. I'm sure you'll find them to be cheaper. What you'll need for each bank, is nine (9) 4164's with an access time of at least 200 nano-seconds. This board will hold an additional three banks for a total board capacity of 320k. In addition to the memory on this board, is a standard parallel port. The connector used on this port is a standard 25 pin 'D' style variety.

Although not as threatening, but just as time consuming, the disk controller board is assembled next. One helpful hint I might offer is that if you have a small amount of epoxy glue, use it to secure the two small 20-turn pots to the circuit board after they're soldered in. This will prevent them from bending away from the circuit board as they're adjusted. Another thing you'll notice about this board is that only enough parts are included with the kit to install one of the two serial ports. My kit also included some of the parts for the second serial port (L525 thru L532 and C566 thru C573). If you also receive these extra capacitors and inductors, I would suggest installing them, even though the instructions don't tell you to. Steps 4 thru 7 on page 38 of the Assembly Manual can be eliminated if you plan on installing the HCA-150-4 Auxiliary Serial Port Upgrade. These steps will just be reversed when that upgrade is installed.

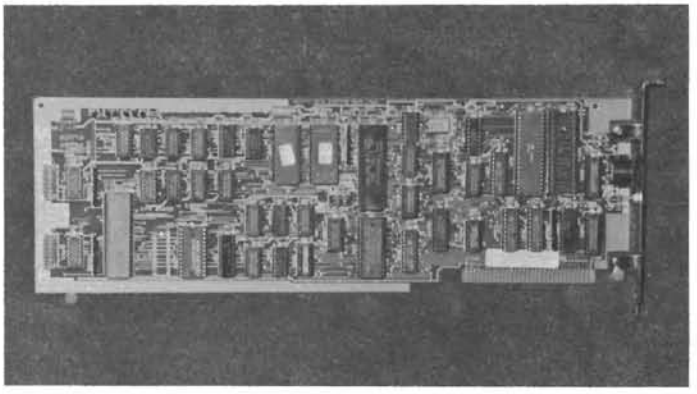
The next boards you construct are the Video Monitor and Video Driver boards. The Video Driver circuit board is a small board that mounts directly on CRT pins. Construction of this board is very easy (since it's so small), and the only problem I had was with L401, which I couldn't find because it was in a mismarked bag.

The Video Monitor board, for the most part, is fairly easy to construct. Most of the parts on this board consist of two lead components (resistors, capacitors, diodes, etc.). There was some confusion with the installation of RX333 (manual wasn't too clear), and a lot of confusion with CR325, R341, and CR326. These last three parts are installed in a section of the circuit board containing many holes, but no screening to indicate precisely where the parts should go. You're instructed to try and identify the proper holes from a picture in the Illustration Booklet, which of course, doesn't quite match the board in the first place. The manual does specifically warn, however, to use the right holes. Other than a few other minor manual errors, this was the weakest part of the whole construction project.

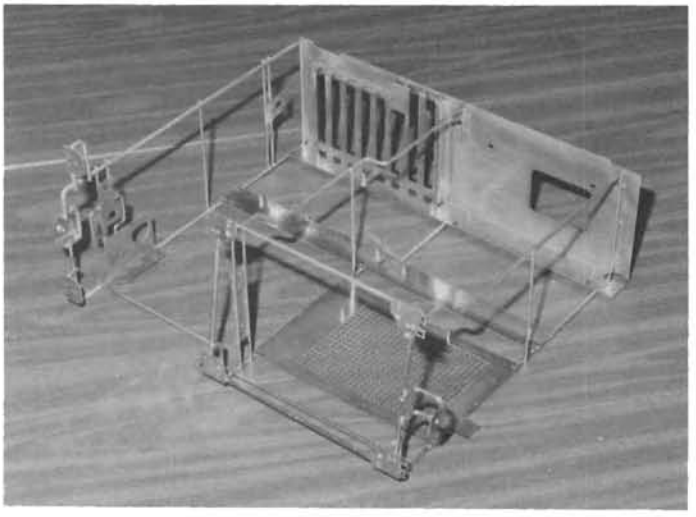
Not much construction is done on the keyboard since it's already pre-wired. You do install the keyboard processor (40-pin IC) and



Assembled Disk Controller with both serial ports installed.



Pre-wired CPU Card. Note the Diagnostic LEDs in the upper left hand corner.



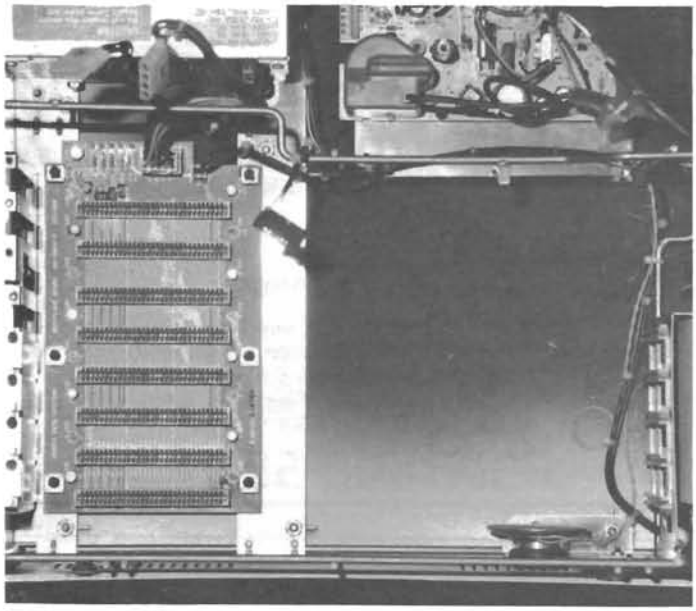
The H-161's skeleton. Everything somehow gets fastened to this wire frame.

assemble the case. The slide latches (used to lock the keyboard in place on the cabinet), appeared to be too loose and wouldn't stay in place. This problem would allow the keyboard to fall out of the cabinet when tipped. A more secure method should be found to hold the keyboard in place. Another problem I noticed was with the nylon pins which hold in the support legs. They should be longer and fit tighter.

Three or four days after starting this project, I finally came to the part of the manual entitled: "Chassis Assembly." I knew at this point I could finally let my soldering iron cool down (almost). Heath was kind enough to include an extra long 1/4" nut-driver in the kit, however, I found no use for it! I did find an 11/32" and 3/8"

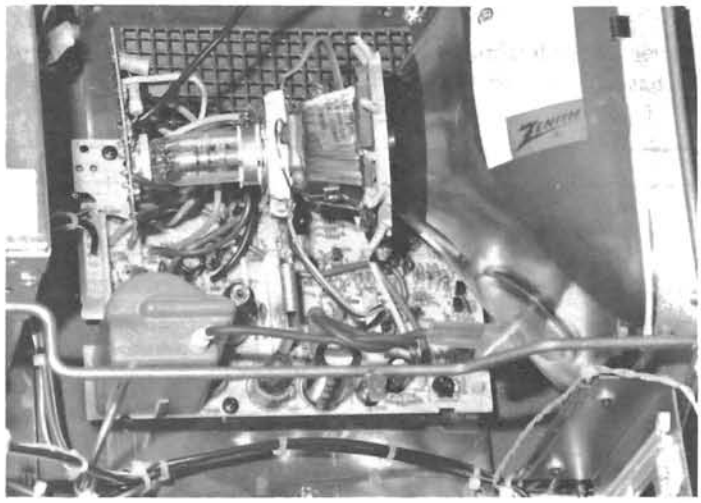
nut-driver very helpful. Having seen the poor contrast and brightness on the amber tube, I ordered my unit with a green CRT. If you receive a CRT with a bent mounting tab like I did, it is possible to bend it back with a pair of pliers or a hammer, whichever you may have handy at the time. When positioning the CRT in the chassis, due to a lack of room, I found it easier to position the yoke over the neck before securing the CRT in place. The manual is backwards in this respect. Another possible problem area is the mounting of the CRT contact spring fingers. Due to very tight tolerance between the CRT mounting brackets and the CRT, a hammer is also useful here. After soldering the coaxial cables to the brightness control, be careful in moving that control around. The problem is that those cables are not very flexible, and the control tabs will break easily.

Finally, "Diagnostic Tests!" I had made it to the part of the manual that tests your pride. I just knew everything would work right. Unfortunately, I had forgotten about Gumperson's Law which states: "The probability of anything happening is inversely proportional to its desirability," or Jenkinson's Law which says: "It won't work." What do they know! All my continuity checks were ok. All my voltage measurements were good. The CPU and Video boards worked as indicated. Apparently they knew quite a bit, my memory board failed. The most common error message you'll receive on the CRT is: "+++ ERROR: RAM failure! Address: 0000:0000, Bit: 2, Chip U407 +++." Replacing the chip usually doesn't correct the problem. The best test instruments one can have in solving a board problem such as this, is an ohmmeter, and plenty of time. Continuity checks and short tests between adjacent IC pins will usually reveal any board problems you may have with any memory board. In my case address lines 1 and 2 were shorted together by an IC socket burr on top of the circuit board. Removing the burr corrected the problem and the board operated properly.

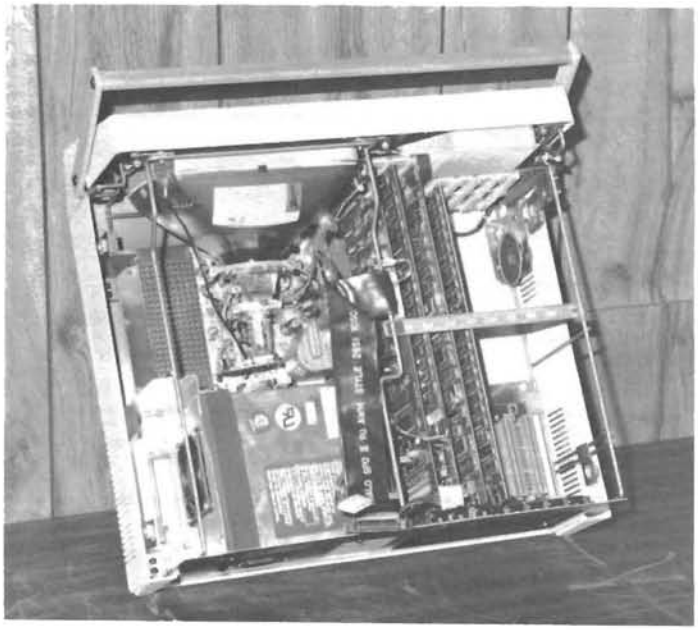


With the Backplane in place, three oversized slots are available for Heath-/Zenith boards, and five normal size slots are available for IBM compatible boards.

Murphey's second law wouldn't allow my disk controller board to work. "If there's a possibility of several things going wrong, count on it." Again, if you were careful in constructing the board, give it a good visual inspection. In my case, removing all the ICs revealed a bent pin inside the IC socket itself. Once some confidence was established in the disk controller, I continued on with the adjustments on that board. Using the audible VOM, I was able to adjust the BIAS control properly, but not the FREQ adjustment. This problem



The Video Monitor and Driver Boards mounted in place. Note the small clearance from the back of the neck of the CRT to the Power Supply.



The completed unit with all supplied circuit boards in place. Note the H-8 style circuit board retaining bar. Also note the CRT contact fingers mentioned in the text.

The H-161 backpanel showing the two RS-232 Ports, the Composite and RGB Video connectors, the Keyboard connector and Parallel Port connector.

turned out to be a defective IC at U504, the voltage controlled oscillator chip. As a rule, I would very seldom suspect an IC as being at fault. In this case however, I remembered that in the H-100, that very chip would often fail.

With all the boards somewhat working, the next test is to boot up the disk containing more diagnostics to check out the system further. Once this test passes, you can proceed with the final assembly portion of the manual.

The problem Heath had with all of its previous computers was, if the computer didn't work, or failed for some reason, the operator had no idea where to start looking for the problem. The design of the H-161 has eliminated this problem. There are diagnostic indicators on the backplane, as well as the CPU board. Depending on the condition of these indicators and the readout on the CRT, practically any problem can be traced to a single circuit board.

This machine is truly IBM compatible and will run most of the software written for the IBM PC. In addition, many user groups are springing up all over the country, dedicated to the IBM type machines. CompuServe, a large computer time share service, has a section totally dedicated to IBM and IBM look-a-like users. Contained in this section, is a very large database with hundreds of public domain programs for this computer. Dozens of hardware manufacturers are selling compatible plug-in circuit boards, as well as compatible peripherals. The IBM PC boom is here and now, and the Heath H-161 transportable, one of the best buys for the money, is the easiest way to join in.

For you kit builders out there, the introduction of the H-161 computer, has put the word 'kit' back into the name Heathkit.

Z-ARTIST:

A Graphics Program For The H/Z-100

Mark Mangerson
5200-D Ecker Dr.
Coloma, MI 49038

Z-ARTIST is a graphics editor designed to run on H/Z-110/120 series computers. It requires 192K main memory and devices of at least 32K in all three banks of video ram. I am told that Z-ARTIST will also run on a Z-150 series computer that has had the Z-319 Z-100 Video Emulator board installed.

The program runs on any version of ZDOS, under ZBASIC Version 1.0 or newer. ZARTIST.BAS should be compiled, if possible--it runs much faster as an .EXE file. When compiling the program the /E and /O switches should be used to make it a stand alone file and run faster. If the program is to be run under the ZBASIC interpreter, the size of the arrays in lines 30 and 40 should be changed from 2100 to 1500 and the for/next loop in line 510 should be changed from 9999 to 1000.

Once you have the program up and running and are starting to create pictures, it is a good idea to always have an empty formatted disk handy. A complete picture file takes up better than 156K of disk space, and you never know when a "messing around" picture might become something that you want to keep.

Command Options

Except for the Help Window, the following commands can be used almost anytime. The Help Window can be summoned only when Z-ARTIST is not in an active mode. The rest will work unless you are in the middle of an operation like "SAVE" or "GET." If the white box indicating the current color is seen flashing, you can use these toggles:

H--Displays a Help Window containing major commands.

M--Toggles single- or double-pixel resolution--status is shown next to cursor coordinates in upper right screen. Composite color blends (non-primary) will not work in single-pixel mode; and diagonal cursor movement angle will differ between the two.

U--Steps through four options: (1) Crosshairs and Cursor Coordinates on; (2) Both are off; (3) Crosshairs on; (4) Coordinates on. If you can't compile the program, option (2) runs fastest.

Left arrow--Moves color selector left across the color bar. Note: Primary colors are shown in the color bar as diamond- shaped. Black (a primary color) is located at the far left.

Right arrow--Moves color selector right.

Up arrow--Lifts the "brush" off of the screen. Status is shown on the top line of the FREEHAND mode, which is the only active mode directly affected by this toggle.

Down arrow--Puts the brush on the screen and ready to draw.

Cursor Movement

Keypad--Direction is controlled by the numbered keys--7 moves to the "Northwest," 2 moves "South," etc.

'5' Key--Moves cursor to the center of the screen.

'0' Key--Saves current cursor coordinates.

'.' Key--Moves cursor to saved coordinates.

'Home' Key--Moves cursor to upper left corner of screen.

Active Modes

Most active drawing modes can be escaped by hitting <RETURN> one or more times. Some, like END or FILES, request a Y(es)/N(o) input; these can be exited by hitting N. It is advisable that you save your work often, since it is sometimes easier to restart at your last stage than to reverse the results of an error (especially when experimenting). The PAINT mode is a particularly good candidate to follow a prudent SAVE.

! - END--Exits Z-ARTIST. Prompts you for a SAVE first.

@ - SAVE--Filenames are to be eight characters maximum, WITHOUT an extension or the preceding period. Will accept a drivespec like B:MYPICTUR or defaults to A:. Writes over old files of the same name.

 # - LOAD--Loads a picture file. Same rules as SAVE.

$ - LINE--Draws a line between user-entered points.

% - FREEHAND--Draws in the selected color in the direction entered by the numeric keypad. As noted previously, diagonal cursor movement angle depends on the state of the 'M' toggle.

^ - FILL--Fills a user-delineated rectangle with a specified color.

& - GET--Saves a screen area as an array in memory. Maximum area is 5600 pixels--some practice will teach you how big a chunk can be successfully "gotten". The named array can be used later.

*** - PUT**--Places the saved array image in a designated spot. Arrays cannot be PUT beyond the screen boundary.

(- CIRCLE--Draws a user-defined circle. Appearance depends on the state of the 'M' toggle. Circle cannot touch or go beyond the screen boundary.

) - TEXT--Prompts for foreground/background colors and position. Adjusts text to the logical line nearest the cursor position.

_ - PAINT--Paints a completely enclosed area of any shape with a specified color. Any gaps in the enclosing "stop color" will cause PAINT to leak out and fill the entire screen. Some practice with this mode is advisable!

+ - FILES--File directory/manipulator. SAVE before using.

` - CLS--Clears the screen.

- - INVERT--Flips a designated portion of the screen vertically, horizontally, or both. INVERT, COPY, and ROTATE operate on a pixel-by-pixel transfer basis and can deal with much larger parts of the screen than GET/PUT. However, you cannot put the secondary image beyond the screen boundary or overlap any part of the original image.

= - COPY--Redraws an image elsewhere, as in a GET/PUT operation. Rules for INVERT apply.

~ - ROTATE--Rotates the secondary image 90 degrees right (clockwise) or left. Rules for INVERT apply. The rotated image will vary slightly from the original due to the aspect ratio of the Z-100's video display.

Being a self-taught ZBASIC programmer, I may have missed some bugs in the program. I would appreciate hearing about any fixes and/or significant enhancements you might come up with. My apologies for the lack of REMs and the multiple-statements-per-line, but time became a factor. But, as they say, "the price is right."

```
10 COLOR 7,0 : CLS : COLOR 7,1
20 LOCATE 13,26 : PRINT " Z-ARTIST  BY MARK MANGERSON " :
   GOSUB 510
30 DIM A%(2100) : DIM B%(2100) : DIM C%(2100) :
   DIM D%(2100) : DIM E%(2100)
40 DIM F%(2100) : DIM G%(2100) : DIM H%(2100) :
   DIM I%(2100) : DIM W%(8509)
50 COLOR 7,0 : CLS : LINE (320,9)-(320,215),7 :
   LINE (321,9)-(321,220),7
60 DIM L%(360) : GET (320,9)-(321,215),L%
70 LINE (0,112)-(639,112),7 : DIM M%(241) :
   GET (0,112)-(639,112),M%
80 DEFINT A-S : DEFINT U-Z : M$="D" : COLOR 7,0 : CLS
90 LINE (0,216)-(639,224),0,BF : T%=2
100 N=-1 : X=0 : X1=28 : FOR C=0 TO 7
110 N=N+1 : FOR C1=N TO 7 : IF C1=C THEN GOSUB 160 :
    GOTO 140
120 FOR X=1 TO 14 STEP 2 : LINE (X1+X,217)-(X1+X,223),C
130 LINE (X1+X+1,217)-(X1+X+1,223),C1 : NEXT X
140 X1=X1+16 : NEXT C1 : NEXT C : X=319 : Y=112
150 LINE (0,9)-(639,215),1,B : XC1=28 : C=0 :
    GOSUB 3500 : GOTO 170
160 CIRCLE (X1+7,220),6,C : PAINT (X1+7,220),C : RETURN
170 LOCATE 1,65 : PRINT M$" X="X"Y="Y
180 GOSUB 2440 : COLOR 0,7 : PRINT " ENTER COMMAND " :
    COLOR 7,0
190 K$=INKEY$ : IF K$=CHR$(29) THEN C=C-1
200 IF K$=CHR$(28) THEN C=C+1 ELSE IF K$="H" OR K$="h"
    THEN GOSUB 2510 : 'HELP
210 IF C<0 THEN C=35 ELSE IF C>35 THEN C=0
220 XC=C*16 : LINE (XC1+XC,216)-(XC1+XC+15,224),7,B
230 C1=POINT (XC1+XC+3,220) : C2=POINT (XC1+XC+4,220)
240 LINE (XC1+XC,216)-(XC1+XC+15,224),0,B
250 IF K$="!" THEN GOSUB 1940 : 'END
260 IF K$="@" THEN GOSUB 1660 : 'SAVE
270 IF K$="#" THEN GOSUB 1750 : 'LOAD
280 IF K$="$" THEN GOSUB 570 : 'LINE
290 IF K$="%" THEN GOSUB 2020 : 'FREE
300 IF K$="^" THEN GOSUB 1030 : 'FILL
310 IF K$="&" THEN GOSUB 1140 : 'GET
320 IF K$="*" THEN GOSUB 1340 : 'PUT
330 IF K$="(" THEN GOSUB 850 : 'CIRCLE
340 IF K$=")" THEN GOSUB 640 : 'TEXT
350 IF K$="_" THEN GOSUB 1810 : 'PAINT
360 IF K$="+" THEN GOSUB 2450 : 'FILES
370 IF K$="`" THEN GOSUB 990 : 'CLS
380 IF K$="-" THEN GOSUB 2760 : 'INVERT
390 IF K$="=" THEN GOSUB 3070 : 'COPY
400 IF K$="~" THEN GOSUB 3230 : 'ROTATE
410 IF K$=CHR$(30) THEN PNT=0 ELSE IF K$=CHR$(31)
    THEN PNT=1
420 IF (K$="U" OR K$="u") THEN U=U+1 : IF U>3 THEN U=0
430 IF (K$="U" OR K$="u") THEN GOSUB 3500
440 IF (K$="U" OR K$="u") AND (U=0 OR U=3) THEN 170
450 IF (K$="U" OR K$="u") AND (U=1 OR U=2)
    THEN LINE (511,0)-(639,8),0,BF
460 IF (K$="M" OR K$="m") AND T%=1 THEN T%=2 : M$="D" :
    GOTO 170
470 IF (K$="M" OR K$="m") AND T%=2 THEN T%=1 : M$="S" :
    GOTO 170
480 IF K$>CHR$(31) THEN 180 ELSE 190
490 RESUME 500
500 GOSUB 2440 : PRINT "DISK ERROR" : GOSUB 3490
510 FOR TI=1 TO 9999 : NEXT TI : GOSUB 2440 : RETURN
520 RESUME 530
530 GOSUB 2440 : PRINT "FILE NOT FOUND" : GOSUB 510 :
    U=0 : GOTO 90
540 RESUME 550
550 GOSUB 2440 : PRINT
    "ARRAY HAS BEEN PLACED BEYOND DISPLAY AREA"
560 GOSUB 3490 : GOTO 510
570 GOSUB 2440 : PRINT "(LINE) FROM POINT" : GOSUB 2040 :
    X1=X : Y1=Y
580 GOSUB 2440 : PRINT "TO POINT" : GOSUB 2040 :
    IF X=X1 AND Y=Y1 THEN 980
```

```
590 IF C1=C2 THEN 620 ELSE GOSUB 2440
600 PRINT "LINE COMMAND REQUIRES A PRIMARY COLOR" :
    GOSUB 510
610 X=X1 : Y=Y1 : GOTO 580
620 GOSUB 3490 : LINE (X,Y)-(X1,Y1),C1 : IF T%=2
    THEN LINE (X+1,Y)-(X1+1,Y1),C1
630 GOSUB 3490 : X1=X : Y1=Y : GOTO 580
640 TX=CSRLIN : TY=POS(I) : GOSUB 2440 : PRINT
    "(TEXT) ENTER POSITION"
650 GOSUB 2040 : K$="{"
660 GOSUB 2440 : PRINT
    "ENTER FOREGROUND COLOR (PRIMARY ONLY)"
670 GOSUB 2040 : TC%=C1 : IF C1<>C2 THEN 660
680 GOSUB 2440 : PRINT
    "ENTER BACKGROUND COLOR (PRIMARY ONLY)"
690 GOSUB 2040 : TB%=C1 : IF C1<>C2 THEN 680
700 GOSUB 3490 : GOSUB 2440 : PRINT "ENTER TEXT" :
    COLOR TC%,TB%
710 TX%=INT(X/8)+1 : TY%=INT(Y/9)+1 :
    IF TX%<2 THEN TX%=2
720 IF TX%>79 THEN TX%=79
730 IF TY%<3 THEN TY%=3
740 IF TY%>22 THEN TY%=22
750 A$=INKEY$ : IF A$=CHR$(11) THEN LOCATE 3,2
760 IF TX%>79 THEN TX%=2 : TY%=TY%+1
770 IF TY%>22 THEN TY%=3
780 IF TX%<2 THEN TX%=79 : TY%=TY%-1
790 IF TY%<3 THEN TY%=22
800 LOCATE TY%,TX% : PRINT A$ :
    IF A$<>"" THEN TX%=TX%+1
10 IF A$=CHR$(8) OR A$=CHR$(29) OR A$=CHR$(127)
    THEN TX%=TX%-2
820 IF A$= CHR$(31) THEN TX%=TX%-1 : TY%=TY%+1
830 IF A$= CHR$(30) THEN TX%=TX%-1 : TY%=TY%-1
840 IF A$=CHR$(13) THEN COLOR 7,0 : GOSUB 2440 :
    GOSUB 3490 : RETURN ELSE 750
850 GOSUB 2440 : PRINT
    "(CIRCLE) ENTER CENTER POINT OF CIRCLE"
860 GOSUB 2040 : X1=X : Y1=Y
870 GOSUB 2440 : PRINT
    "SPACE CURSOR TO THE RIGHT TO INDICATE RADIUS"
880 GOSUB 2040 : R=X-X1 : IF R=0 THEN 980
 rtur ls1000
1,10)-(638,214),0,BF : GOSUB 2440 :
     GOSUB 3490 : RETURN
1030 GOSUB 2440 : PRINT "(FILL) ENTER UPPER LEFT CORNER"
1040 GOSUB 2040 : X1=X : Y1=Y
1050 GOSUB 2440 : PRINT "ENTER LOWER RIGHT CORNER"
1060 GOSUB 2040 : IF X1=X AND Y1=Y THEN GOSUB 2440 :
     RETURN
1070 IF X=>X1 AND Y=>Y1 THEN 1100
1080 GOSUB 2440 : PRINT "ERROR IN ENTRY" : GOSUB 510
1090 GOSUB 3490 : X=X1 : Y=Y1 : GOSUB 3490 : GOTO 1030
1100 GOSUB 2440 : PRINT
     "ENTER COLOR YOU WANT THIS AREA FILLED (ANY COLOR)"
1110 GOSUB 3490 : K$="{" : GOSUB 2040 :
     FOR XF=X1 TO X STEP 2
1120 LINE (XF,Y)-(XF,Y1),C1 : LINE (XF+1,Y)-(XF+1,Y1),C2 :
     NEXT XF
1130 GOSUB 3490 : GOSUB 2440 : RETURN
1140 GOSUB 2440 : PRINT "(GET) ENTER UPPER LEFT CORNER" :
     GOSUB 2040
1150 X1=X : Y1=Y : GOSUB 2440 : PRINT
     "ENTER LOWER RIGHT CORNER"
1160 GOSUB 2040 : YDIF=Y-Y1 : IF X1=X AND Y1=Y THEN 1330
1170 IF X=>X1 AND Y=>Y1 THEN 1200
1180 GOSUB 2440 : PRINT "ENTRY ERROR" : GOSUB 510
1190 GOSUB 3490 : X=X1 : Y=Y1 : GOSUB 3490 : GOTO 1140
1200 D=4+INT((X-X1)*(Y-Y1)*3/8)
1210 IF D>2100 THEN GOSUB 2440 : PRINT
     "ARRAY AREA TOO BIG" : GOTO 510
1220 AN=AN+1 : IF AN>9 THEN GOSUB 2440 : PRINT
     "NOT ENOUGH MEMORY" : RETURN
1230 GOSUB 2510 : GOSUB 2630 : GOSUB 3490
1240 IF AN=1 THEN N1$=A$ : GET (X1,Y1)-(X,Y),A% :
     YDIF1=YDIF
1250 IF AN=2 THEN N2$=A$ : GET (X1,Y1)-(X,Y),B% :
     YDIF2=YDIF
1260 IF AN=3 THEN N3$=A$ : GET (X1,Y1)-(X,Y),C% :
     YDIF3=YDIF
1270 IF AN=4 THEN N4$=A$ : GET (X1,Y1)-(X,Y),D% :
     YDIF4=YDIF
1280 IF AN=5 THEN N5$=A$ : GET (X1,Y1)-(X,Y),E% :
     YDIF5=YDIF
1290 IF AN=6 THEN N6$=A$ : GET (X1,Y1)-(X,Y),F% :
     YDIF6=YDIF
1300 IF AN=7 THEN N7$=A$ : GET (X1,Y1)-(X,Y),G% :
     YDIF7=YDIF
1310 IF AN=8 THEN N8$=A$ : GET (X1,Y1)-(X,Y),H% :
     YDIF8=YDIF
1320 IF AN=9 THEN N9$=A$ : GET (X1,Y1)-(X,Y),I% :
     YDIF9=YDIF
1330 GOSUB 3490 : GOTO 980
1340 GOSUB 2440 : PRINT "(PUT) ENTER UPPER LEFT CORNER"
1350 GOSUB 2040 : ON ERROR GOTO 540
1360 GOSUB 2440 : PRINT "ENTER ARRAY NUMBER <0> TO EXIT" :
     AN$=STR$(AN)
1370 GOSUB 2510 : GOSUB 2640 : GOSUB 1380 : GOSUB 2630 :
     GOTO 1420
1380 N$=INKEY$ : IF N$>"/" AND N$<":" THEN 1390 ELSE 1380
1390 IF " "+N$=<AN$ THEN RETURN ELSE GOSUB 2440
1400 PRINT
     "IMPROPER ARRAY NUMBER - TRY AGAIN.  HIT <0> TO EXIT"
1410 GOSUB 510 : GOTO 1380
1420 IF N$="0" THEN 980 ELSE GOSUB 3490
1430 IF N$="1" AND YDIF1+Y<215 THEN PUT (X,Y),A% :
     GOTO 1520
1440 IF N$="2" AND YDIF2+Y<215 THEN PUT (X,Y),B% :
     GOTO 1520
1450 IF N$="3" AND YDIF3+Y<215 THEN PUT (X,Y),C% :
     GOTO 1520
1460 IF N$="4" AND YDIF4+Y<215 THEN PUT (X,Y),D% :
     GOTO 1520
1470 IF N$="5" AND YDIF5+Y<215 THEN PUT (X,Y),E% :
     GOTO 1520
1480 IF N$="6" AND YDIF6+Y<215 THEN PUT (X,Y),F% :
     GOTO 1520
1490 IF N$="7" AND YDIF7+Y<215 THEN PUT (X,Y),G% :
     GOTO 1520
1500 IF N$="8" AND YDIF8+Y<215 THEN PUT (X,Y),H% :
     GOTO 1520
1510 IF N$="9" AND YDIF9+Y<215 THEN PUT (X,Y),I%
     ELSE 550
1520 IF PST=1 THEN PST=0 : GOTO 1650
1530 GOSUB 2440 : PRINT
     "IS THE ARRAY IN THE PROPER PLACE?"
1540 K$=INKEY$ : IF K$="Y" OR K$="y" THEN 1560
1550 IF K$="N" OR K$="n" THEN PST=1 :
     GOTO 1430 ELSE GOTO 1540
1560 IF N$="1" AND YDIF1+Y<215 THEN PUT (X,Y),A%,PSET
1570 IF N$="2" AND YDIF2+Y<215 THEN PUT (X,Y),B%,PSET
1580 IF N$="3" AND YDIF3+Y<215 THEN PUT (X,Y),C%,PSET
1590 IF N$="4" AND YDIF4+Y<215 THEN PUT (X,Y),D%,PSET
1600 IF N$="5" AND YDIF5+Y<215 THEN PUT (X,Y),E%,PSET
1610 IF N$="6" AND YDIF6+Y<215 THEN PUT (X,Y),F%,PSET
1620 IF N$="7" AND YDIF7+Y<215 THEN PUT (X,Y),G%,PSET
1630 IF N$="8" AND YDIF8+Y<215 THEN PUT (X,Y),H%,PSET
1640 IF N$="9" AND YDIF9+Y<215 THEN PUT (X,Y),I%,PSET
1650 GOSUB 3490 : GOTO 980
1660 GOSUB 2440 : INPUT
     "(SAVE) WHAT DO YOU WANT THIS PICTURE TO BE CALLED";
     N$
1670 IF N$="" THEN GOSUB 2440 : RETURN
1680 GOSUB 2440 : PRINT
     "MAKE SURE DISK IS INSTALLED AND HIT RETURN WHEN READY"
1690 A$=INKEY$ : IF A$<>CHR$(13) THEN 1690
1700 GOSUB 3490 : LINE (511,0)-(639,8),0,BF
1710 U=1 : GOSUB 2440 : ON ERROR GOTO 490
1720 DEF SEG=&HE000 : BSAVE N$+".GRN",0,&HCCCC
1730 DEF SEG=&HC000 : BSAVE N$+".BLU",0,&HCCCC
1740 DEF SEG=&HD000 : BSAVE N$+".RED",0,&HCCCC : RETURN
1750 GOSUB 2440
1760 INPUT
     "(LOAD) ENTER THE NAME OF THE PICTURE YOU WANT LOADED";
     N$
1770 IF N$="" THEN GOSUB 2440 : RETURN ELSE ON ERROR GOTO 520
```

```
1780 CLS : DEF SEG=&HE000 : BLOAD N$+".GRN",0
1790 DEF SEG=&HC000 : BLOAD N$+".BLU",0
1800 DEF SEG=&HD000 : BLOAD N$+".RED",0 : U=0 : GOTO 90
1810 GOSUB 2440
1820 PRINT
     "WARNING! IS THE AREA COMPLETELY SEALED IN STOP COLOR?
     (Y/N)"
1830 A$=INKEY$ : IF A$="N" OR A$="n" THEN GOSUB 2440 :
     RETURN
1840 IF A$="Y" OR A$="y" THEN 1850 ELSE 1830
1850 GOSUB 2440 : PRINT
     "(PAINT) ENTER POINT YOU WANT PAINTED" : GOSUB 2040
1860 GOSUB 2440 : PRINT
     "ENTER COLOR YOU WANT THIS AREA PAINTED
     (PRIMARY ONLY)"
1870 K$="{" : GOSUB 2040 : PC=C1 : IF C1<>C2 THEN 1860
1880 GOSUB 2440 : PRINT
     "ENTER COLOR YOU WANT PAINT TO STOP AT
     (PRIMARY ONLY)"
1890 GOSUB 2040 : PC1=C1 : IF C1<>C2 THEN 1880
1900 IF PC1=POINT (X,Y) AND POINT (X,Y)>0
     THEN PSET (X,Y),0 : PSET (X+1,Y),0
1910 IF PC1=POINT (X,Y) AND POINT (X,Y)=0
     THEN PSET (X,Y),7 : PSET (X+1,Y),7
1920 GOSUB 3490 : LINE (0,9)-(639,215),PC1,B :
     PAINT (X,Y),PC,PC1 : GOSUB 2440
1930 GOSUB 3490 : RETURN
1940 GOSUB 2440 : PRINT
     "(END) ARE YOU SURE YOU WANT TO QUIT?"
1950 A$=INKEY$ : IF A$="Y" OR A$="y" THEN 1970
1960 IF A$="N" OR A$="n" THEN 980 ELSE 1950
1970 GOSUB 2440 : PRINT
     "DO YOU WANT TO SAVE YOUR PICTURE?"
1980 A$=INKEY$ : IF A$="N" OR A$="n" THEN CLS : END
1990 IF A$="Y" OR A$="y" THEN GOSUB 1660 :
     END ELSE 1980
2000 PSET (X,Y),CS : IF T%=2 THEN PSET (X+1,Y),CS1
2010 GOTO 2040
2020 IF PNT=0 THEN PNT$="  < UP > "
     ELSE PNT$=" < DOWN >"
2030 GOSUB 2440 : PRINT
     "(FREEHAND) USE ANY COLOR BRUSH IS"PNT$
2040 A$=INKEY$ : IF A$=CHR$(13) THEN GOSUB 2440 :
     RETURN
2050 IF (A$<"1" OR A$>"9") AND A$<>"."
     AND A$<>CHR$(11) OR U=1 OR U=3 THEN 2070
2060 PUT (X,9),L% : PUT (0,Y),M%
2070 IF K$="{" THEN 2200
2080 IF A$="1" THEN X=X-T% : Y=Y+1
2090 IF A$="2" THEN Y=Y+1
2100 IF A$="3" THEN X=X+T% : Y=Y+1
2110 IF A$="4" THEN X=X-T%
2120 IF A$="5" THEN X=319 : Y=112
2130 IF A$="6" THEN X=X+T%
2140 IF A$="7" THEN X=X-T% : Y=Y-1
2150 IF A$="8" THEN Y=Y-1
2160 IF A$="9" THEN X=X+T% : Y=Y-1
2170 IF A$="0" THEN XSVE=X : YSVE=Y
2180 IF A$="." THEN X=XSVE : Y=YSVE
2190 IF A$=CHR$(11) THEN X=1 : Y=10
2200 IF A$=CHR$(29) THEN C=C-1 : IF C<0 THEN C=35
2210 IF A$=CHR$(28) THEN C=C+1 : IF C>35 THEN C=0
2220 XC=C*16 : LINE (XC1+XC,216)-(XC1+XC+15,224),7,B
2230 C1=POINT (XC1+XC+3,220) : C2=POINT (XC1+XC+4,220)
2240 LINE (XC1+XC,216)-(XC1+XC+15,224),0,B
2250 IF X>637 THEN X=1 ELSE IF X<1 THEN X=637
2260 IF Y<10 THEN Y=214 ELSE IF Y>214 THEN Y=10
2270 IF (A$="M" OR A$="m") AND T%=1 THEN T%=2 : M$="D" :
     GOTO 2290
2280 IF (A$="M" OR A$="m") AND T%=2 THEN T%=1 : M$="S"
2290 IF A$="U" OR A$="u" THEN U=U+1 : PUT (X,9),L% :
     PUT (0,Y),M%
2300 IF U>3 THEN U=0
2310 IF A$="U" OR A$="u" THEN LINE (511,0)-(639,8),0,BF
2320 IF (U=0 OR U=3) AND A$>CHR$(31)
     THEN LINE (528,0)-(639,8),0,BF ELSE 2340
2330 LOCATE 1,65 : PRINT M$" X="X"Y="Y
2340 IF A$=CHR$(30) THEN PNT=0 ELSE IF A$=CHR$(31)THEN PNT=1
2350 IF K$="%" AND A$=CHR$(30) THEN LOCATE 1,36 :
     PRINT "  < UP >  "
2360 IF K$="%" AND A$=CHR$(31) THEN LOCATE 1,36 :
     PRINT " < DOWN >"
2370 CS=POINT (X,Y) : IF CS=0 THEN CT=7 ELSE CT=0
2380 CS1=POINT (X+1,Y) : IF CS1=0 THEN CT1=7 ELSE CT1=0
2390 PSET (X,Y),CT : IF T%=2 THEN PSET (X+1,Y),CT1
2400 IF (A$<"1" OR A$>"9") AND A$<>"." AND A$<>CHR$(11)
     OR U=1 OR U=3 THEN 2420
2410 PUT (X,9),L% : PUT (0,Y),M%
2420 PSET (X,Y),C1 : IF T%=2 THEN PSET (X+1,Y),C2
2430 IF PNT=1 AND K$="%" THEN 2040 ELSE 2000
2440 LINE (0,0)-(511,8),0,BF : LOCATE 1,1 : RETURN
2450 GOSUB 2440
2460 PRINT "(FILES) THIS WILL DESTROY YOUR PICTURE.
     PROCEED? (Y OR N)"
2470 A$=INKEY$ : IF A$="Y" OR A$="y" THEN CLS :
     LOCATE 5,1 : FILES : GOTO 2490
2480 IF A$="N" OR A$="n" THEN GOSUB 2440 :
     RETURN ELSE 2470
2490 GOSUB 2440 : PRINT "HIT RETURN TO CONTINUE" : U=0
2500 A$=INKEY$ : IF A$=CHR$(13) THEN 80 ELSE 2500
2510 GET (168,63)-(447,143),W% :
     LINE (168,63)-(447,143),1,BF : COLOR 7,1
2520 IF K$="&" OR K$="*" THEN 2640
2530 LOCATE 8,23 : PRINT "<!> END" : LOCATE 9,23 :
     PRINT "<@> SAVE"
2540 LOCATE 10,23 : PRINT "<#> LOAD" : LOCATE 11,23 :
     PRINT "<$> LINE"
2550 LOCATE 12,23 : PRINT "<%> FREEHAND" : LOCATE 13,23 :
     PRINT "<^> FILL"
2560 LOCATE 14,23 : PRINT "<&> GET" : LOCATE 15,23 :
     PRINT "<*> PUT"
2570 LOCATE 8,46 : PRINT "<(> CIRCLE" : LOCATE 9,46 :
     PRINT "<)> TEXT"
2580 LOCATE 10,46 : PRINT "<_> PAINT" : LOCATE 11,46 :
     PRINT "<+> FILES"
2590 LOCATE 12,46 : PRINT "<~> CLS" : LOCATE 13,46 :
     PRINT "<-> INVERT"
2600 LOCATE 14,46 : PRINT "<=> COPY" : LOCATE 15,46 :
     PRINT "<> ROTATE"
2610 LOCATE 16,24 : PRINT
     "HIT <RETURN> TO EXIT HELP MENU."
2620 A$=INKEY$ : IF A$<>CHR$(13) THEN 2620
2630 LOCATE 1,1 : COLOR 7,0 : PUT (168,63),W%,PSET :
     RETURN
2640 TL%=CSRLIN : TC%=POS(I) : TL%=8 : TC%=22 :
     LOCATE TL%,TC%
2650 PRINT "1",N1$ : LOCATE TL%+1,TC% : PRINT "2",N2$ :
     LOCATE TL%+2,TC%
2660 PRINT "3",N3$ : LOCATE TL%+3,TC% : PRINT "4",N4$ :
     LOCATE TL%+4,TC%
2670 PRINT "5",N5$ : LOCATE TL%+5,TC% : PRINT "6",N6$ :
     LOCATE TL%+6,TC%
2680 PRINT "7",N7$ : LOCATE TL%+7,TC% : PRINT "8",N8$ :
     LOCATE TL%+8,TC%
2690 PRINT "9",N9$ : IF K$="*" THEN RETURN
2700 GOSUB 2440 : PRINT "ENTER ARRAY NAME" :
     TL%=TL%+AN-1 : TC%=28 : A$=""
2710 I$=INKEY$ : IF I$>CHR$(31) AND I$<CHR$(126)
     THEN TC%=TC%+1 ELSE 2730
2720 IF I$>CHR$(31) AND I$<CHR$(126)
     THEN LOCATE TL%,TC% : PRINT I$
2730 IF TC%>48 OR I$=CHR$(13) THEN RETURN ELSE A$=A$+I$
2740 IF I$=CHR$(29) OR I$=CHR$(8) THEN LOCATE TL%,TC% :
     PRINT " " : TC%=TC%-1
2750 GOTO 2710
2760 GOSUB 2440 : PRINT
     "(INVERT) <A> VERT. <B> HORIZ. <C> BOTH <RETURN> END"
2770 Z$=INKEY$ : IF Z$="A" OR Z$="a" OR Z$="B"
     OR Z$="b" THEN 2800
2780 IF Z$="C" OR Z$="c" THEN 2800
2790 IF Z$=CHR$(13) THEN GOSUB 2440 :
     RETURN ELSE 2770
2800 GOSUB 2440 : PRINT
     "ENTER UPPER LEFT CORNER OF ORIGINAL IMAGE"
2810 GOSUB 2040 : X1=X : Y1=Y
2820 GOSUB 2440 : PRINT
```

```
    "ENTER LOWER RIGHT CORNER OF ORIGINAL IMAGE"
2830 GOSUB 2040 : X2=X : Y2=Y : XD=X2-X1 : YD=Y2-Y1
2840 GOSUB 2440 : PRINT
    "ENTER UPPER LEFT CORNER FOR INVERTED IMAGE"
2850 GOSUB 2040 : X3=X : Y3=Y : X4=X3+XD : Y4=Y3+YD
2860 IF XD=0 AND YD=0 THEN 980 ELSE IF XD<0 OR YD<0
    THEN 2900
2870 IF X3=>X1 AND X3=<X2 AND Y3=>Y1 AND Y3=<Y2
    THEN GOSUB 2440 : GOTO 2900
2880 IF X4=>X1 AND X4=<X2 AND Y4=>Y1 AND Y4=<Y2
    THEN GOSUB 2440 : GOTO 2900
2890 IF X4>638 OR Y4>214 THEN GOSUB 2440 ELSE 2910
2900 PRINT "INVALID COORDINATES" : GOSUB 510 :
    GOTO 2760
2910 IF Z$="A" OR Z$="a" THEN 3020
2920 IF Z$="C" OR Z$="c" THEN 2970 ELSE GOSUB 2440 :
    PRINT "HIT DELETE TO ABORT"
2930 XS=X4+1 : Y3=Y3-1 : GOSUB 3490 : FOR Y5=Y1 TO Y2 :
    Y3=Y3+1 : X4=XS
2940 FOR X5=X1 TO X2 : X4=X4-1 : Z$=INKEY$
2950 CLR=POINT (X5,Y5) : PSET (X4,Y3),CLR :
    IF Z$=CHR$(127) THEN 3510
2960 NEXT X5 : NEXT Y5 : GOTO 3510
2970 GOSUB 2440 : PRINT "HIT DELETE TO ABORT" :
    GOSUB 3490
2980 YS=Y4+1 : X3=X3-1 : FOR X5=X2 TO X1 STEP -1 :
    X3=X3+1 : Y4=YS
2990 FOR Y5=Y1 TO Y2 : Y4=Y4-1 : Z$=INKEY$
3000 CLR=POINT (X5,Y5) : PSET (X3,Y4),CLR :
    IF Z$=CHR$(127) THEN 3510
3010 NEXT Y5 : NEXT X5 : GOTO 3510
3020 GOSUB 2440 : PRINT "HIT DELETE TO ABORT" :
    GOSUB 3490
3030 YS=Y4+1 : X3=X3-1 : FOR X5=X1 TO X2 : X3=X3+1 :
    Y4=YS
3040 FOR Y5=Y1 TO Y2 : Y4=Y4-1 : Z$=INKEY$
3050 CLR=POINT (X5,Y5) : PSET (X3,Y4),CLR :
    IF Z$=CHR$(127) THEN 3510
3060 NEXT Y5 : NEXT X5 : GOTO 3510
3070 GOSUB 2440 : PRINT
    "(COPY)  ENTER UPPER LEFT CORNER OF ORIGINAL IMAGE"
3080 GOSUB 2040 : X1=X : Y1=Y
3090 GOSUB 2440 : PRINT
    "ENTER LOWER RIGHT CORNER OF ORIGINAL IMAGE"
3100 GOSUB 2040 : X2=X : Y2=Y : XD=X2-X1 : YD=Y2-Y1
3110 GOSUB 2440 : PRINT
    "ENTER UPPER LEFT CORNER FOR SECONDARY IMAGE"
3120 GOSUB 2040 : X3=X : Y3=Y : X4=X3+XD : Y4=Y3+YD
3130 IF XD=0 AND YD=0 THEN 980 ELSE IF XD<0
    OR YD<0 THEN 3170
3140 IF X3=>X1 AND X3=<X2 AND Y3=>Y1 AND Y3=<Y2
    THEN GOSUB 2440 : GOTO 3170
3150 IF X4=>X1 AND X4=<X2 AND Y4=>Y1 AND Y4=<Y2
    THEN GOSUB 2440 : GOTO 3170
3160 IF X4>638 OR Y4>214 THEN GOSUB 2440 ELSE 3180
3170 PRINT "INVALID COORDINATES" : GOSUB 510 :
    GOTO 3070
3180 GOSUB 2440 : PRINT "HIT DELETE TO ABORT" :
    GOSUB 3490
3190 XS=X3-1 : Y3=Y3-1 : FOR Y5=Y1 TO Y2 : Y3=Y3+1 :
    X3=XS
3200 FOR X5=X1 TO X2 : X3=X3+1 : Z$=INKEY$
3210 CLR=POINT (X5,Y5) : PSET (X3,Y3),CLR :
    IF Z$=CHR$(127) THEN 3510
3220 NEXT X5 : NEXT Y5 : GOTO 3510
3230 GOSUB 2440 : PRINT
    "(ROTATE)  <A> RIGHT  <B> LEFT  <RETURN> END"
3240 R$=INKEY$ : IF R$=CHR$(13) THEN 980
3250 IF R$="A" OR R$="a" OR R$="B" OR R$="b"
    THEN 3260 ELSE 3240
3260 GOSUB 2440 : PRINT
    "ENTER UPPER LEFT CORNER OF ORIGINAL IMAGE"
3270 GOSUB 2040 : X1=X : Y1=Y
3280 GOSUB 2440 : PRINT
    "ENTER LOWER RIGHT CORNER OF ORIGINAL IMAGE"
3290 GOSUB 2040 : X2=X : Y2=Y : XD=X2-X1 : YD=Y2-Y1
3300 GOSUB 2440 : PRINT
    "ENTER UPPER LEFT CORNER FOR SECONDARY IMAGE"
3310 GOSUB 2040 : X3=X : Y3=Y : X4=X3+YD*2 : Y4=Y3+XD/2
3320 IF XD=0 AND YD=0 THEN 980
3330 IF XD<0 OR YD<0 THEN GOSUB 2440 : GOTO 3370
3340 IF X3=>X1 AND X3=<X2 AND Y3=>Y1 AND Y3=<Y2
    THEN GOSUB 2440 : GOTO 3370
3350 IF X4=>X1 AND X4=<X2 AND Y4=>Y1 AND Y4=<Y2
    THEN GOSUB 2440 : GOTO 3370
3360 IF X4>638 OR Y4>214 THEN GOSUB 2440 ELSE 3380
3370 PRINT "INVALID COORDINATES" : GOSUB 510 :
    GOTO 3230
3380 IF R$="B" OR R$="b" THEN 3440
3390 GOSUB 2440 : PRINT "HIT DELETE TO ABORT" :
    GOSUB 3490
3400 YS=Y3 : X4=X4+2 : FOR Y5=Y1 TO Y2 : Y3=YS :
    X4=X4-2 : A$=INKEY$
3410 FOR X5=X1 TO X2 STEP 2 : CLR1=POINT (X5,Y5) :
    CLR2=POINT (X5+1,Y5)
3420 PSET (X4,Y3),CLR1 : PSET (X4+1,Y3),CLR2 : Y3=Y3+1
3430 IF A$=CHR$(127) THEN 3510 ELSE NEXT X5 : NEXT Y5 :
    GOTO 3510
3440 GOSUB 2440 : PRINT "HIT DELETE TO ABORT" :
    GOSUB 3490
3450 YS=Y4 : X3=X3-2 : FOR Y5=Y1 TO Y2 : Y4=YS :
    X3=X3+2 : A$=INKEY$
3460 FOR X5=X1 TO X2 STEP 2 : CLR1=POINT (X5,Y5) :
    CLR2=POINT (X5+1,Y5)
3470 PSET (X3,Y4),CLR1 : PSET (X3+1,Y4),CLR2 : Y4=Y4-1
3480 IF A$=CHR$(127) THEN 3510 ELSE NEXT X5 : NEXT Y5 :
    GOTO 3510
3490 IF U=1 OR U=3 THEN RETURN
3500 PUT (X,9),L% : PUT (0,Y),M% : RETURN
3510 GOSUB 2440 : IF U=0 OR U=2 THEN 3500 ELSE RETURN
```

Finally, text processing that fits your computer to the letter.

Newline. Matched to the Zenith 100, 150 and the IBM-PC. If you have one of these systems, Newline has the right line of text processing software for you. Because Newline's Professional Text Processor (PTP) is matched to the keyboard and display characteristics of each of these computers.

There are no complicated key sequences to learn. And you'll be able to use labeled editing keys. Which means it's exceptionally easy to use.

New features for Newline's PTP. Even better, now PTP has more powerful text processing than ever before. There's everything from full screen text editing and on-screen bold, underline, paragraph fill and justification to cut and paste to configurable macro keys and much more. Plus, you'll be able to use our software with any printer.

Update from our old line. If you're presently using the Newline TxtPro software on your Zenith 100, now you can update to our more powerful version. It's called the PTP-100. If you currently have the ZDOS TxtPro, you can upgrade to ZDOS PTP-100. If you have the CP/M-85 TxtPro, you can upgrade to CP/M-86 PTP-100, but you'll also need to upgrade your system to CP/M-86. Just specify which one you have when you order. And if you return your old TxtPro disk to us with your order, you qualify for a special reduced price.

For Zenith 150 and IBM-PC users, there's the new PTP-PC. And it has all the same features as the PTP-100.

Software development editing. For programmers, Newline's PTP also offers auto indent and produces ASCII files for use with assemblers, interpreters and compilers. And that's not just different, it's unique.

Just give us the word. Newline's Professional Text Processor (PTP) is available right now. So place your order today. And get the text processing software that fits *your* system to the letter.

NAME _____

ADDRESS _____

CITY _____ STATE _____ ZIP _____

	Qty	Amount
PTP-100 (for Z100) @ **$99. or**		
PTP-100 (Z100 Upgrade) @ **$50.***		
☐ CP/M-86 (replaces CP/M-85)	____	____
☐ ZDOS	____	____
PTP-PC (for Z150 or IBM-PC) @ **Special Introductory Price of $149.**		
☐ CP/M-86	____	____
☐ MS-DOS or PC-DOS	____	____
RI Residents Add 6% Sales Tax		____
Shipping ($3. per program)		____
TOTAL ENCLOSED		____

*To qualify for Z100 upgrade price of $50. you must return your TxtPro disk.

Payment in U.S. Funds Only • Allow 2 wks. for Personal Checks • CALL (401) 624-3322 FOR C.O.D. DELIVERIES

Newline Software

P.O. Box 289 • Tiverton, RI 02878 • (401) 624-3322

An Old Friend, and a Nice Surprise

Pat Swayne
Software Engineer

(A Review of Newline's HDOS upgrade for the Z-100 and the TXTPRO editor)

In this article, I will review an operating system upgrade and a text editor. If you are wondering how those two things go together, it is because the seller, Newline Software, was selling both together as a special deal at the HUG Conference in July.

The HDOS Upgrade for the Z-100

HDOS was the first disk operating system available on computers produced by Heath Company. Those of us who have been around long enough to use and appreciate it wish that it had somehow remained in vogue, because in some ways it is still one of the most advanced operating systems around. When the Z-100 series computers came out, HDOS was not provided for it. Even though there is a HUG program that emulates HDOS (885-1223[-37]) and lets you run many HDOS programs on the '100, it still does not provide all of the advantages of HDOS. For example, with real HDOS you can have 3 or 4 printers connected to your computer and support all of them, even from a BASIC program.

Newline's HDOS upgrade is an effort to fill the HDOS void for Z-100 users. It consists of a program called Z-100 HDOS that modifies standard HDOS 2.0 so that it will work on the Z-100. To install it, you must already have HDOS 2.0 and a computer that can run it and create soft sector disks that are readable by the Z-100. That means either an H89 or H8 with a soft sector controller board. Ron Rocheleau of Newline told me that he had performed the installation himself for people who did not have the proper equipment, but only if they sent him their original HDOS distribution disks.

Since the Z-100 has different port hardware than the H89 and the H8 computers, the upgrade package also contains a program that automatically modifies the printer device driver source files that are part of standard HDOS, so that you can reassemble them and make device drivers that will work on the '100. Newline also sells their own device drivers and spoolers for use with Z-100 HDOS to provide greater flexibility in using printers.

To make your first Z-100 bootable HDOS disk, you must make a soft sector bootable disk on your H8 or H89 and then run the Z-100 HDOS program to modify the system. This program puts a new SY.DVD on the disk and modifies other system parts so that HDOS will work in the Z-100 environment. Once a modified bootable disk has been made, you must run a Z-DOS program called HBOOT before you can boot HDOS. However, you only need to run HBOOT once, and from then until you shut the power off on your computer, you can boot HDOS like you boot any other disk. That is, you can run Z-DOS for a while, and then you can CTRL-RESET and boot HDOS again. If you are using the original Heath soft sector device driver on your H8 or H89, your initial Z-100 HDOS system disk will have slow disk access, and you will have to INIT and SYSGEN a new disk on the Z-100 for maximum efficiency. However, if you are using one of the newer drivers, such as HUG's 885-1127, your first system disk will have normal access speed.

The upgraded HDOS works quite well, and I only found one problem with it. One of the drives on my hodge-podge Z-100 is a Shugart SA-455 thinline, and when you format a disk on one of those drives, the formatting program must provide a slight delay when it switches sides of the disk. The disk device driver that the upgrade provides was written before those drives came along, and did not format double-sided disks properly on my Shugart drive. (The device driver, not the INIT program, does the actual formatting of the disk.) It did, however, make good single-sided disks on the Shugart, and also made good double-sided disks on my other drive, which is a Tandon. Those disks could be read properly by the Shugart. The modification, as supplied to me, supports only 5.25-inch disks, but support for 8-inch disk and/or the Winchester may be available in the future.

Newline's modification of HDOS provides some features that even real HDOS can't provide. It allows you to swap processors in the Z-100, and to make Z-DOS system calls. Unlike 8-bit CP/M on the Z-100, it provides a real timer interrupt and a keyboard interrupt. Software requiring those features should work properly.

The TXTPRO Editor

Those who bought Z-100 HDOS during Newline's special sale also received the Z-100 HDOS version of their TXTPRO editor. This editor is also available in versions for HDOS and CP/M on the H8, H19 or H89, for CP/M-85 on the Z-100, and for Z-DOS/MS-DOS

on the Z-100. Each version is tailored specifically for the hardware it is designed for. For example, the Z-100 HDOS version would not work properly on my office H8/H29. The name TXTPRO stands for Text Processor, which is a bit of a misnomer, because it is actually an editor with some word processing capabilities. For those who are confused with the terms "editor," "text processor," and "word processor," here are what seems to be the official definitions. An editor is a program used for preparing text files on a computer. A text processor is used for processing files that were prepared on an editor, to make straight margins, paginate, etc. (such as HUG's RUNOFF). A word processor is a program that can perform both editing and text processing in the one program (such as WordStar).

TXTPRO turned out to be the highlight of the deal. It is simple enough that you can learn it in a few minutes, but powerful enough to perform some word processing duties. For example, it has a paragraph formatting capability that works like WordStar's, and even suggests where to put hyphens in words like WordStar does. A nice difference, however, is that TXTPRO only places real ASCII characters in the file, not funky control characters that make some printers go crazy if you try to print a "raw" WordStar file.

TXTPRO uses the arrow keys for cursor movement and paging forward and backward in the file. By using the SHIFT key along with the arrow and HOME keys, just about all of the cursor and text movement capabilities you need are available at the keypad. TXTPRO uses the function keys for entering commands, with the most important functions labeled on the 25th line. Some keys select sub functions, and when you press them, a new set of labels appears on the 25th line. The F0 key is an "oops" key, and can be used to undo most other actions. The HELP key causes information on the arrow and function keys to appear. Several screens of help information are provided, and you can go through all of them or press the "oops" key at any time to terminate the help session. The help screens are nicely done using graphics and reverse video effectively. I liked nearly everything about TXTPRO's usage of special keys, but I missed having the 2, 4, 6, and 8 keys serve as duplicate arrow keys. I think that every Z-100 editor should do that.

The documentation consists of 18 pages of copy, apparently prepared using a letter quality printer and possibly TXTPRO itself. All of the information on the editor is presented concisely, and for a new user, perhaps dryly, but a SAMPLE.TXT file is provided on the disk to guide new users through TXTPRO, and a demo version of TXTPRO is also provided that lets you non-destructively edit the sample file to your heart's content. The demo version will not write the edited file back to the disk when you are done, but does everything else that the real one does. Newline allows you to give away copies of the demo version.

I am kind of picky when it comes to editors (as I guess most people who write for a living are), but TXTPRO is one that I could get along with. I found no "bugs" in the program itself, but I did find a couple of typos in the SAMPLE.TXT file. For example, in one place it said "f5 key" where it should have said "f4 key." The $59.95 price tag doesn't hurt too much, either. (Z-100 HDOS sells for $79.95. Check with Newline to see if the special including TXTPRO is still on.) If you are an H8 or H89 owner using HDOS exclusively, and think that all of the good editors are only available for CP/M, give TXTPRO a try. TXTPRO cannot do every word processor trick, like bold printing or underline, (however, you can insert control characters or ESCape sequences) but it will certainly make editing files easier, especially if you are still using the HDOS EDIT program.

TXTPRO and Z-100 HDOS are available from Newline Software, P.O. Box 289, Tiverton, RI 02878, ph. (401) 624-3322.

When it comes to communications...

Pro Driver Version 2, is a professional MS-DOS communications package for Z100, Z150, and IBM PC computers. It allows transmitting and receiving of both ASCII and binary files. Clear, uncluttered menu-driven displays, plus on-line help messages make using Pro Driver a pleasure. Several new enhancements make Pro Driver even better than before! Its new "System Command Processor" even lets you run other programs such as editors or utilities, without exiting Pro Driver. Original features such as the Usage Reporter allows you to keep a detailed record of all calls made including the length of time per call, the phone number and the date of the call.

Pro Driver Version 2 will run in any Z100, Z150/160 or IBM PC computer that has at least 192K of RAM. The well documented, indexed manual includes pictorial screen displays with major sections organized with tab dividers.

The price? Still just $49 for the entire package, plus $2 shipping. Call, write or check with your local Heathkit Electronic Center.

Standard Version 1 Features

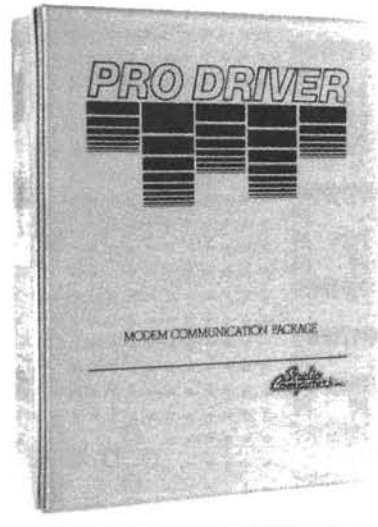
- Menu-driven request and control options
- Supports both acoustic and autodial smartmodems
- Report Generator for detailed tracking of calls
- Support of XMODEM, Compuserve B and X/ON X/OFF protocols
- 32 User-Definable Automatic Logon Sequences
- Printer Trace function for hardcopy of all data
- DOS System commands; Rename, Delete, Dir, Reset
- Selectable Baud rates from 15 up to 19,200
- Configurable entry and exit foreground/background colors, cursor modes and keyclick options
- Help files for quick assistance on Pro Driver functions
- Detailed documentation of all functions

Version 2 Enhancements Include:

- Simple Install Program
- Local Terminal Mode
- User-Definable Modem Init String
- Faster Autolog Operations
- Accepts Up to 40 digit Phone Numbers
- U.S. Robotics S100 Modem Support
- VT-52 and VT-100 Emulation Modes
- System Command Processing (Req. 256K)
- DOS 2.0 Sub-Directory Support

we know what we're talking about.

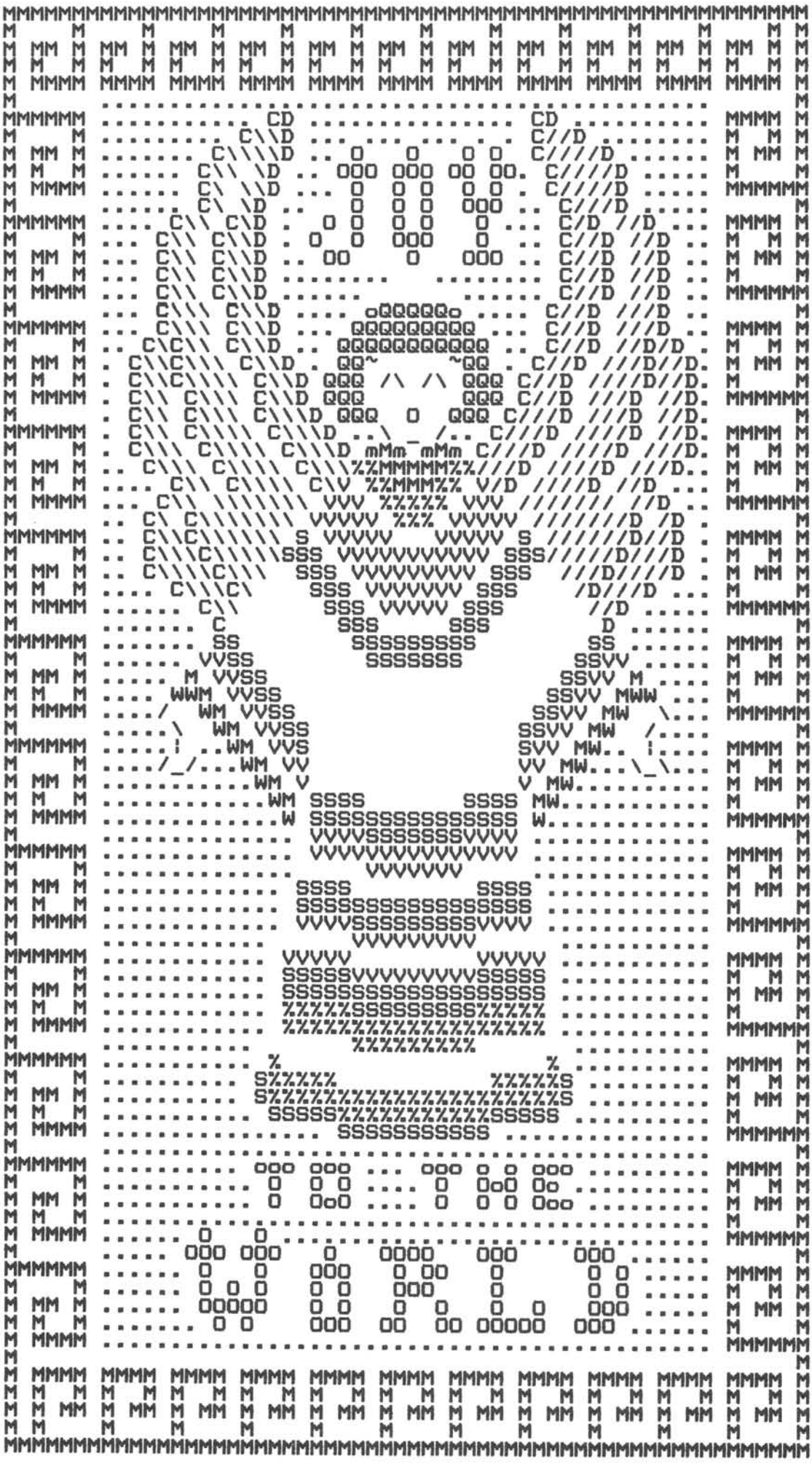
Note: Users of Z100 computers who have not upgraded to MS-DOS 2, must order version 1 of Pro Driver.



Studio Computers inc.
999 South Adams
Birmingham, MI 48011
Phone (313)-645-5365

Call or write us for our new 1985 catalog.

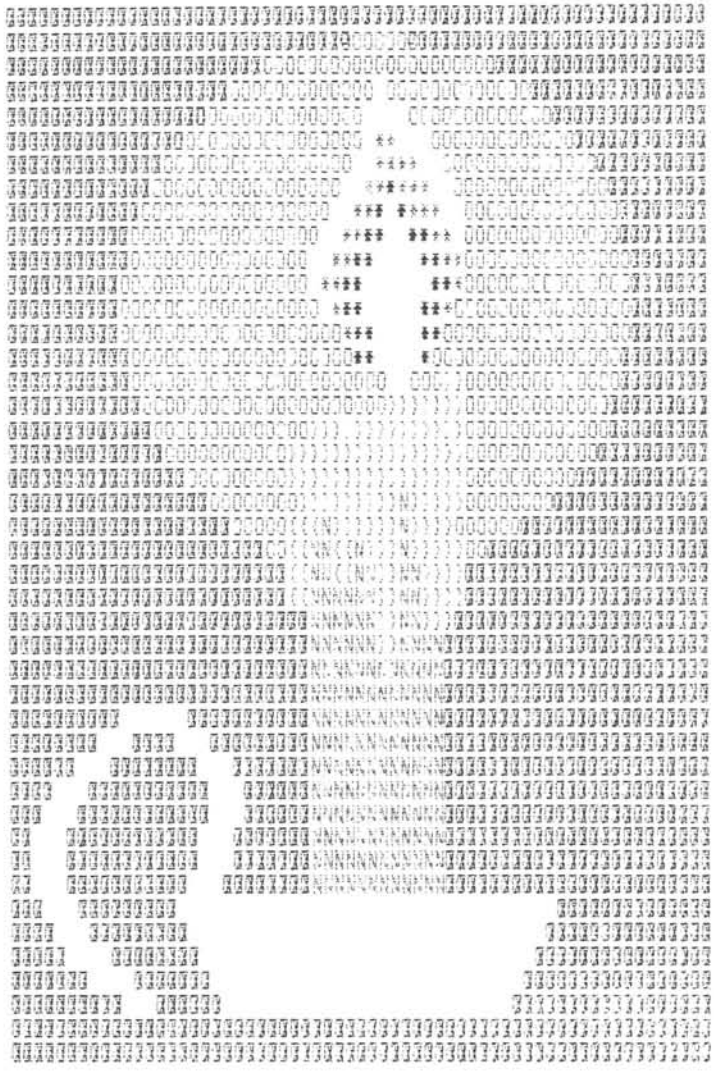
VISA MasterCard



Tom
&
Jen
McGraw

Christmas Greetings in ASCII

Jennifer T. McGraw
12741 SW 68 Terrace
Miami, FL 33183



I believe I was eighteen when I started to make Christmas cards, rather than buying them in a box. With the exception of three or four years, I've been making them ever since, using reasonably simple methods of reproduction. Silk screen printing, linoleum block, and stencils were the mainstay. Then we got a computer and a printer; so of course, in the last four years our cards are what you might call computer-generated.

Having inflicted these on our friends and family, I'll inflict them on you with a few words about how to design and print them.

The process I use will work on any printer, because ordinary printing characters are used which is a limiting factor. The card will be an impression, not a photocopy; but it is also a great advantage because it forces ingenuity. The printer used should be capable of doing a carriage return without a line feed if any especially dark areas are needed which can be done most easily with overprinting. If the printer is capable of double strike or emphasized mode, this can be handy too, because it adds more gradations from light to dark. In the examples, the ordinary printing characters are sometimes sideways or upside down when the card is complete.

Your first consideration is size. Most likely, the card is going to go in an envelope. Business envelopes are a good size, the cheapest to be bought that still have reasonable tasting glue on the flap. Then decide whether you want to trim the card, which adds more work, but frequently makes the best shape. Your final decision will be based on the design of course, but it's always a good idea to have a size in mind. The easiest to use is based on the size of a single sheet, 8-1/2 by 11 inches, with some of that going to borders.

The second thing to remember is that you cannot have a lot of detail because of the limits of the ASCII character set. The finished work will be an abstraction, and in some cases, it will look very much like a needlework design. Bearing this limitation and the size parameters in mind, you now have to develop the design itself.

The biggest problem is getting an idea. If you have a great sense of decoration and design you might just take one of the traditional greetings and make very fancy letters with fancy borders, etc, keeping in mind that this is going to be in just two colors, the ribbon color and the color of your paper. Take a look at last year's cards for an idea. Don't directly copy one; that's plagiarism, but make a few changes and you have "the sincerest form of flattery." I try several ideas and bounce them off the spouse. He's very helpful; he always says, "Yeah, yeah I like it." You must understand that this is the man who wanted to send his favorite photograph of a statue of Buddha as a Christmas card one year.

Anyway, scribble down any ideas, gradually eliminating and refining. Of course, you have lots of scratch paper; anyone who owns a printer has lots of scrap paper. When down to one or two ideas, reassess them once again in the light of size and shape. Do you want to fill a whole page, a half, a third or whatever? Is it long and skinny, a square or somewhere in between? If it is larger than the envelope, are the folds going to affect it's appearance? In the example of the horse and sleigh, the fold line is between the picture and the greeting. The printing was then glued on to a piece of red construction paper with room on the construction paper for a written message and/or signature. When folded in half the card fit comfortably in the business envelope.

The next step is to print a lot of the letter "O." You can use your editor or write a Basic program, which is easier to change. Use an old ribbon, because in the next step your design is drawn over the O's.

First, print a block of O's using regular parameters; 6 lines per inch, 80 characters across (6 LPI, 80 CPI) with numbers across the top and down the sides. The block should be the approximate size you want the finished printout to be. If your design has a lot of detail, try a block at 8 LPI with compressed characters, 132 CPI; some printers will even do 10 LPI. Print several blocks of your choice in case you need to make changes later.

Fill in the basic outline of the design on the block, indicated here by apostrophes. (It's a rayed star, sometimes known as a Disney star.)

```
0123456789012345678
100000000'00000000
2000000'0'0'000000
30000000'''0000000
40'''''''''''''''0
50000000'''0000000
6000000'0'0'000000
700000000'00000000
800000000'00000000
```

Then roughly fill it in with a preliminary choice of characters. Print a chart of blocks of characters as in Figure A to help make choices, considering not only the relative shade that a character makes, but the apparent texture. After these preliminary choices are made, use your editor and start entering it in to the computer. Make sure the editor does not substitute TABs for SPACEs, because if you have to move any characters, this will screw things up nicely.

```
0123456789012345678
1         *
2       * * *
3        ***
4 * * ******* * *
5        ***
6       * * *
7         *
8         *
```

This part is difficult because you can't really see what the end product is going to be, since most screens and printers do not use quite the same proportions for characters. But follow your plan. When completed, copy to another disk and then print it. After looking at the first print, start substituting characters and making changes you feel are needed.

```
0123456789012345678
1         |
2       \ | /
3        /*\
4 ---- <***> ----
5        \*/
6       / | \
7         |
8         |
```

Here's where patience and tenacity come in. Keep pushing it around, referring to your chart of characters. If you need any especially dark areas, consider printing first with an "O" and then overprinting with an "X" as in the CANDLE illustration. You'll probably end up printing and changing things many times before reaching some degree of satisfaction. It will not ever be exactly what you are aiming for, but there always comes a point, usually at three in the morning, when a small voice will say "That's it. No more changes." Save it, and make a copy on another disk.

If you need to do overprinting, make a duplicate line immediately underneath that line. Then go through and make the necessary changes, substituting spaces for the characters that are not to be overprinted and changing those that will. Then, if you are going to use the BASIC program below to print the card, insert "CR" at the beginning of the original line; repeat: the beginning of the ORIGINAL line. Take a look at the program to see how it is used. Or perhaps

your text formatter has a code that will allow a RETURN without line feed. If you are going to use the formatter to print, use that code in its proper position.

After all those decisions, there is one more to make: paper. If the card is going to be printed on construction paper, smooth, lightweight paper is needed. Smooth, so it doesn't jam up the printer with bits and pieces, light for ease of transport through the printer. If the printer uses tractor feed you will need a paper porter of some sort. In fact, two paper porters are better so that you can be loading one while the other is being printed. That's why I no longer use construction paper;

too much work. If you're going to use regular printer paper, use a good grade. HCS-1 from Heath is pretty good and is now laser cut for smooth edges. Or go very high class and get the bond paper, HCS-4. There are also some stores selling tractor feed paper in pastel colors. Check around.

Ok, now you should be ready to print the final copies. First, put in a new ribbon. Then there are a couple of ways to go. One is to print each copy however you usually print letters or documents. If you use a text formatter there might be a problem with control characters. Magic Wand's PRINT.COM assumes that all back slashes are control characters. TEXT.COM, from The Software Toolworks, uses a period at the left margin, as well as back slashes. If you have used any of these characters in the card, you'll have to do whatever your formatter tells you to do about getting them to print rather than controlling. Or use the following simple BASIC program:

```
10 REM  PRINT A CARD
20 DEFINT A-Z:CR$=CHR$(13)
30 INPUT "FILE NAME OF CARD TO PRINT";F$
40 OPEN "I",1,F$
50 PRINT:PRINT TAB(30)"Make sure printer is ready"
60 A$=INPUT$(1):REM     Waits for any input
61 REM  This is for ZBASIC or CP/M MBASIC. In HDOS, add
62 REM  OPEN,"O",2,"LP:" and then change all LPRINT
63 REM  lines to PRINT #2,
64 REM  The following codes are to an MX-80 printer,
65 REM  setting the printer to 8 LPI and emphasized
        printing.
66 REM  Substitute the proper codes for your printer or
67 REM  leave the line out.
70 LPRINT CHR$(27)"O"CHR$(27)"E";
80 IF EOF(1) THEN 130
90 LINE INPUT #1,L$
95 REM  Next line checks for CR code for overprinting.
100 IF LEFT$(L$,2)="CR" THEN L$=RIGHT$(L$,LEN(L$)-2):
    CR=1
110 LPRINT L$; : IF CR=1 THEN CR=0 :
    LPRINT CR$ ELSE LPRINT
120 GOTO 80
130 CLOSE
140 END
```

The program could be much fancier. A loop could be added to print the number of copies you want. In that case, it should send the codes for page length and form feed to the printer or count lines and do enough line feeds to get to the next printing position.

Or write a program to print it. If you're doing overprinting, that's the best way to go anyway. You could just go back to your editor, load in the card file, add line numbers and whatever else you need to make it a BASIC program.

```
10 REM PRINT A SMALL STAR
11 LPRINT "          |
12 LPRINT "        \ | /
13 LPRINT "         /*\
14 LPRINT "  ---- <***> ----
15 LPRINT "         \*/
16 LPRINT "        / | \
17 LPRINT "          |
18 LPRINT "          |
```

If you are using an editor which has macro-command facilities, this becomes easier. For instance, in PIE on the H-89, position the cursor at the beginning of the first line to be printed. Then enter the following keys, ignoring the commas but paying attention to the spaces:

```
ENTER,BLUE KEY,IC, LPRINT ",
IC,RETURN,BLUE KEY,ENTER,60,
BLUE KEY
```

and PIE will proceed to put "space LPRINT space quote" at the beginning of the next 60 lines. Then enter the line numbers and any beginning set up lines and closing lines that you need. You may run in to a problem here, since PIE will accept only 80 characters per line. If that is the case, stick a semicolon at the end of the first part, and add a line number, LPRINT and quote to the second part. Save it, load BASIC, load the card program and run.

Or you could just print from the system. With HDOS, just use PIP LP:=CARD.CRD. With CP/M, PIP LST:=CARD.CRD. (Sorry, I don't know the command in ZDOS or MSDOS.) This gets tedious, because each card requires that the command be repeated.

If you have a hardware printer buffer, like the ANGEL reviewed in REMark some time ago, it can make as many copies as you want with any of the above methods.

The parameters in the examples of cards are as follows:

CANDLE is 8 LPI, 132 CPI and printed upside down.
SANTA is 10 LPI, 80 CPI and printed sideways.
JOYTO is 8 LPI and 80 CPI.

I hope this is enough to get you started printing your own Christmas cards. It will also work for posters, or any type of picture you want to make. Look at the examples and do your own thing.

Figure A: Appearance and texture of some characters

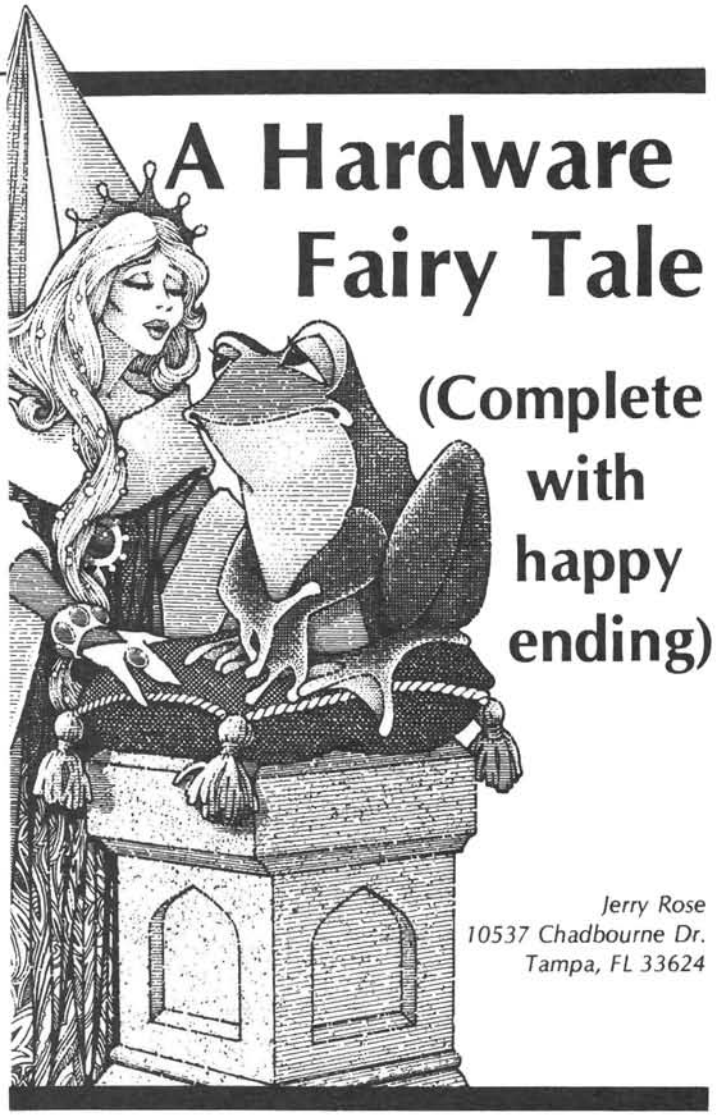
```
.......  '''''''  ,,,,,,,  ///////  <<<<<<<
.......  '''''''  ,,,,,,,  ///////  <<<<<<<
.......  '''''''  ,,,,,,,  ///////  <<<<<<<
.......  '''''''  ,,,,,,,  ///////  <<<<<<<

:::::::  [[[[[[[  LLLLLLL  YYYYYYY  ZZZZZZZ
:::::::  [[[[[[[  LLLLLLL  YYYYYYY  ZZZZZZZ
:::::::  [[[[[[[  LLLLLLL  YYYYYYY  ZZZZZZZ
:::::::  [[[[[[[  LLLLLLL  YYYYYYY  ZZZZZZZ

*******  VVVVVVV  %%%%%%%  KKKKKKK  XXXXXXX
*******  VVVVVVV  %%%%%%%  KKKKKKK  XXXXXXX
*******  VVVVVVV  %%%%%%%  KKKKKKK  XXXXXXX
*******  VVVVVVV  %%%%%%%  KKKKKKK  XXXXXXX

0000000  &&&&&&&  #######  MMMMMMM  WWWWWWW
0000000  &&&&&&&  #######  MMMMMMM  WWWWWWW
0000000  &&&&&&&  #######  MMMMMMM  WWWWWWW
0000000  &&&&&&&  #######  MMMMMMM  WWWWWWW
```

OOOOPPPSS!

In the article "An Easy Method For Drawing the United States In Z-BASIC" (Nov. '84, Pg. 17), the file "Natpark.doc" should be "Maps.doc". The last 2 lines, starting with "Death Valley . . ." should be listed under the file "Natpark.doc". Sorry for any inconvenience.

A Hardware Fairy Tale

(Complete with happy ending)



Jerry Rose
10537 Chadbourne Dr.
Tampa, FL 33624

Once upon a time, I sat here at the keyboard of my favorite home computer: Harry the H89. Harry and I had been together for about a year and a half. We got along all right for the most part, but there were problems that existed in our relationship. My major complaint was the fact that his built in disk drive would always fill up right in the middle of saving a file I'd just spent hours editing and fixing up. His major complaint was that I was wearing out his disk drive door from swapping disks back and forth while copying files, trying to get the most out of that limited storage space. We both agreed that something needed to be done. This article shares my experiences in dealing with these complaints.

I first went to back issues of REMark that I have and looked for any articles I could find on the subject of expanding disk storage capacity on the H89. I found two such articles in Issue 32 (September 1982). One dealt with adding the soft-sectored, double-density controller board, the other with modifying the CP/M BIOS and adding a double-sided 96 tpi (tracks per inch) drive. I felt I'd rather not try to mix hard and soft so the first article was forgotten. The second sounded much more like what I wanted to do, but to me, 96 tpi floppies are like 120 minute audio cassettes or 3600 feet reel-to-reel tapes. Although there's not any danger of the floppies stretching (a common problem with those length tapes) and causing problems, I felt caution should be used in trying to cram too much onto a single disk. It could also be very time consuming to try to adapt the methods in the article to my purposes.

The information in this article was kept in mind as I began searching the hardware ads to get a feel for what was currently on the market. I actually started reading the ads in several available computer magazines, including the one you are now reading, instead of just admiring the pictures as I had in the past. I soon discovered that there was a wide variety of disk drives available, many of which that could easily be connected to Harry's hard disk controller. The biggest problem was that they all cost money!!

How I solved that problem is not really very interesting, so for the sake of the fairy tale we'll pretend I found a willing Good Fairy to support the project. As it turned out, this was not a rich fairy, so I concentrated on looking for a standard size drive and enclosure. What I wanted originally was a clone of the built-in drive. This would take care of Harry's complaint, since I could then copy disks without wearing out his door. I'd only pick up an extra 5k of storage per disk though, figuring that the extra drive would service my data diskettes which wouldn't have to be bootable. But, that was still double what I had now.

I was now ready to get serious and start narrowing the field to some specific drives and then shop for the best supplier of the selected drives looking for the best deal. I found that this turned out to be a chicken versus the egg search. I ended up searching the dealer ads to see what was available and used the advertising information to determine what models I was interested in, writing down the company name and price as I went. I was looking for single-side 48 tpi drives. The model that ended up on my list the most was the Tandon TM-100-1. That doesn't mean that that is the most popular drive, just that it is the one I wrote down the most that satisfied my requirements. There is a wide range in prices, from $150 (no brand name listed) to around $199 (TM-100-1) for a bare drive. If the enclosure and power supply were included, the price naturally was higher. I did run across some ads for unknown drives for around $199 complete, so you really have a lot of options available to choose from.

I also came to the conclusion that a half-height drive would not be that much more expensive and would leave room in a standard enclosure for further expansion someday (provided the power supply was suitable for dual drives). I was coming to the realization that another 95k really wasn't going to satisfy me.

I invested $3 in postage and sent a request for information to 15 companies stating specifically what I was looking for and what hardware it would attach to. I found the Heath/Zenith Related Vendors section of the annual Policy & Reference Issue of REMark (January 1984) to be extremely helpful for obtaining possible suppliers.

It took about 8 days before the first reply arrived and they came in at the rate of about one a day for the next week or so. Some companies sent fliers on drives and peripherals, others just sent their standard catalog/price sheet. No company took the time to respond directly to my questions. The most interesting piece of information I got from all these catalogs was not about any drive!

In a discussion of the Tandon drives in the catalog from Studio Computers, Inc. (a REMark advertiser), they mentioned that with the BIOS-80© software package (offered on another page) the H89 with standard hard sectored controller could handle dual-sided, 48 or 96 TPI drives!! And I thought I had a wide variety of choices just considering the single-sided drives.

I immediately turned to the page indicated to see if I could indeed turn my stock '89 into a souped-up model. BIOS-80© is a BIOS modification done by Livingston Logic Labs that provides far more than the article in REMark (mentioned at the beginning, remember?) does. It seemed to be just the thing I could use and $40 was an attractive price. I could have attempted to make the BIOS modifica-

tions myself, guided by the information in REMark, but with my busy schedule I felt the price was much cheaper than the time I would have had to expend doing it myself (not to mention faster!).

I then re-searched all the ads and priced dual headed drives. Again, Tandon was the most noticed name and although the prices were higher, the k per dollar was lower than the single head drives. It was kind of like buying the large economy size box at the supermarket. This takes us back to my earlier thoughts on going to a 96 tpi drive. The 80 track drives offered the best value on a cost per k basis. So, if you really want the most you can get on a single disk and can afford the extra cost, you can go that route. Harry and I decided our Good Fairy could only afford spending half again what we'd planned in order to get twice the storage capacity by getting a double-sided 48 tpi drive. We are talking about $300 for 190k versus about $200 for 95k (formatted usable capacity).

Having made the decision, I checked my price comparison sheet and found that I could get a HI-TECH 500 Series, Model 548-50, 48 tpi, DSDD (double-sided, double-density) for $180 (plus shipping) from MRD Systems (10172 Mardel Dr., Cypress, CA 90630). Granted that was the only place I had seen that brand drive, but the price was not so low that I smelled a come-on and yet low enough to warrant serious consideration. Harry and I duly considered it and ordered it! An order also went out to Floppy Disk Services, Inc. (39 Everett Drive Bldg. D, Lawrenceville, NJ 08648) for BIOS-80©, a dual-drive vertical half-height enclosure and power supply (Model 5SVA&T, $68), and a data cable ($30). Unless you are into making cables and can therefore buy the various parts in quantity, I don't think a homemade cable would be that much cheaper. The one hitch I ran into was that Floppy Disk Services was in the middle of moving and it ended up taking almost 3 weeks for everything to finally arrive.

So, now the fun began! The first thing to do was to make a copy of the BIOS-80© disk and use that copy to create my new multi-disk system. That done, it was a simple matter of following the excellent instructions found in the BIOS-80© installation guide to create my new BIOS. I made one minor deviation from the printed instructions in that I used SWEEP to copy files rather that PIP. The utility used makes little difference. The important thing is to get the proper files on the new system disk.

There turned out to be more advantages to buying the modified BIOS rather than trying to do it myself. The package includes 3 utility programs: FORMAT80, MOVCPM80, CONFIG80 that are the BIOS-80© versions of the normal Heath CP/M utilities FORMAT, MOVCPM17 and CONFIGUR. Three items of public domain software are also included that constitute an extra added bonus:

1. XDIR-an enhanced directory listing that combines the features of various other DIR type commands, including a sorted list of file names with the room each uses, directory entries used and number left, number of k used and number left out of the total available.

2. FINDBAD-a non-destructive disk test and bad block lockout utility that will mark any unusable (bad) blocks found and thus prevent any program trying to access them.

3. ZCPR-a Z80 console command processor that enhances current commands (DIR, ERA, REN, etc). For example, ERA will list all files deleted, which is handy if you use a wildcard; TYPE will automatically stop after a screen full of data has been displayed and will continue when any key is pressed. It also provides several new commands, including LIST to send data directly to a printer. Overall, an excellent value for the $40 purchase price.

I then connected the data cable to the FR-1 pins on the back of Harry, colored edge (pin 1) to the right (looking from the back). The printed circuit board (PCB) in the drive is marked with a "2" at the top of the card edge connection so the card edge connector of the data cable was connected with the colored edge at the top. There is a small drive select jumper located just in front of the connector. It is labeled "DS0," "DS1," "DS2" and "DS3." These relate to CP/M's drives A, B, C and D respectively. Mine is jumpered to DS1 and is my B: drive. I later tested it as drive C: (DS2) just to make sure it would work ok. It did! I cut a piece of cardboard just high enough to fit in the empty space in the dual drive enclosure and about twice as wide. I then scored it on one side and folded back the edges so that it fits perfectly in the missing drive's space. I did this to cut down on the amount of dust that would flow through the empty space. It's not particularly pretty, but it was cheap!

The enclosure comes with an additional power connector so that I can add another drive when the financing is available. The power supply is a heavy duty model with an on/off switch and fuse mounted in the back of the enclosure. Slots are cut in the heavy duty metal frame to allow for heat dissipation. I did have to rummage around to find the two machine screws needed to mount the drive to the bottom of the enclosure. Otherwise, installation was a breeze.

I was now ready for the trial by fire! I booted my new system disk and attempted:

```
A>B:<cr>
```

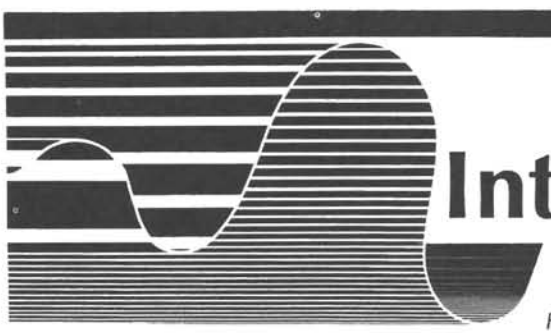
The new drive's select light came on briefly followed by a Bdos select error. What a beautiful error that was! Feeling much braver at this point, I did a reset and put a blank disk in the new drive. I then ran FORMAT80, told it the disk was on drive B: and it asked me if I wanted to format one side or two. I decided to go all out and selected two. As the clock bonged Midnight (I'm serious, ask my wife!), I successfully formatted my first double-sided disk! This brings up another interesting point. I have not purchased any double-sided diskettes (although I always insist on double-density. I don't want my data fading away). The way I figure it is that all single-sided diskettes are those that flunked the double-side certification test of their manufacturer. In 4 months, I have had only one single-side certified disk that has been formatted double-sided that has given me any trouble and I merely reformatted it as single-sided and continue to use it! As long as you don't attempt to store irreplacable data on one without a backup (who would ever do such a thing?), I feel you can safely save the $5-$10 extra per box that double-side certified diskettes cost. This is, of course, my personal opinion!!

The questions may rightfully be asked: Was it worth it? Did I really need another disk drive? As I said after I got my microwave oven, my VCR, and Harry: You bet! I don't know how I ever survived without one.

And Harry and I are living happily ever after.

About the Author

Jerry Rose *is in charge of Systems Programming for the Digital Equipment Corporation PDP 11/70s that make up the Intelligence Data Handling System for the United States Readiness Command at MacDill AFB, Florida. He has also written several applications programs for this system including an automated budgeting system that is now in use throughout the Air Force's automated intelligence communications facilities. At home he uses his H89 to provide word processing for his church position and has developed a budgeting system in Microsoft BASIC he hopes to submit to the HUG library.*



Interlaced Anyone?

Frank T. Clark
Zenith Data Systems
Software Consultant
St. Joseph, MI 49085

Introduction

There has been a lot of talk and wishful thinking by H-100 owners concerning the potential for increasing the video resolution of their computers by using the interlaced operation of the video. I noticed in particular at the HUG conference a lot of questions and talk about it but nothing concrete. It seemed that a lot of software developers downplay interlaced graphics as not being too interesting even though a lot of people express an interest. There are interlaced software packages available but the vendors offering them keep a low profile. This is probably because of the problem of support. I suspect that this is not more available because offering an interlaced product would require an awful lot of work. In this article I will attempt to briefly explain some of the basics of interlaced operation and why it is not commonly available. This will not be an exhaustive or even in depth coverage of the topic. It would require an entire series of articles to even begin to cover the subject as will be soon explained.

What Is Interlaced Video?

The first thing that needs to be done is to explain what interlaced operation of the video is. This is necessary for those who know something about it as a review and for those who know nothing about it so that they can follow along also. This article will attempt, as far as possible, to keep from getting too technical on this subject. This is necessary both to allow for general interest and to keep the content down to a manageable volume. A little bit of technical information will be presented and explained. The Z-100 Technical Manual (model number TM-100) is a must reading for those who are interested in the details of what will be discussed.

First, as a review, we will discuss the normal operation of the video. The H-100 video screen is composed of 25 rows by 80 columns. Each character row is made up of nine lines. Multiplying nine by 25 gives us 225 lines of information in the display. These are referred to as scan lines. Each scan line is divided into 640 pieces which can be individually lighted or dark. These pieces are referred to as pixels. There are eight pixels per character so 640/8 = 80 columns. It is important to understand these terms because they appear over and over again in any discussion of video displays. For the sake of simplicity we will only talk about black and white (or green, or amber) operation. This is referred to as a monochrome display. The word monochrome means one color no matter what that color is. Black or something very close to it is almost always the background color.

The scan lines are drawn on the screen one by one at a very high rate of speed. This process is controlled in part by a device called a CRTC (CRT Controller). The entire screen is drawn 60 times every second. This is referred to as the refresh rate. The reason you do not see the screen blinking is that the color which makes up the pixel keeps glowing long enough after it is drawn that it gets drawn again before it fades very much. The color which makes up a pixel is referred to as the phosphor.

If you look very closely at a screen with writing on it, you can see that there is a small gap between every scan line. What the CRTC can do is to shift the screen of information every other time it is refreshed (drawn) so that the space between the scan lines is filled in. This procedure of placing every other screen with the scan lines woven together is called interlacing.

There are two ways the CRTC can perform this interlacing procedure. One way is for the two screens of information to be exactly the same, except for the fact that they are drawn in slightly different places on the screen. This is referred to as interlaced video only. The second way is for the two screens to be different parts of the same picture. This is referred to as interlaced video and memory. The memory is of course the area where the picture image comes from. The interlaced video only method does not require any extra memory to store the video image. The interlaced video and memory mode requires double the amount of memory for the same picture.

Why Would You Want It?

A very good question to ask is why would you want to use the interlaced mode of operation. The interlaced video only mode has the advantage of filling in the spaces between the pixels which gives the image on the screen a more solid appearance. The interlaced video and memory mode provides double the resolution of the image that is drawn on the screen. The interlaced video only mode is usually not considered to be too attractive for most people because the advantages of filling in the spaces are not that great. The interlaced video and memory mode is often of considerable interest because of the ability to double the vertical resolution of the picture. The normal mode of operation usually has somewhere near double the number of pixels per inch horizontally than it does vertically. Doubling the vertical resolution brings the number of pixels per inch to somewhere very nearly equal, which provides for a smoother picture and easier graphics calculations.

There are some problems with interlaced operation. The main problem is that the picture on the screen is only updated every other time the refresh rate drops to 30 frames per second instead of 60. This rate is slow enough that on most screens the phosphor fades out before it is refreshed again. For some people this fading out is perceived as a flicker to the screen which some find to be very annoying. There are video monitors available which have a long persistence phosphor which fades out so slowly that the flickering is not a problem. These monitors tend to be a bit more expensive than the usual monitor.

Since the interlaced video and memory mode is the most useful and

popular mode of operation for the purposes of the rest of this discussion, we shall talk only about this mode and simply refer to it as interlaced. It is important to be aware that when some people talk about interlaced operation they may be referring to the interlaced video only mode, so don't be surprised.

Who Can Do It?

As most people are aware the H-100 computer has been designed from a hardware viewpoint with the potential for interlaced mode of operation in mind. There are several things that it is necessary to be aware of. Since interlaced operation requires double the memory space, the 32k video RAM chips which are perfectly alright for normal video are insufficient for interlaced operation. If a particular H-100 computer has 32k RAM video chips, it will need to be changed to 64k RAM video chips for successful interlaced operation.

When using interlaced mode, the flicker may be considered objectionable. It might be necessary to replace or augment the current video monitor with one that uses the long persistence phosphor. This problem is one of personal taste. I have been using interlaced operation for over a year with a normal monitor and I do not find it objectionable. It is also important to note that different monitors often have different persistence of their phosphors. This is particularly true with different colors of phosphors where the amber tend to be longer persistence and the white a short persistence with green somewhere in between. Careful adjustment of the monitor brightness, contrast and the room lighting can also minimize the problems with flicker.

From a software viewpoint the situation is very grim. None of the standard system software is designed with or even intended for adaptability to use in the interlaced mode. Since the video considerations are at such a deep and pervasive level, it is extremely difficult to develop software to handle the interlaced mode. As an example, if one wanted to do a directory with a DIR command and have it appear in interlaced, it would require bypassing the BIOS and replacing practically all of the ROM character handling functions. Later versions of the ROM tried to implement some procedures to improve the situation to reduce the level of difficulty, but the result has in fact made it worse. This has happened because the development effort has now been fragmented between different possible methods based on different versions of the ROM. If there had only been one general way to do it, then all the effort from the various third party software developers would have reinforced each other. It is still possible (just about anything is possible if you try hard enough) but the difficulty level is very high.

How Is It Done?

In spite of all these glum problems let us discuss just how interlacing can be implemented without going into any great detail. In particular, just for curiosity sake a very simple little program will allow you to turn on and off the interlaced video only mode. This least interesting of the interlaced modes does not require any changes to software because only the video output changes. The following simple program is written in assembly language and can be entered with EDLIN and then assembled using the following commands:

edlin interlac.asm
masm interlac;
link interlac;
exe2bin interlac.exe interlac.com

The source code is as follows:

```
        name interlac
        title.  INTERLAC - turn on or off the interlace
                           video only mode

code    segment
        assume  cs:code,ds:code,es:code,ss:code
        org     0100h
begin:
        cli
        mov     al,8
        out     0dch,al
        mov     al,ds:[082h]
        and     al,1
        out     0ddh,al
        sti
        ret
code    ends
        end     begin
```

The program is run by entering INTERLAC 1 to turn the interlacing on and INTERLAC 0 to turn it off. This simple demonstration of the interlace mode should work for everyone and will allow you to evaluate the flicker problem for yourself. I would point out that the interlaced video and memory mode will not be quite as bad because not all the dots will be flickering at the same time which masks the problem.

When one considers the problem of using the regular interlaced video and memory mode, there are certain categories that are immediately apparent. We can simplify the problem and by accepting certain limitations we can reduce it to a more manageable size.

Since the H-100 is a pure graphics machine, characters are drawn on the screen just like any other graphic item such as a line or a circle. Drawing characters is much more complex though because of the considerations of the 96 character set, positioning on the screen and other such things. As mentioned earlier you cannot use the normal methods to write characters when you are in interlaced mode. Therefore, one way of simplifying the problem is to forgo the use of characters or draw them just like anything else.

When we place the consideration of characters aside, the next thing that is apparent is that to simplify the problem further an interlaced graphics program must operate independently of all normal video features. Therefore, the program must initialize the CRTC for interlaced operation, perform all video display itself and then restore normal operation. In particular, program input must not be allowed to echo because this would require the use of normal video.

This leaves us with a very primitive program and yet the level of computer software required to do even this very simple task is still very high. It must be done in assembly language and would probably be beyond the abilities of the average hobbyist. It might possibly be even beyond the abilities of the above average hobbyist. This means that the hobbyist is at the mercy of the almost professional and professional hobbyists most of whom work for software companies or third party vendors. If there really is an interest in interlaced operation, possibly a future article could deal with a very primitive program to do interlaced video and memory graphics.

Even though there is some interest in interlaced operation there have not been the articles and discussion from the HUG members that usually accompany an interesting topic. If the HUG members themselves do not develop something, then the only other source is the software vendors. Most companies do not believe there is enough interest to make it worth the cost of developing such an expensive piece of software that too few people will buy to make it worth the investment. As you have heard here in REMark and other places, one of the things that is strangling the software market is the problem of piracy. In the case of a package that will be expensive and of low interest, the problem of piracy becomes a self-fulfilling prophecy.

The casual user will say to himself that he won't use it enough to

justify the cost, so he will just get a copy from a friend. With the potential of lost sales in mind, the software company has to charge more than would have been necessary, if everybody who used it purchased the package. This increase in cost means that more people will be tempted to pirate it and we have a vicious circle which escalates to the point where no company that wants to stay in business will spend the money on resources for a product it can't possibly sell enough of to recoup the investment.

Conclusion

Interlaced graphics is an extremely interesting idea to play with but regrettably is not commercially viable. Since it is too complex to be developed by the average hobbyist and does not have enough interest to draw the attention of software vendors, we probably can not expect to see anything generally available. At first, it looked like interlaced operation would be a really hot topic, but the interest dried up so fast that its chances don't look good any more. Even in an open and expressive forum like REMark there have really only been a few people who have expressed an interest. I am sure that if enough people expressed enough interest that something would become available. Remember that ideas come from the readers and if the HUG members do not bring forth ideas into public discussion and submit them to be made available to others then the ideas will pass their time and die.

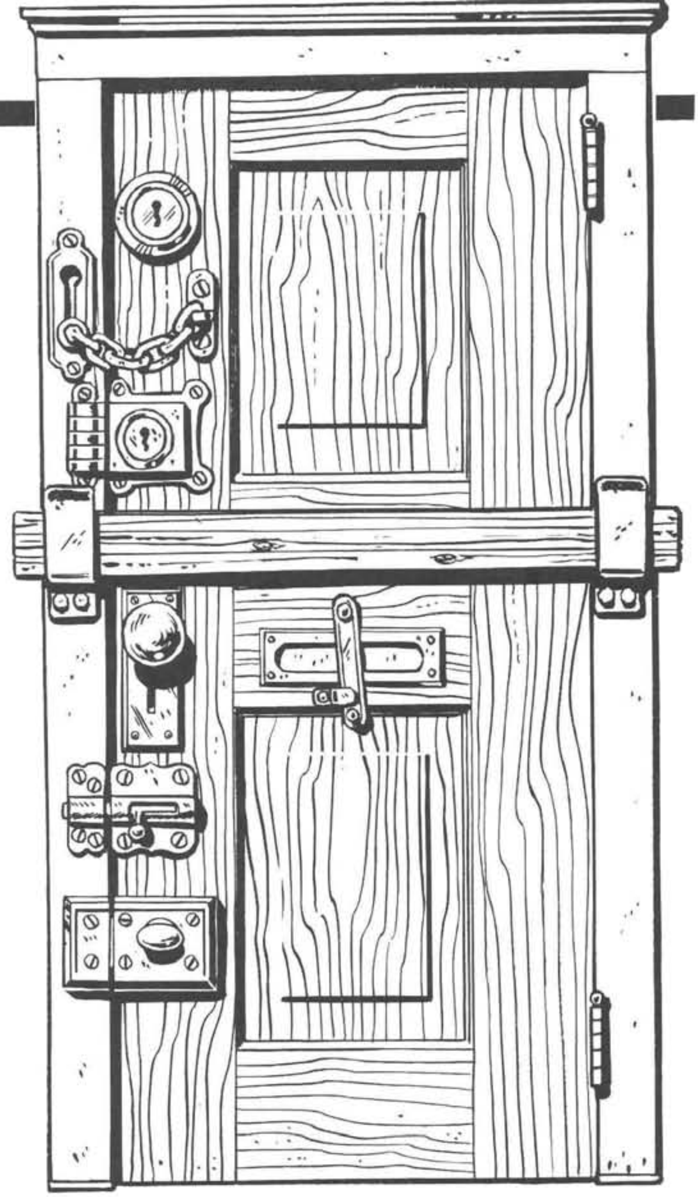
What's The HUG B.B. or HUG SIG?

It's HUG members just like yourselves using their computers and modems to exchange ideas and solve problems.

It's also a large Data Base containing hundreds of Public Domain programs that are shared amongst the HUG and SIG members.

If you're not part of the HUG/SIG, then why not get started today. Join HUG's Micronet Connection (PN 885-1122-[37] HDOS; PN 885-1224-[37] CP/M; $16.00) which gives you your ID number and password and your first hour of connect time *FREE* or call Jim Buszkiewicz at (616) 982-3837 for more information.

MBASIC Revealed: / MBASIC Concealed



John Broome
229 Hancock
Lake Bluff, Illinois 60044

Microsoft's Basic has become a de facto standard for microcomputers, and rightly so. It is tremendous for "getting the job done." This is not to say it is perfect. For instance, if you owned a small company, would you be comfortable with a BASIC payroll program? I doubt it. It would be just too easy for one of your employees to give himself an unofficial raise at line number 2230. Again, if you made your living by writing programs, would you use BASIC? Unlikely, with Bluebeard's Software Factory just across town, in days your weeks long effort would be on the street at half your price. While absolute security is as elusive as absolute perfection, you can make improvements - if you know a little of the mechanism by which MBASIC handles your program.

Some of what follows has been covered in depth by Mr. Dodgen ("Inside Microsoft Basic," page 13, REMark, August 1984). While I recommend that you read his article, it is not essential for you to understand what we will be attempting. I have tried to give enough detail so you won't have to continually switch between the two REMark issues.

Now let's get down to specifics. I use MBASIC version 4.82 on an H-89A running under HDOS 2.0. If you are set up differently, I will show you how to find out if these ideas will work for your system. No promises that they will, but at least you will be able to look at your programs from the point of view of your computer. All BASIC languages are a sea of machine code. MBASIC is an ocean. If you want to find your program, you have to know where the water ends.

Listing 1 gives a general purpose BASIC program which finds its start. It looks for the four slashes in the REM statement on the first line. Why four slashes? Firstly, because neither the "logo" which appears when you load MBASIC, nor the error prompts have this string. Secondly, the ASCII code for a slash is 47 (all numbers are decimal), in 8080/Z80 code this is the CPL command. This changes all the ones in the A register to zero and all the zeros to ones. It is remotely possible that machine code could have two consecutive bytes with a value of 47 - one of them being part of a totally different instruction - but it would make no sense to have four in a row. Hence when ////

```
10 REM////
20 DEFINT A-Z
30 J=17535:        REM USERFWA - 1
40 :
100 J=J+1
110 P=PEEK(J)
120      IF P<>47 THEN 100
130 :
200 REM            ONE "/" FOUND.
210 P=PEEK(J+1)
220      IF P<>47 THEN 100
230 :
300 REM            TWO "/"'s FOUND.
310 P=PEEK(J+2)
320      IF P<>47 THEN 100
330 :
400 REM            THREE "/"'s FOUND.
410 P=PEEK(J+3)
420      IF P<>47 THEN 100
430 :
500 REM            ALL "/"'s FOUND.
510 FOR I=J-6 TO J+15
520      P=PEEK(I)
530      PRINT I,P,
540                IF P>32 AND P<128 THEN PRINT CHR$(P);
550      PRINT CHR$(7)
560 NEXT
570 END
```

Program 1. A BASIC program to find its own start in RAM. The target is the collection of slashes on the first line. Once this is found, the contents of the memory is dumped to the screen - see Figure 1.

is reached, we can be pretty sure that the target has been found. At this point, the program scans both backwards and forwards in RAM to output the contents of each byte. If a byte is a printable ASCII character, this is also printed.

```
29595       0
29596       166
29597       115       s
29598       10
29599       0
29600       143
29601       47        /
29602       47        /
29603       47        /
29604       47        /
29605       0
29606       176
29607       115       s
29608       20
29609       0
29610       174
29611       32
29612       65        A
29613       243
29614       90        Z
29615       0
29616       197
```

Figure 1. A facsimile of the screen produced by running Program 1. See the text for interpretation.

If this program is run on my system, the screen display is that given in Figure 1. At first glance you can see the four slashes, the "A" and "Z" on line 20 and the two line numbers, 10 and 20. The rest is not too illuminating. So let's make haste slowly. The screen starts with a zero byte at address 29595. This is actually the last byte of MBASIC. For some reason Microsoft BASIC programs have to have a zero byte "behind" them, so it is included in MBASIC rather than the program. The next two bytes are very important, combined they are a pointer. In the usual Z80 way, the first is the Least Significant Byte (LSB), the second the Most Significant Byte (MSB). You can find out where this word is pointing by multiplying the MSB by 256 and adding the LSB. In this case the answer is 29606, which is the byte after another zero byte. Look again, the zero at address 29605 comes just after the fourth slash and is obviously the end of the first line. Hence, this pointer is pointing to the start of the second line (# 20), which is another pointer. Should you calculate the value of this pointer, it will point to the pointer in the third line, and so on. What about the last line of a program? In fact, the program will end with three zero bytes, one for the end of the line, two to indicate no more program. Why don't you modify Program 1 to confirm this? It is this "linked list" structure which is largely responsible for moving MBASIC through your program. Only when you use a GOTO or a GOSUB are the line numbers literally involved.

As we have seen, with Microsoft BASIC the line numbers are rather dull. You can see them at address 29598, 29599 and again at 29608, 29609. They are two byte integers which are evaluated just like the pointers mentioned in the last paragraph. They play a slightly subordinate role in the scheme of things. In the case of MBASIC, there is one intriguing difference which we will examine after about another two paragraphs. Now there remains only one byte in the first program line which has not fallen into place. Namely, the 143 at address 29600. When you entered the program, you typed "R," "E" and "M"; this has shrunk to "143" - a single byte. This is an important point. MBASIC maintains a list of reserved words. When any of these are used, it replaces it by a "token" - in this case REM = 143. Much space is saved. If you had made this first line a multi-statement line, with colons separating each statement, you would find that while the colon is not strictly a token, it is allowed to remain in its ASCII form.

Can you assume that all MBASIC programs start at address 29596? Categorically, no. When you load MBASIC, you have the option of using the "F" switch to specify the number of disk files which will be open at the same time. The more you want, the larger is MBASIC. Fortunately, there is an infallible way to find the start of your application program. MBASIC depends on a slew of pointers - RAM which contains not machine code, but addresses to tell where routines, variables and arrays are located. One of these always points to the start of your program. It is at 10427 and 10428 in the usual LSB, MSB format. Hence,

```
PEEK(10427)+256*PEEK(10428)
```

will return your starting address. Indeed, if you were to poke a higher address into this word, type NEW, <RETURN> and then load your program, it would start at the address you specified - BASIC is truely relocatable!

The address of this "BASIC Start" pointer just given applies to my Version 4.82 MBASIC. There is no reason to believe that other versions use the same absolute address, although the pointer almost certainly exists. How to find it? The only answer I can give you (Watergating Microsoft aside) is entirely devoid of elegance. My protocol was as follows. Break the absolute start address of BASIC into LSB, MSB format. Do the same for the second line of a program which printed wherever the first address pair appeared - there were many. In turn, poke the second line address bytes into each of these sites and check if I had amputated the first program line. If I had not, I restored the word to its status quo and moved on. Tedious, but once that first line was lost, I was sure I was home-free. If you are forced to travel this route, do observe the two rules given below.

The reason for this preamble lies on page 1-2 of my manual: "Line numbers must be in the range of 0 to 65529." Why 65529, when the limit of an unsigned word is 65535? I guess I must have the instincts of an anarchist. My next thought was what happens if the line number is between 65530 and 65535? I soon found that if you type

```
65535REM
```

what you get - on listing - is

```
6553 5REM
```

which is not particularly useful. But, we now know where the line numbers are; we can poke them in.

So far, we have been systematically PEEKing at the grist for the MBASIC mill. In no way can this damage software. At this juncture, we are about to start some calculated POKEing: if misplaced, this can create havoc with software. Hence, a few precautions are prudent.

Rule #1: Do not assume the addresses I have given apply to YOUR system. Run Listing 1. Does it check out? If it does not, THINK before POKE.
Rule #2: Use the RESET option of MBASIC to replace the disk(s) with disk(s) having both write-protect tabs and back-ups.

This is the time for you to turn on your system, run Program 1, also select one of your programs which you know is bug-free: we have some real-time surgery to do. Observing both rules, load your program. Next, add a new first line such as

```
1 REM
```

to ensure that this line number was never referenced by the program. Now find where the program starts in RAM as given above. We know that the address you have found is that of the LSB of the pointer. Add one to it and that is the pointer MSB. Increment again and we are at

the LSB line number address. POKE 255 here. Increment to get to the MSB and POKE 255 again. We have a line with the number 65535. Run your program - it runs fine. Now try listing it - no listing! Try editing a line number you know to be present. It can't be done. Can it be saved? No problem. Will it load. Again, no problem. We have a program which is RUNable, SAVEable, LOADable, but unLISTable and unEDITable! I thought that was rather interesting until I found how easy it was to defeat. In the command mode again, type RENUM. Now it is listable again, although the line numbers are different. Some protection!

For a time I was stuck. On the one hand, I wanted to disable RENUM. While I was sure I could find the relevant machine code, I could not figure out how to guarantee that the section which I patched was not used by some other routine. Then the obvious dawned. DON'T TOUCH THE CODE, BUT CHANGE THE NAME. It's tough to renumber a program if each time you enter RENUM you get a syntax error. Every BASIC interpreter has to have some kind of look-up table so that it understands its keywords. Program 2 gives a program so that you can locate yours. It scans the interpreter, looking for printable ASCII. While much of what it finds is machine code which just happens to be printable, it will catch all true ASCII. You will need to be nimble with your Control S's and Control Q's, but try and hold a screen with a chunk of RAM around 9500. Figure 2 shows the relevant section of my RAM. You can recognize that this is a list of keywords with their first and last bytes missing. They are not really missing, but have bits set, presumably to flag start and end, which make them unprintable. We can change RENUM to RE*UM with the command POKE 9524,42. It's as simple as that.

```
10 DEFINT A-Z
20 I=8192:     REM        START IN LOW CORE
30 K=29595:    REM        END OF INTERPRETER
40 PRINT CHR$(27);"E"
99 :
100      IF I= K THEN END
110 L= PEEK(I)
130      IF L>32 AND L<128 THEN GOSUB 1000
140 I= I+1
150 GOTO 100
160 :
1000 PRINT I,
1010     IF L<32 OR L>127 THEN PRINT: RETURN
1020 PRINT CHR$(L);
1030 I= I+1
1040 L= PEEK(I)
1050 GOTO 1010
```

Program 2. This finds any printable ASCII in the RAM between the limits given by variables I and K. When found, it is printed on the screen. Use the Control S keys to hold a screen and the Control Q keys to restart scrolling.

We are now at the stage where we can formulate a reasonable protection strategy. Let's start with your fully developed application program. It is bug-free and bullet-proof no matter how many typos it is forced to input. We are going to make this into a "turnkey" disk. Also, if you are thinking of selling your program, arrange that one screen prominently displays the name of the legitimate licensee, so as to deter piracy. This disk will comprise a minimal, but adequate, collection of HDOS system files, your program and a file named PROLOGUE.SYS. HDOS is set up to process any file called PROLOGUE.SYS at the end of a boot, before control is passed to the operator. Our PROLOGUE is taken from the back of the first section of the second volume of the HDOS manual - the MENU program. This loads device drivers, MBASIC and finally, runs your program. By this stage, your program should have a few additions. Namely, a preface which checks if PROLOGUE is present on SY0:. If it is not, it

9494	ESTOR	9554	G
9501	E	9557	Q
9504	ESUM	9560	I
9510	SE	9563	TR
9514	IGHT	9567	TRING
9520	N	9574	PACE
9523	ENU	9580	YSTE
9528	ESE	9587	HE
9534	TO		
9538	WA		
9542	AV		
9546	PC		
9550	TE		

Figure 2. An ASCII dump of the section of RAM holding BASIC keywords. The first and last bytes of each keyword has bit 7 set so that they are not printable. RENUM starts at address 9522.

gracefully exits to HDOS, because that is not the way we had planned for it to be used. Having crossed this hurdle, the preface clobbers RENUM. Also, your exit should be changed from an END to a SYSTEM, so that the BASIC Command Mode is normally never entered during execution. Finally, the first program line is a do-nothing REM line numbered 65535.

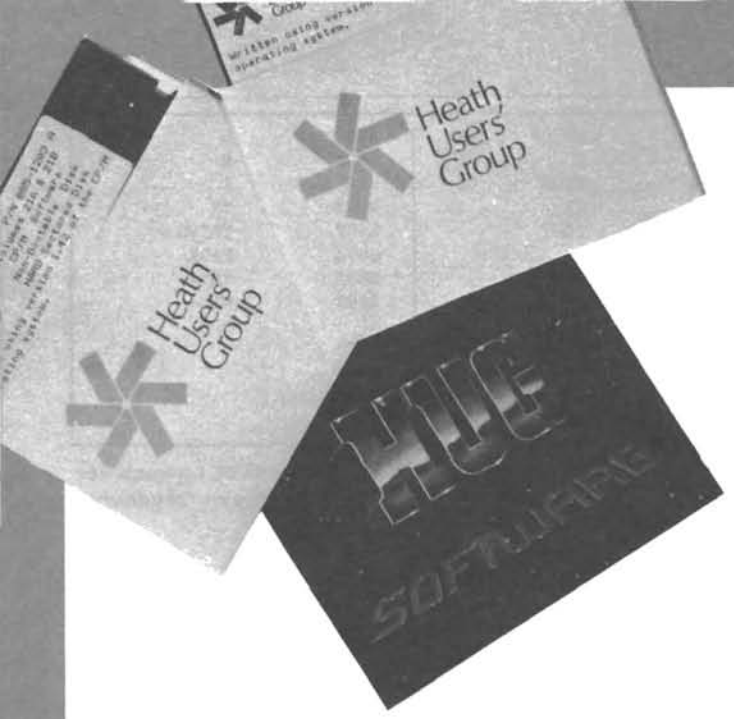
One way to implement the preface and error trap is given in Figure 3. The REM statements explain its operation. How secure is a scheme such as this? Well, for starters, anyone who has read this article would have little difficulty in cracking it. However, providing your program works well, it is unlikely to anger the user. There should be

```
1 REM   Line to be given a number of 65535.
10 ON ERROR GOTO 50000: REM Set up error trap.
20 OPEN "I", #1, "PROLOGUE.SYS":
   REM See if file is on SY0:
30 CLOSE #1: REM Free up channel #1.
40 POKE 9524, 42: REM Clobber RENUM.
100 REM Your program starts here.


9000 SYSTEM: REM End of your program.

49999 REM Error Trap.
50000    IF ERR= 53 AND ERL= 20 THEN 9000:
         REM PROLOGUE.SYS not found.
50010 REM Your error trapping routines follow.
```

Figure 3. The security aspects of a protected BASIC program. The REM statements explain the rationale.

no valid reason for the user to need to edit, list or renumber your program. We could incorporate further difficulties, such as renaming POKE, but this could irritate the user. Think of that ultimate abomination, the copy-protected disk.

On the other hand, pretend you have not read this article and that your intent is to tamper, nefariously, with a program you believe to be written in BASIC. Unknown to you, the program has been protected as we have outlined. You boot-up and you are in the application program - you can't avoid this because of the presence of the PROLOGUE file. No problem, hit control C. But, this is where your problems really start. You can't get a listing. If you go to HDOS and try TYPEing the file, you have screens of garbage because of all those tokens. If you are going to succeed, you must make the inspired guess to transfer the program file to a disk which does not have the PROLOGUE file and avoid RUNing the transferred program. Then you have to have another inspired guess: that a useable RENUM is the key. Of course, any bright fourteen year old will instinctively arrive at the correct solution, but would you?



HUG NEW PRODUCTS

NOTE: The [-37] means the product is available in hard-sector or soft-sector. Remember, when ordering the soft-sectored format, you must include the "-37" after the part number; e.g. 885-1223-37.

885-1132-[37] HDOS Tiny BASIC Compiler $25.00

Introduction: TBC, or Tiny BASIC Compiler, is a set of programs, written to compile standard BASIC source code into assembly language source code. This assembly source is then assembled into a standard executable binary file (.ABS). Although it is called "tiny," TBC has commands available that can be found in more sophisticated versions of BASIC, as well as H/Z-19 graphics commands not found in any versions of BASIC!

Requirements: TBC will run on any H-8 or H/Z-89 with 48k of memory and one drive, although two or more drives are highly recommended for ease of use. Version 1.6 or higher of HDOS is also needed. If an H-8 is used, an H/Z-19 terminal is needed to make use of the graphics commands, as well as other compile time features. A printer is needed to get a hardcopy of the 18 page manual that comes on Disk #2.

The following programs and files are included on the HUG P/N 885-1132-[37] Tiny BASIC Compiler disks:

Disk 1

TBC	.ABS
CHOMP	.TBC
CHOMP	.ABS
XFER	.TBC
XFER	.ABS
XFER	.DOC
LIB1	.ACM
LIB2	.ACM
LIB3	.ACM
LIB4	.ACM
LIB5	.ACM
LIB6	.ACM
LIB7	.ACM
READ	.ME
XTBC	.ABS
README	.DOC

Disk 2

TBC	.MAN

Author: Patrick J. O'Connor

Program Content: The following is a list of commands, keywords, and extensions, to give some idea of the capability of this Tiny BASIC Compiler. This list is taken directly from the manual.

Tiny BASIC Compiler With Z19 Extensions

Standard Keywords:

1. PRINT
2. IF/THEN
3. LET
4. INPUT
5. ON/GOSUB
6. FOR/STEP
7. NEXT
8. GOSUB
9. RETURN
10. GOTO
11. CHR$
12. READ
13. OUT
14. INP
15. RND
16. DATA
17. RESTORE
18. REM
19. ABS
20. PEEK
21. ON/GOTO
22. DIM
23. END

Standard Keyword Extensions:

1. LINK
2. ASM
3. ENDASM
4. PAUSE
5. INKEY$
6. IMPARY
7. POINTER
8. CALL

File/Device Extensions:

1. OPEN
2. WRITE$
3. READ$
4. CLOSE
5. EOF

String Handling Extensions:

1. DIM$
2. LET$
3. LINE-IN
4. RIGHT$
5. LEFT$
6. MID$
7. INSTR
8. PRINT$
9. STRING$
10. ADD$
11. LEN
12. VAL
13. IF$.EQ.
14. IF$.NE.

Bit Manipulation:

1. AND
2. OR
3. XOR
4. SHR
5. SHL

Z19 Extensions:

1. GOTOXY
2. SKETCH
3. DRAW
4. GRAFON
5. GRAFOFF
15. RVON
16. RVOFF
17. CURSAV
18. CURSET
19. CUROFF

6. CLRSCRN
7. CURHOM
8. CURFOR
9. INSTROFF
10. INSRTON
11. CURDWN
12. CURUP
13. DELCHAR
14. CURBAK
20. CURON
21. REVINX
22. EN25TH
23. DIS25TH

Comments: Since this is not an interpretive type BASIC, the source code will have to be created using some sort of text editor. Any will suffice. Also, since assembly language source code is created by this compiler and since assembly language source can be inserted directly into the BASIC source, this compiler very much reminds one of the "C" language.

TABLE C Rating: (10), (0)

885-1133-[37] HDOS
Games Collection I $25.00

Introduction: The programs on these disks are a collection of games written in various programming languages. These include Assembly, Tiny Pascal, Benton Harbor BASIC, and MBASIC. Most of these games will appeal to the younger crowd, including pre-schoolers. The game of Donkey is aimed at these younger folk, however, I found it quite challenging in trying to beat the computer at Monopoly (MONOP).

Requirements: This game package requires the HDOS operating system on an H/Z-89 with 56k of system memory. All of the games, except OTH(ello), will work on an H-8 with an H/Z-19. OTH(ello) requires a Z80 processor board. MBASIC is required for MONOP, and PAKRAT. Benton Harbor BASIC is needed for BHCRAZY8, MATCH, and TTT.

The following programs or files are included on the HUG P/N 885-1133-[37] HDOS Games Collection I disks.

Disk 1

CATNDOG	.ABS
FIFTEEN	.ABS
FIFTEEN	.ABS
OTH	.ABS
OTH	.ASM
OTH	.TXT
PROCLIB	.TPI
RATS	.ABS
RATS	.DOC
RATS	.TPS
WARI	.ABS
README	.DOC

Disk 2

BHCRAZY8	.BAS
DONKEY	.ABS
DONKEY	.ASM
DONKEY	.DOC
MATCH	.BAS
MONOP	.BAS
MONOP	.DOC
PAKRAT	.BAS
PAKRAT	.DOC
TTT	.BAS

Authors:

CATNDOG -- David G. Pelowitz
FIFTEEN -- John A. Curtis
OTH -- Jack Curtis
RATS -- Gary Gramblitt
WARI -- David G. Pelowitz
BHCRAZY8 -- R.E. Wethington
DONKEY -- James R. Gockley
MATCH -- Kenneth Kirby
MONOP -- Robert Ripberger
PAKRAT -- Gordon Wheaton
TTT -- Kenneth Kirby

CATNDOG is a game geared to the young user, 4 to 10 years of age. It is a checker board type game in which the player (DOGS) must try to trap the computer (CAT). It only uses one step look ahead with a random factor to break ties. This keeps the game simple, but still allow some variation.

FIFTEEN is a computer rendition of the old favorite plastic puzzle we had as kids with fifteen numbered squares and one empty square. When randomized, the puzzle is solved by moving the numbered squares one at a time into the blank space, with the final goal of getting all the squares in numerical order.

OTH is the full game of Othello written in assembly language using some Z80 op-codes. This program also uses the H19/H89 graphics capabilities.

Rats And Dragons is a fast action graphics game in which the object is to move a mouse around the screen in order to eat a piece of cheese without being caught by an evil dragon who relentlessly pursues the mouse. Each time the mouse runs into the cheese, the score is increased and a new piece of cheese randomly appears on the screen.

WARI is a game of logic for a more mature player. It requires some lower math skills and is patterned from the popular Oh-Wah-Ree game published by 3M Bookshelf Games. It is perported to be over 3500 years old. This game uses H19/H89 graphics to build the playing board.

BHCRAZY8 is a card game which uses H19/H89 screen graphics to play Crazy Eights. The game is similar to "War," except that Eights are wild and will match any suit. You play against the computer, and the winner is the first player to use his last card.

DONKEY is an excellent graphics game for the pre-schooler. The object of the game is to drive a car down a road without hitting any donkeys. The only interaction needed is for the player to hit any key to make the car change lanes to avoid hitting any donkeys.

MATCH is a graphics game that pits two people against each other, with each one trying to match up the most hidden symbols. The players receive points for each match, and the player with the most points when all matches have been made, wins.

MONOP is a full screen graphic game of Monopoly. This version allows one player to play against the computer or against another player.

PAKRAT is a program that gives you a rat's eye view of a computer generated maze. Using H19/H89 graphics, this program shows you what a maze looks like from the inside. It has three levels of difficulty.

TTT is a graphics game in which two people can play Tic-Tac-Toe, or one person can play the computer.

Comments: None.

TABLE C Rating: (0), (1), (2), (9)

UPDATE
885-3009 ZDOS
Dungeons & Dragons (DND) $20.00

This is an announcement that our popular DND game for ZDOS has be updated with several enhancements.

The functional description in REMark, Volume 5, Issue 3, March 1984 is still accurate. Play has not been changed in any way. The files included now have different names, since a compilable version is also included with this disk.

The modifications made to DND are mainly code corrections and bugs too numerous to mention. The two biggest mods are: (1) now the game plays in full color, not just a minimal implementation and (2) a compilable version is included. The original will not execute properly when compiled.

Bug corrections and enhancements by Leslie L. Bordelon, further bug corrections by Jim Buszkiewicz.

Anyone wishing to update their version of DND can do so by sending the original disk and $5.00 to Nancy Strunk, HUG, Hilltop Road, St. Joseph, MI 49085.

885-6001-37 MS-DOS Z-150 KEYMAP

Function Key Mapper $20.00

Introduction: Z-150 KEYMAP is a program that lets you designate the responses produced by your computer's function keys. It works like the KEY command in GW-BASIC (BASICA) except that more keys can be defined, and the defined keys are part of the "system" and can be used with any program or the operating system, not just with BASIC. For example, you could have the F1 key produce the command DIR A: (including the RETURN), so that a single press of that key would give you a list of the files on your disk.

Requirements: Z-150 KEYMAP requires the MS-DOS or PC-DOS operating system (version 1.25 or higher) on an H/Z-150, H/Z-160, or any PC-compatible computer, and at least 128k of memory.

This disk contains the following files:

README	.DOC	KEYBAS	.DOC
KEYMAP	.DOC	KEYWS	.COM
KEYMAP	.COM	KEYWS	.DOC
KEYCON	.COM	UNMAP	.COM
KEYSYS	.COM	KEYMAP	.ASM
KEYSYS	.DOC	KEYCON	.ASM
KEYBAS	.COM	UNMAP	.ASM

Author: Patrick Swayne, HUG

KEYMAP -- This is the executable KEYMAP program, provided in unconfigured form so that you can set up the keys the way you want to. It allows you to define a response of up to 20 characters for each of the following keys: the Arrow keys, Home, PgUp, PgDn, End, Ins, Del, F1 through F10, SHIFT-F1 through SHIFT-F10, CTRL-F1 through CTRL-F10, and ALT-F1 through ALT-F10. You can designate any one of those keys as an alternate response key, which gives all of the other definable keys two responses of up to 20 characters each. A total of 50 different responses can be produced without an alternate response key, or 99 responses with one. In addition to the ability to define keys, Z-150 KEYMAP offers these other features:

** On/Off toggle. A control code (normally CTRL-SHIFT-6) is provided to toggle KEYMAP on or off, so that it can be temporarily disabled to allow other programs to control the function keys.

** Coexistence with other programs. Not all of the keys must be configured with KEYMAP. Some can be left "unconfigured" so that KEYMAP can coexist with programs such as GW-BASIC. For example, if you configure only the shifted function keys, the GW-BASIC key command will still work properly with the unshifted function keys.

** Optional 25th line prompt. If you wish, you can have KEYMAP display a prompt on the 25th screen line designating the response of the keys F1 through F10, like KEY ON in GW-BASIC. While the prompt is on, output to the screen appears on the first 24 screen lines, and the the 25th line remains under the control of KEYMAP, until a program accesses it directly.

** "Scan code" mapping. A key can be mapped to its own or any other "scan code." ("Scan codes" are the codes normally produced by function or keypad keys when KEYMAP is not in use.) For example, you could reverse the functions of the left and right arrow keys, and have each one produce the other's code.

KEYCON -- This program is used to configure the KEYMAP program, and allows you to designate the response of each mappable key. It is self prompting, and makes setting up custom key responses easy.

KEYSYS -- This is a pre-configured KEYMAP for use with the operating system. Commands such as DIR, DATE, TIME, etc. are available at the press of a key.

KEYBAS -- This is a pre-configured KEYMAP for use with GW-BASIC. 30 BASIC commands are "programmed" into your keys, without interfering with the KEY command or the editing keys.

KEYWS -- This is a pre-configured KEYMAP for use with WordStar. It provides more control via the function and keypad keys than you get without KEYMAP, and is similar to the configuration used in other releases of KEYMAP. Use it as it is, or use it as a guide to setting up WordStar to your own preferences.

UNMAP -- This program disables KEYMAP, and lets you install a differently configured KEYMAP without re-booting.

TABLE C Rating: (1), (3), (10)

NOTE: For use on other Heath/Zenith computers, HUG offers the following KEYMAP programs:

885-1230[-37] -- CP/M KEYMAP, for use on an H8 (with Heath-/Zenith terminal), H/Z-89,90, or Z-100 and 8-bit CP/M.

885-3010-37 -- Z-DOS KEYMAP, for use on a Z-100 series computer under Z-DOS or MS-DOS.

885-5001-37 -- CP/M-86 KEYMAP, for use on a Z-100 series computer under the Heath/Zenith release of CP/M-86.

885-6002-37 MS-DOS CP/EMulator II

And ZEMulator $20.00

Introduction: CP/EMulator II is a program that lets you run standard 8-bit (8080) CP/M programs under MS-DOS on any MS-DOS compatible computer, and does not require an 8-bit processor. ZEMulator is a program that provides most of the Z-100's keyboard functions, escape code functions, and block graphic (H19) characters on any IBM-PC compatible computer, including the Z-150 and Z-160, and does not require the Z-319 video card. Together these programs add greatly to the number of programs that you can run on your Z-150 or other PC compatible computer.

Requirements: CP/EMulator II requires any version of ZDOS or MS-DOS 1.25 or higher on any Heath/Zenith or other MS-DOS compatible computer. The computer should have at least 128K of

RAM. ZEMulator requires an H/Z-150, H/Z-160, or any IBM-PC compatible computer, 128k (min) of RAM, and MS-DOS/PC-DOS 2.x or 1.25.

The following files are included on the CP/EMulator II disk:

README	.DOC	ZEMI	.COM
CPM2	.COM	ZEM	.DOC
CPM2	.DOC	ZEM	.ASM
CPM2	.ASM	SETZEM1	.COM
COMD	.SYS	SETZEM2	.COM
COMD	.ASM	SETZEMI	.COM
ZEM1	.COM	SETZEM	.ASM
ZEM2	.COM	ZTEST	

Program Authors:

CPM2 -- P. Swayne, HUG. Developed from the first CP/EMulator by Robert A. Metz.
COMD -- Developed from ZCPR by P. Swayne, HUG
ZEM -- Original program by Robert Metz. Enhanced and modified by P. Swayne, HUG.
SETZEM -- P. SWAYNE, HUG

CPM2 -- This is the CP/EMulator II program. It supports nearly all BDOS calls of real CP/M version 2.2 and non-disk BIOS calls. The Set DMA and Select Disk BIOS calls are also supported. It will run most CP/M programs including MBASIC and editors. It can be used in two modes: a direct mode that allows you to run a CP/M program directly from the MS-DOS prompt, and a command mode that provides a CP/M-like environment with the regular CP/M commands such as DIR, SAVE, etc.

CP/EMulator II interprets the 8080 code in the CP/M program and runs everything using the 16-bit processor. It, therefore, allows you to run 8-bit programs on Z-150 or ET-100 computers which have no 8-bit processor. It offers Z-150 owners an alternative to purchasing an expensive CP/M board from a non-Heath manufacturer in order to run CP/M programs. Note: H/Z-100 owners should use the original CP/EMulator program (885-3007-37), instead of this program to run CP/M programs under Z-DOS/MS-DOS.

CP/EMulator II offers the following advantages:

1. You can put your favorite programs, whether they are CP/M or MS-DOS, on a single disk.

2. CP/EMulator II provides a very large 63.5k TPA. Your programs will have all the memory space they need.

3. CP/EMulator II allows you to use CP/M editors on MS-DOS files.

Some disadvantages of CP/EMulator II are:

1. Because the 8080 code in the CP/M program is interpreted, it will run slower than under a real 8-bit processor. Some programs, such as CP/M WordStar, are slowed down so much as to be impractical. With others, the slow-down is hardly noticed.

2. Extension programs, such as DESPOOL or the CP/M version of KEYMAP, will not work except under rare circumstances.

3. Programs that use BIOS disk routines (except set DMA and select disk) will not work.

4. No TIC counter is supported. Game programs that require the TIC counter provided on some Heath/Zenith computers will not work.

COMD -- This program is a system command processor for CP/EMulator II that provides the usual CP/M commands such as DIR, ERA, SAVE, etc.

ZEM1, ZEM2, ZEMI -- These are versions of the ZEMulator program. ZEM1 and ZEM2 are for H/Z-150 and H/Z-160 computers only, and provide the full H19 graphic character set along with Z-100 escape codes and key codes. The two versions cover variances in video configurations that have been encountered. ZEMI is for use with any PC-compatible computer, and provides a partial H19 graphic character set. That is, some of the characters are not the correct shape, and all of them are derived from the IBM-PC block graphic character set. ZEMI provides the same keyboard and escape code capability as the other versions.

Zemulator allows you to run many programs that were written for the H/Z-100 on your H/Z-150 or IBM-PC. These programs include editors such as HUG's Fast Eddy or Z-100 WordStar version 3.21 (note: Z-100 WordStar version 3.3 requires a Z-319 card), and any CP/M program you might want to run under CP/EMulator II that requires H19 graphics or the function keys. ZEMulator supports both expanded output (escape codes) and the non-expanded mode (8-bit codes) of the Z-100 keyboard. Note: ZEMulator does not require the installation of a Z-319 video card to operate. However, Z-100 programs that utilize video memory directly, whether for graphics or text, will not work without the Z-319 card.

SETZEM1, SETZEM2, SETZEMI -- This program is used to disable and re-enable ZEMulator once it has been installed. SETZEM1 and SETZEM2 are for the H/Z-150 versions, and SETZEMI is for the PC-compatible version.

ZTEST -- This is a text file that lets you determine which of ZEM1 or ZEM2 is correct for your H/Z-150 or H/Z-160.

TABLE C Rating: (0), (3), (9)

885-8032-[37] HDOS

CASTLE $20.00

Introduction: Now, play a version of Dungeons And Dragons (tm) in real-time! CASTLE is a real-time graphics action game for the H-89 and H-8/H-19 computers. It's cousins, CASTLE4, MCASTLE, and MCASTLE4 are designed to be used with an H-89 having the modifications for music and/or 4 MHz CPU speed. The object of the game is to travel through the rooms of a castle and take its treasures, all while avoiding the creatures which inhabit each room.

Requirements: CASTLE requires an H-89 or H-8 with an H-19, 16k of memory, HDOS 2.0, and one disk drive. An H-89 with 4 MHz and music conversions are necessary to benefit from CASTLE4, MCASTLE, and MCASTLE4.

The following is a list of files on the HUG P/N 885-8032-[37] HDOS CASTLE disk:

CASTLE	.ABS	MCASTLE	.ASM
MCASTLE	.ABS	MUSGEN	.ACM
CASTLE4	.ABS	MUSGEN4	.ACM
MCASTLE4	.ABS	README	.DOC
INSTRUCT	.DAT		

Author: Vincent H. Lee

Program Content:

The Castle:

The castle consists of three floors of ten rooms each. The bottom floor is the "easiest" in which to play. The second and third floors have lava pools which cannot be touched. The third floor features one-way doors.

The Player:

The player may move his man up, down, left, right, or diagonally using the keypad of the H-19 (H-89). When he touches a door, he exits the room. When he touches a treasure, he captures it. When he touches a creature or lava pool, however, he turns into a skeleton and dies. The player may fire any arrows he may have in the direction he is moving. Two arrows may be on the screen at any one time.

Treasures:

In the rooms are four kinds of treasures for the player to take.

- Pots Of Gold — Worth 500 points each
- Silver Chalices — Worth 250 points each
- Sheaths Of Arrows — Worth 100 points each plus 10 extra arrows for defense
- Pass Keys — Worth 100 points and unlocks doors

When the player enters a room with a key, he must capture it in order to leave that room.

Creatures:

Four kinds of creatures inhabit the rooms:

- Gimmlings — Gimmlings are large creatures which move only vertically and horizontally
- Archods — Archods resemble large spiders and can move vertically, horizontally, or diagonally
- Snakes — Snakes are smaller creatures which move only diagonally
- Fireballs — Fireballs, like snakes, move only diagonally, but are much smaller

Summary: The object of CASTLE is to take all the treasures from the castle while avoiding and shooting the creatures.

Comments: Here's a real-time action game that will keep you busy for hours!

TABLE C Rating: (0), (1), (2), (10)

HUG Price List

Part Number	Decription of Product	Selling Price	Vol. - Issue

HDOS HARDCOPY SOFTWARE

885-1008	Volume I Documentation	9.00	
885-1013	Volume II Documentation	12.00	
885-1015	Volume III Documentation	9.00	
885-1037	Volume IV Documentation	12.00	8
885-1058	Volume V Documentation	12.00	

MISCELLANEOUS HDOS COLLECTIONS

885-1032	Disk V H8/89	18.00	8
885-1044-[37]	Disk VI H8/89	18.00	
885-1064-[37]	Disk IX H8/89 Disk	18.00	
885-1066-[37]	Disk X H8/89	18.00	10
885-1069	Disk XIII Misc H8/89	18.00	

GAMES

HDOS

885-1010	Adventure Disk H8/89	10.00	4
885-1029-[37]	Disk II Games 1 H8/89	18.00	8
885-1030-[37]	Disk III Games 2 H8/89	18.00	8
885-1031	Disk IV MUSIC H8 Only	20.00	25
885-1067-[37]	Disk XI H8/19/89 Games	18.00	12
885-1068	Disk XII MBASIC Graphic Games	18.00	10
885-1088-[37]	Disk XVII MBASIC Graphic Games	20.00	14
885-1093-[37]	D$D H8/89 Disk	20.00	16
885-1096-[37]	MBASIC Action Games H8/89	20.00	18
885-1103	Sea Battle HDOS H19/8/89	20.00	20
885-1111-[37]	HDOS MBASIC Games H8/89	20.00	23
885-1112-[37]	HDOS Graphic Games H8/89	20.00	23
885-1113-[37]	HDOS Action Games H8/89	20.00	23
885-1114	H8 Color Raiders $ Goop	20.00	23
885-1124	HUGMAN $ Movie Animation Pkg	20.00	41
885-1125	MAZEMADNESS	20.00	41
885-1130	Star Battle	20.00	47
885-8009-[37]	HDOS $ CP/M Galactic Warrior	20.00	32
885-8022	HDOS SHAPES	16.00	45
885-8026	HDOS Space Drop	16.00	49

CP/M

885-1206-[37]	CP/M Games Disk	20.00	11
885-1209-[37]	CP/M MBASIC D$D	20.00	19
885-1211-[37]	CP/M Seabattle	20.00	20
885-1220-[37]	CP/M Action Games	20.00	32
885-1222-[37]	CP/M Adventure	10.00	35
885-1227-[37]	CP/M Cassino Gamess	20.00	38
885-1228-[37]	CP/M Fast Action Games	20.00	39
885-1236-[37]	CP/M Fun Disk I	20.00	55

ZDOS

885-3004-37	ZDOS ZBASIC Graphic Games	20.00	37
885-3009-37	ZDOS ZBASIC D$D	20.00	50
885-3011-37	ZDOS ZBASIC Games Disk	20.00	52
885-3016-37	ZDOS/MSDOS Adventure	10.00	57
885-3017-37	ZDOS Contest Games Disk	25.00	58

UTILITIES

HDOS

885-1022-[37]	HUG Editor (ED) Disk H8/89	20.00	20
885-1025	Runoff Disk H8/89	35.00	
885-1060-[37]	Disk VII H8/89	18.00	
885-1061	TMI Load H8 ONLY Disk	18.00	
885-1062-[37]	Disk VIII H8/89 (2 Disks)	25.00	
885-1063	Floating Point Disk H8/89	18.00	
885-1065	Fix Point Package H8/89 Disk	18.00	10
885-1075	HDOS Support Package H8/89	60.00	
885-1077	TXTCON/BASCON H8/89	18.00	
885-1079-[37]	HDOS Page Editor	25.00	15
885-1080	EDITX H8/H19/H89 Disk	20.00	
885-1082	Programs for Printers H8/89	20.00	
885-1083-[37]	Disk XVI Misc H8/89	20.00	11
885-1089-[37]	Disk XVIII Misc H8/89	20.00	20
885-1090-[37]	Disk XIX Utilities H8/89	20.00	22
885-1092-[37]	Relocating Debug Tool H8/89	30.00	14
885-1098	H8 Color Graphics ASM	20.00	19
885-1099	H8 Color Graphics Tiny PASCAL	20.00	19
885-1105	HDOS Device Drivers H8/89	20.00	24
885-1116	HDOS Z80 Debugging Tool	20.00	27
885-1119-[37]	BHBASIC Support	20.00	29
885-1120-[37]	HDOS 'WHEW' Utilities	20.00	33
885-1121	HDOS Hard Sec Sup Pkg 2 disks	30.00	37
885-1123	XMET Robot $ Cross Assembler	20.00	40
885-1126	HDOS Utilities by PS:	20.00	42
885-1127-[37]	HDOS Soft Sector Support Pkg	30.00	45
885-1128-[37]	HDOS DISKVIEW	16.00	46
885-1129-[37]	HDOS CVT Color Video Terminal	20.00	46
885-8001	SE (Screen Editor)	25.00	28
885-8003	BHTOMB	25.00	28
885-8004	UDUMP	35.00	28
885-8006	HDOS SUBMIT	20.00	31
885-8007	EZITRANS.	30.00	30
885-8015	HDOS TEXTSET Formatter	30.00	42
885-8017	HDOS Programmers Helper	16.00	42
885-8024	HDOS BHBASIC Utilities Disk	16.00	46

CP/M

885-1210-[37]	CP/M ED (same as 885-1022)	20.00	20
885-1212-[37]	CP/M Utilities H8/89	20.00	21
885-1213-[37]	CP/M Disk Utilities H8/89	20.00	22
885-1217-[37]	HUG Disk Duplication Utilities	20.00	26
885-1223-[37	]HRUN HDOS Emulator 3 disks	40.00	37
885-1225-[37]	CP/M Disk Dump $ Edit Utility	30.00	40
885-1226-[37]	CP/M Utilities by PS:	20.00	40
885-1229-[37]	XMET Robot $ Cross Assembler	20.00	40
885-1230-[37]	CP/M Function Key Mapper	20.00	42
885-1231-[37]	Cross Ref Utilities for MBASIC	20.00	43
885-1232-[37]	CP/M Color Video Terminal	20.00	46
885-1235-37	CP/M RDZDOS	20.00	54
885-1237-[37]	CP/M Utilities	20.00	55
885-5001-37	CP/M 86 KEYMAP	20.00	51
885-5002-37	CP/M 86 HUG Editor	20.00	52
885-5003-37	CP/M 86 Utilities by PS:	20.00	54
885-8018-[37]	CP/M FAST EDDY $ BIG EDDY	20.00	43
885-8019-[37]	DOCUMAT and DOCULIST	20.00	43
885-8025-37	CP/M 85/86 FAST EDDY	20.00	49

ZDOS

885-3005-37	ZDOS ETCHDUMP	20.00	39
885-3007-37	ZDOS CP/EMulator	20.00	47
885-3008-37	ZDOS Utilities	20.00	47
885-3010-37	ZDOS KEYMAP	20.00	51
885-3012-37	ZDOS HUG Editor	20.00	52
885-3014-37	ZDOS/MSDOS Utilities II	20.00	54
885-8029-37	ZDOS FAST EDDY	20.00	53

PROGRAMMING LANGUAGES

HDOS

885-1038-[37]	Wise on Disk H8/89	18.00	
885-1042-[37]	PILOT on Disk H8/89	19.00	
885-1059	FOCAL-8 H8/89 DISK	25.00	13
885-1078-[37]	HDOS Z80 Assembler	25.00	21
885-1085	PILOT Documentation	9.00	
885-1086-[37]	Tiny HDOS Pascal H8/89	20.00	13
885-1094	HDOS Fig-Forth H8/89 2 Disks	40.00	18

CPM

885-1208-[37]	CP/M Fig-Forth H8/89 2 Disks	40.00	18
885-1215-[37]	CP/M BASIC-E	20.00	26

BUSINESS, FINANCE AND EDUCATION

HDOS

885-1047	Stocks H8/89 Disk	18.00	
885-1048	Personal Account H8/89 Disk	18.00	
885-1049	Income Tax Records H8/89 Disk	18.00	
885-1055-[37]	MBASIC Inventory Disk H8/89	30.00	
885-1056	MBASIC Mail List	30.00	
885-1070	Disk XIV Home Fin H8/89	18.00	
885-1071-[37]	MBASIC SmBusPk H8/H19/H89	75.00	17
885-1091-[37]	Grade/Score Keeping H8/89	30.00	14
885-1097-[37]	MBASIC Quiz Disk H8/89	20.00	18
885-1118-[37]	MBASIC Payroll	60.00	30
885-1131-[37]	HDOS CHEAPCALC	20.00	47
885-8010	HDOS CHECKOFF	25.00	32
885-8021	HDOS Student's Statistics Pkg	20.00	44
885-8027	HDOS SCICALC	20.00	50

CP/M

885-1218-[37]	CP/M MBASIC Payroll	60.00	31
885-1233-[37]	CP/M CHEAPCALC	20.00	47
885-1239-[37]	Spread Sht. Contest Disk I	20.00	
885-1240-[37]	Spread Sht. Contest Disk II	20.00	
885-1241-[37]	Spread Sht. Contest Disk III	20.00	
885-1242-[37]	Spread Sht. Contest Disk IV	20.00	
885-1243-[37]	Spread Sht. Contest Disk V	20.00	
885-1244-[37]	Spread Sht. Contest Disk VI	20.00	
885-8011-[37]	CP/M CHECKOFF	25.00	32

ZDOS

885-3006-37	ZDOS CHEAPCALC	20.00	47
885-3013-37	ZDOS Checkbook Manager	20.00	54
885-3018-37	ZDOS Contest Spreadsheet Disk	25.00	58
885-8028-37	ZDOS SCICALC	20.00	50
885-8030-37	ZDOS MATHFLASH	20.00	55

DATA BASE MANAGEMENT SYSTEMS

HDOS

885-1107-[37]	HDOS Data Base System H8/89	30.00	23
885-1108-[37]	HDOS MBASIC Data Base Sys.	30.00	23
885-1109-[37]	HDOS Retriever ASM (3 disks)	40.00	23
885-1110	HDOS Autofile (2 disks)	30.00	23
885-1115-[37]	HDOS Navigational Program	20.00	25
885-8008	Farm Accounting System	45.00	30

CP/M

885-1219-[37]	CP/M Navigational Program	20.00	31

AMATEUR RADIO

HDOS

885-8016	Morse Code Transceiver Ver 2.0	20.00	42

CP/M

885-1214-[37]	CP/M MBASIC Log Book (64k)	30.00	23
885-1234-[37]	CP/M Ham Help	20.00	49
885-1238-[37]	CP/M ASCRITY	20.00	57
885-8020-[37]	CP/M RF Comp. Aided Design	30.00	44
885-8031-[37]	CP/M Morse Code Transceiver	20.00	57

COMMUNICATION

HDOS

885-1122-[37]	HDOS MicroNET Connection	16.00	37

CP/M

885-1207-[37]	CP/M TERM $ HTOC	20.00	26
885-1224-[37]	CP/M MicroNET Connection	16.00	37
885-3003-[37]	CP/M ZTERM (Z100 Modem Pkg)	20.00	34
885-5004-37	CPM86 TERM86 and DSKED	20.00	56
885-8005	MAPLE (Modem Appl. Effector)	35.00	29
885-8012-[37]	CP/M MAPLE (Modem Program)	35.00	34
885-8023-37	CP/M 85 MAPLE	35.00	45

MISCELLANEOUS

885-0004	HUG Binder	5.75	
885-1221-[37]	Watzman ROM Source Code/Doc	30.00	33
885-4001	REMark Volumes 1 to 13	20.00	
885-4002	REMark Volumes 14 to 23	20.00	
885-4003	REMark Volume III Issues 24-35	20.00	
885-4004	REMark Vol 4 Issues 36-47	20.00	
885-4500	HUG Software Catalog	9.75	
885-4600	Watzman/HUG ROM	45.00	41
885-4700	HUG Bulletin Board Handbook	5.00	50
885-3015-37	ZDOS SKYVIEWS	20.00	55

NOTE: The [-37] means the product is available in hard sector or soft sector. Remember, when ordering the soft sectored format, you must include the "-37" after the part number; e.g. 885-1223-37.

Some Fixes For MS-DOS 2.13

Pat Swayne
Software Engineer

Users of MS-DOS version 2.13 for Z-100 (not Z-150) computers have experienced problems in a couple of areas. One problem is that output to the PRN or AUX devices is very slow. This is particularly noticeable when you are dumping screen graphics to a printer, or when your printer has a large input buffer. Another problem is that during some serial I/O operations, characters are occasionally lost. Patches for both of these problems appeared in BUSS, the Independent Newsletter of Heath Co. Computers, and are published here with the permission of BUSS. (BUSS is published by Sextant Publishing Co., 716 E Street, S. E., Washington, DC 20003.)

Unfortunately, the patches are not easy to implement. The author of the patch to correct slow printing suggested that the patch be made to memory, and offered a program that you could buy that would automatically apply the patch upon boot-up. The author of the other patch suggested that you use DEBUG to access specific sectors of the disk directly to make the patch. Neither of these approaches is entirely satisfactory, and either could potentially cause problems. Both of the patches are actually done to the part of the operating system called IO.SYS, which is a file on the disk that is normally unaccessible to the user. It would be nice if you could just load in IO.SYS with DEBUG, make the patches, and write it back to the disk again.

The small program in the listing following this article is a utility that makes it possible to access IO.SYS or any other unaccessible file. It does this by allowing you to alter the attributes of the file that make it unaccessible. File attributes are special markers in the directory entry for the file. They can cause the file to not appear in the directory listing and not be recognized by programs such as DEBUG.

If you type in the assembly code after this article as ATTRIB.ASM, you can assemble it with the following commands:

```
A>MASM ATTRIB;
A>LINK ATTRIB;
A>ERASE ATTRIB.OBJ
A>EXE2BIN ATTRIB ATTRIB.COM
A>ERASE ATTRIB.EXE
```

This example assumes that you have MASM, LINK, EXE2BIN, and ATTRIB.ASM on your system or logged disk. If you are using the EXE2BIN from Z-DOS version 1, the command syntax for it is

```
A>EXE2BIN ATTRIB.EXE.COM
```

The resulting program, ATTRIB.COM, can be used to alter the attributes of a file. If you enter

```
A>ATTRIB <filename>
```

where <filename> is any file descriptor (including wild cards), ATTRIB will remove all attributes from the file(s) specified. If you enter

```
A> ATTRIB <filename> <attributes>
```

the attributes specified will be set, and any that are already set, but not specified, will be removed. The attributes can be specified in any order. If you run the program without any argument at all, it will explain the attributes. Notes: ATTRIB is for use only with MS-DOS version 2. The Read Only attribute protects a file from being erased, but not from being written over.

ATTRIB can be used along with DEBUG to apply the following patches to IO.SYS. The commands and responses are shown below as they will appear on your screen, with some additional comments in parentheses.

```
A>ATTRIB IO.SYS
```

(Remove attributes from IO.SYS)

```
A>DEBUG
-NIO.SYS
-L
-U20E0
10D6:20E0 2E            CS:
10D6:20E1 803D00        CMP     BYTE PTR [DI],00
10D6:20E4 F9            STC
10D6:20E5 B80500        MOV     AX,0005
```

(Examine old code at the patch area. It should look like this. Only the first 4 lines are shown here, but more will appear on your screen. The segment address, shown here as 10D6, may be different on your system.)

```
-A20E0
10D6:20E0 CMP AX,5
10D6:20E3 NOP
10D6:20E4           (hit RETURN)
-A20E8
10D6:20E8 JZ 20F3
10D6:20EA NOP
10D6:20EB NOP
10D6:20EC NOP
10D6:20ED NOP
10D6:20EE NOP
10D6:20EF NOP
10D6:20F0 NOP
10D6:20F1 NOP
10D6:20F2           (hit RETURN)
```

(The above cures slow printing. The next patch is for the serial I/O problem.)

```
-U1F32
10D6:1F32 8AE0          MOV     AH,AL
10D6:1F34 247F          AND     AL,7F
10D6:1F36 3C13          CMP     AL,13
10D6:1F38 7502          JNZ     1F3C
```

(Again, only four lines are shown here.)

```
-A1F32
10D6:1F32 MOV BH,AL
10D6:1F34
-A1F47
```

Listing of ATTRIB.ASM

```
        TITLE   FILE ATTRIBUTE ALTERATION PROGRAM
        PAGE    ,132
;       FILE ATTRIBUTE ALTERATION PROGRAM
;
;       THIS PROGRAM ALTERS THE ATTRIBUTES ON A FILE
;       TO USE IT, ENTER
;
;       ATTRIB <filename> <attributes>
;
;       WHERE <filename> IS A VALID MS-DOS FILE DESCRIPTOR,
;       AND <attributes> IS ONE OR MORE ATTRIBUTES DESIGNATED
;       AS FOLLOWS
;
;       R       READ ONLY
;       H       HIDDEN FILE
;       S       SYSTEM FILE
;       B       MARKED FOR BACKUP
;
;       ATTRIBUTES NOT SPECIFIED ARE REMOVED FROM THE FILE,
;       IF THEY WERE SET PREVIOUSLY

ATTR    SEGMENT
        ASSUME  CS:ATTR,DS:ATTR,ES:ATTR,SS:ATTR

        ORG     5CH
DFCB:                                   ;DEFAULT FCB
        ORG     80H
CMDBUF:                                 ;COMMAND BUFFER
        ORG     100H

START:  MOV     AH,19H
        INT     21H                     ;GET DEFAULT DISK
        INC     AL                      ;MAKE IT 1-N
        MOV     BYTE PTR DEFDRV,AL      ;SAVE IT
        CLD                             ;ENSURE FORWARD DIRECTION
        MOV     SI,OFFSET CMDBUF        ;POINT TO COMMAND BUFFER
        LODSB                           ;GET CHARACTER COUNT
        OR      AL,AL                   ;ANY ENTRY
        JNZ     GOTARG                  ;YES
EXPL:   MOV     DX,OFFSET EXMSG
        MOV     AH,9
        INT     21H                     ;ELSE, EXPLAIN PROGRAM
        INT     20H                     ;AND EXIT

;   GET ATTRIBUTES FROM COMMAND LINE

GOTARG: CALL    SOS                     ;SKIP SPACES
FATTR:  LODSB                           ;GET NEXT CHAR
        CMP     AL,' '                  ;SPACE
        JZ      FOUND                   ;FOUND END OF FILE NAME
        CMP     AL,13                   ;CR ALREADY?
        JNZ     FATTR                   ;NO
        JMP     SHORT SETATTS           ;ELSE, REMOVE ATTRIBUTES ONLY
FOUND:  CALL    SOS                     ;SKIP SPACES
GETLP:  MOV     DI,OFFSET ATTBL         ;POINT TO ATTRIBUTE TABLE
        CALL    MUC                     ;MAKE UPPER CASE
        MOV     CL,1                    ;SET FIRST ATTRIBUTE IN CL
TSTATT: CMP     AL,[DI]                 ;COMPARE WITH TABLE ENTRY
        JZ      GOTFLG                  ;GOT A MATCH
        SHL     CL,1                    ;MOVE BIT IN CL TO NEXT ATT.
        CMP     CL,40H                  ;ALL LEGAL PLACES USED?
        JZ      BADATT                  ;YES, BAD ATTRIBUTE
        INC     DI                      ;MOVE TO NEXT TABLE ENTRY
        JMP     SHORT TSTATT
GOTFLG: OR      BYTE PTR ATTRS,CL       ;ADD BIT TO OTHER ATTRIBUTES
        LODSB                           ;GET NEXT CHARACTER
        CMP     AL,13                   ;END OF ENTRY?
        JZ      SETATTS                 ;YES, GO SET ATTRIBUTES
        JMP     SHORT GETLP             ;ELSE, LOOP UNTIL DONE
BADATT: MOV     DX,OFFSET BADMSG
        MOV     AH,09
        INT     21H                     ;SAY "BAD ATTRIBUTE"
        JMP     SHORT EXPL              ;AND EXPLAIN PROGRAM

;       SET ATTRIBUTES ON FILE(S)

SETATTS:MOV     SI,OFFSET DFCB          ;POINT TO ENTRY FCB
        MOV     DI,OFFSET FCB1          ;PUT FILE NAME HERE
        MOV     CX,37                   ;MOVE 37 CHARACTERS
        REP     MOVSB                   ;MOVE THE ENTRY
        MOV     DX,OFFSET XFCB2
        MOV     AH,1AH
        INT     21H                     ;SET DTA ADDRESS TO OUR 2ND FCB
        MOV     AH,11H                  ;SEARCH FOR FIRST DIR. ENTRY
SRCHLP: MOV     DX,OFFSET XFCB1
        INT     21H                     ;LOOK FOR USER'S ENTRY
        OR      AL,AL                   ;GOOD ENTRY?
```

```
1DD6:1F47 MOV AL,BH
1DD6:1F49 POP BX
1DD6:1F4A
-A1F91
1DD6:1F91 MOV BH,AL
1DD6:1F93
-A1FA6
1DD6:1FA6 MOV AL,BH
1DD6:1FA8 POP BX
1DD6:1FA9
-W                      (Write file to disk)
Writing 38C9 bytes
-Q
```

A>ATTRIB IO.SYS RHSB

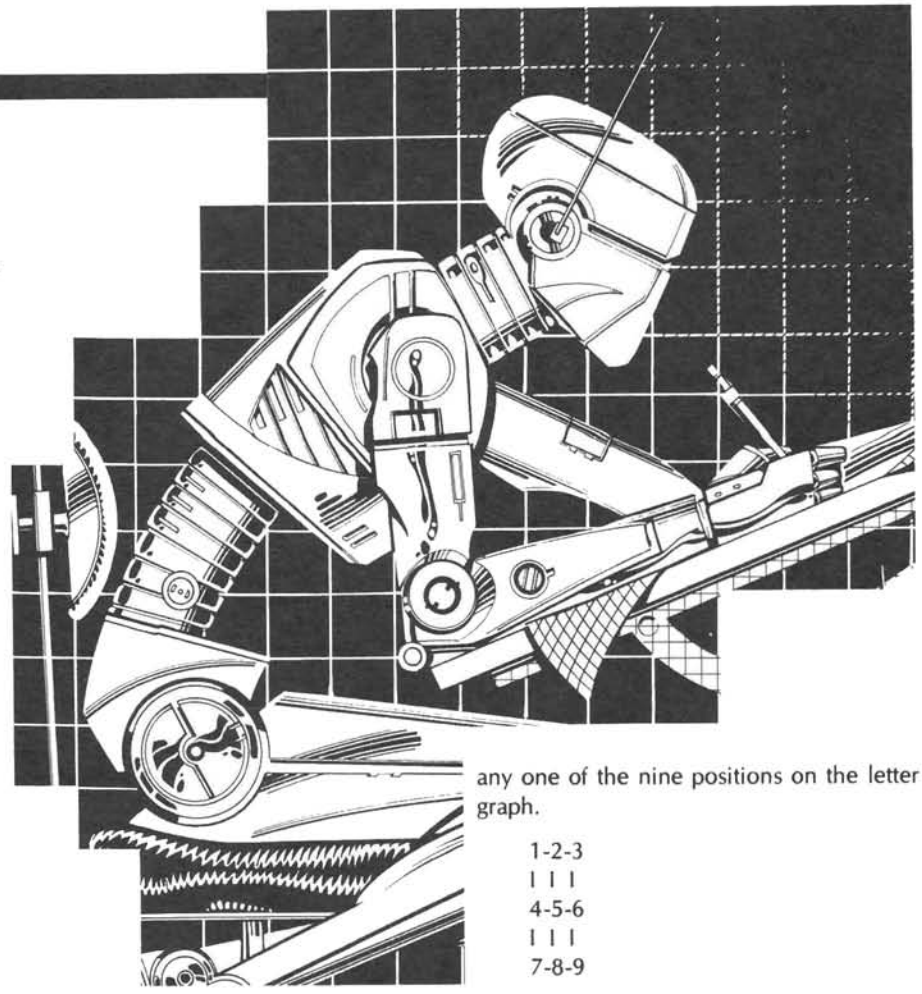
(Replace all attributes on IO.SYS. It normally has all of them set.)

After you make these patches, you can use FORMAT/S or SYS to transfer the patched system to another disk. Do not use an essential system disk when you make the patch, in case something goes wrong. If something does go wrong, you can just erase IO.SYS (with the attributes removed), remove the attributes from the IO.SYS on another system disk, and copy it over to the first disk. Be sure to reset all attributes on any IO.SYS that you remove them from.

Readers of BUSS may notice that the addresses given here are 100H higher than the addresses of the patches in BUSS. That is because the DEBUG program loads in files starting at an offset of 100H.

```
        JZ      GOTFIL              ;YES
        CMP     WORD PTR FDCNT,0    ;ANY FILES DONE?
        JNZ     EXIT                ;IF SO, NO ERROR
        MOV     DX,OFFSET NFMSG
        MOV     AH,09
        INT     21H                 ;ELSE, SAY "FILE NOT FOUND"
EXIT:   INT     20H
GOTFIL: CALL    XNAME               ;EXTRACT FILE NAME FROM FCB
        MOV     DX,OFFSET XDRV      ;POINT TO EXTRACTED NAME
        MOV     AH,43H
        MOV     AL,01
        MOV     CX,WORD PTR ATTRS
        INT     21H                 ;SET ATTRIBUTES ON FILE
        JNB     GDFLG               ;GO ON IF OPERATION OK
        MOV     DX,OFFSET NCMSG
        MOV     AH,9
        INT     21H                 ;SAY "CAN'T CHANGE ATTRIBUTES"
        MOV     SI,OFFSET XDRV      ;POINT TO EXTRACTED FILE NAME
PNAME:  LODSB                       ;GET A CHARACTER
        OR      AL,AL               ;END?
        JZ      GDFLG               ;YES
        MOV     DL,AL               ;ELSE, CHAR TO DL
        MOV     AH,2
        INT     21H                 ;PRINT CHAR
        JMP     SHORT PNAME
GDFLG:  INC     WORD PTR FDCNT      ;COUNT THIS FILE
        MOV     AH,12H
        JMP     SHORT SRCHLP        ;TRY FOR ANOTHER ONE

;   EXTRACT ASCII FILE NAME FROM SEARCH FCB

XNAME:  MOV     SI,OFFSET FCB2FN    ;POINT TO FILE NAME
        MOV     DI,OFFSET XNAM      ;PUT IT HERE
        MOV     CX,8                ;CHARS IN NAME
MOVNAM: LODSB                       ;GET A CHAR
        CMP     AL,' '              ;SPACE?
        JZ      GOTNAM              ;IF SO, END OF NAME
        STOSB                       ;ELSE, STORE CHAR
        LOOP    MOVNAM
GOTNAM: MOV     AL,'.'              ;PERIOD AFTER NAME
        STOSB
        MOV     CX,3                ;3 CHARS IN EXT.
        MOV     SI,OFFSET FCB2EX    ;GET EXT. FROM HERE
MOVEXT: LODSB
        CMP     AL,' '
        JZ      GOTEXT
        STOSB                       ;STORE IT
        LOOP    MOVEXT
GOTEXT: XOR     AL,AL               ;TERMINATE NAME
        STOSB
        MOV     SI,OFFSET FCB2      ;POINT TO ENTRY DRIVE
        MOV     AL,[SI]
        CMP     AL,0                ;ANY DRIVE GIVEN?
        JZ      NODRV               ;NO, GET DEFAULT DRIVE
INSDRV: ADD     AL,'@'              ;ADD ASCII BIAS
        MOV     XDRV,AL
        MOV     BYTE PTR XDRV+1,':'  ;":" AFTER DRIVE NAME
        JMP     SHORT XNX
NODRV:  MOV     AL,BYTE PTR DEFDRV
        JMP     SHORT INSDRV
XNX:    RET

;   SKIP OVER SPACES

SOS:    LODSB                       ;GET NEXT CHARACTER
        CMP     AL,' '              ;SPACE?
        JZ      SOS                 ;IF SO, SKIP IT
        RET

;   MAP LETTERS TO UPPER CASE

MUC:    CMP     AL,'a'              ;LESS THAN "a"?
        JC      NMAP
        CMP     AL,'z'+1            ;MORE THAN "z"?
        JNC     NMAP
        AND     AL,5FH              ;ELSE MAP TO UPPER CASE
NMAP:   RET

EXMSG   DB      13,10,10,'To use this program, enter',13,10,10
        DB      'ATTRIB <filename> <attributes>',13,10,10
        DB      'Valid attributes are:',13,10
        DB      'R - Read only',13,10
        DB      'H - Hidden file',13,10
        DB      'S - System file',13,10
        DB      'B - Mark file for Backup',13,10,10
        DB      'The filename can be ambiguous.$'
NFMSG   DB      13,10,'ERROR -- File not found.$'
BADMSG  DB      13,10,'ERROR -- Invalid attribute.$'
NCMSG   DB      13,10,'ERROR -- Can',27H,'t change attributes '
        DB      'on file $'

ATTBL   DB      'RHS',80H,80H,'B'
DEFDRV  DB      0
FDCNT   DW      0
ATTRS   DB      0,0
XFCB1   DB      0FFH,0,0,0,0,0,27H
FCB1    DB      37 DUP (0)
XFCB2   DB      0FFH
        DB      6 DUP (0)
FCB2    DB      0
FCB2FN  DB      8 DUP (0)
FCB2EX  DB      28 DUP (0)
XDRV    DB      0,0
XNAM    DB      11 DUP (0)

ATTR    ENDS
        END     START
```

ALPHABOT



Dan and Mark Klunk
USAE Box 41
AFCENT APO NY 09011

No self respecting robot should be illiterate, and HERO is no exception. This program allows HERO to print and speak a sequence of eight letters of the alphabet using a magic marker tied to his gripper with a rubber band. The instructions are easy to follow and the only difficulty is finding a sturdy, level writing surface at shoulder height for HERO. Additionally, HERO moves himself to get in position to write each letter so he must have a smooth surface to travel parallel to the writing surface. In total, four motors are used: the drive motor, to position the robot for each letter; the shoulder motor, to raise and lower the pen to the writing surface; the arm extend motor, to form the vertical strokes of letters; and the wrist pivot motor, to form the horizontal strokes of letters.

We learned some interesting 'programming on the M6800' lessons writing this program. First, the direct mode of addressing allows the use of addresses 0000 to 00FF without having to specify the first address byte (00), thus forming 2 byte instructions. Secondly, if the index register is used (in the case of this program set to hex 0000), jump to subroutines can also be done with a 2 byte instruction. Finally, the unique motor move command, E300-motor move wait abs (index), also works off the index register for a 2 byte command. The final result of all this is that a motor move table (hex address 0040 to 004F) position and write subroutines (hex address 0050 to 00CF) can be set in low memory so that the formation of each letter can now be defined in 16 bytes or less. All the letters can now be put in a large subroutine table (starting at hex address 0200) with each letter starting on a 16 byte boundary, and the jump address for the letter subroutine easily computed by left shifting 4 times the hex letter input code. Similarly, the speech phonemes for each letter are in subroutine table with each entry starting on an 8 byte boundary.

One poor programming practice used, for which we could find no convenient alternative, was modifying in-line code when dealing with the jump address for the large letter table and also incrementing the input table (hex address 00F0 to 00FB). This practice has two consequences. First, the program will not work in read only memory (ROM) unless the modified code is moved outside the ROM, and secondly, modified code must be initialized to the proper beginning values each time the program is run. Hence, the need for subroutine Init (hex address 0180).

Nothing our HERO does makes him quite the HERO with the neighborhood kids as writing and spelling their names.

ALPHABOT Program Description

Program ALPHABOT consists principally of tables and subroutines with only a small portion of in-line logic code. There are three tables. The motor move table (hex address 0040 to 004F) has 2 byte entries which give the SS,XX values for wrist pivot motor left (8868), center (8854) and right (8840); arm extend top (3050), center (3028), and bottom (3000). These then can move the pen to any one of the nine positions on the letter graph.

```
1-2-3
I I I
4-5-6
I I I
7-8-9
```

The last two entries are for the arm pivot (484F) pen up, and (484B) pen down. These last two entries are set by subroutine Pen Position at the beginning of the program. The second table, hex 00F0 to 00FF is reserved for storing the input letter codes. The third table, hex 0400, consists of an 8 byte entry of phoneme values to speak each letter of the alphabet.

There are four sets of subroutines. First, are the position subroutines which move the pen to any of the nine letter graph positions. For example, subroutine position 1 (hex address 0057) starts by lifting the pen (E34C), moving the wrist position left (E340), moving the arm extend to top (E346), dropping the pen to paper (E34E), and finally return (39). Note that these are absolute commands, so if the motor is already properly positioned there is no extraneous motion, so HERO looks like he knows what he is doing. The next set of subroutines is the letter table starting at hex location 0200. Each entry in the table is a subroutine of 16 bytes maximum which use the motor move table and position subroutines to describe a letter. For example, letter "A" starting at hex location 0210 consists of the following commands: call subroutine Position 1 (AD57), write a vertical stroke to position 7 (E34A), call subroutine Position 1 (AD57), write a horizontal stroke to position 3 (E344), write a vertical

stroke to position 9 (E34A), call subroutine Position 4 (AD74), write a horizontal stroke to position 6 (E344) and finally return (39). The third set of subroutines are the half diagonal writing strokes for letters like "M" and "W." These are located at hex location 00B0 for right diagonal and 00C0 for left diagonal. The final set of subroutines are the main program subroutines that are called by the driver. These subroutines along with the driver begin at hex location 0100.

ALPHABOT Run Instructions

1. Load the ALPHABOT program.
2. Initialize HERO using the utility command 31.
3. Check HERO's orientation. The program is set to run with HERO facing front, both body and head; his arm is in the back and rises clockwise to the left; the arm extend in the straight slot. Using the pendant, raise his arm to 9 o'clock exactly horizontal, rotate the wrist motor horizontal and gripper closed.
4. Tie a magic marker in his gripper using a rubber band.
5. Put HERO on a smooth surface so his wheels roll easily and construct a level writing surface on HERO's left side at shoulder level.
6. ALPHABOT has two start addresses:
   a. To precisely set the magic marker on the paper, use the pendant and arm pivot motor, then enter AD0100, go to Step 7.
   b. If the magic marker and paper has been set on a previous run, enter AD0103.
7. HERO will display 00----. Enter up to 8 letter codes from the input table. End with FF. HERO will now print and speak the letter sequence.

Input Table

00-SPACE	08-H	10-P	18-X
01-A	09-I	11-Q	19-Y
02-B	0A-J	12-R	1A-Z
03-C	0B-K	13-S	FF-LAST ENTRY
04-D	0C-L	14-T	
05-E	0D-M	15-U	
06-F	0E-N	16-V	
07-G	0F-O	17-W	

NOTE: If anyone wants a HERO cassette of this program, send $10.00 to:

Mark Klunk
USAE Box 41 AFCENT
APO New York, NY 09011

ALPHABOT Program Listing

```
ADDRESS  0 1 2 3 4 5 6 7 8 9 A B C D E F   REMARKS
 0040    8868885488403050302830000484F484B  BASIC MOTOR POSITIONS
 0050    E34CE340E34639E34CE340E346E34E39   0050-HOME  0057-POS1
 0060    E34CE342E346E34E39E34CE344E34601   0060-POS2  0069-POS3
 0070    E34E3901E34CE340E348E34E39E34C01   0074-POS4  007D-POS5
 0080    E342E348E34E3901E34CE344E348E34E   0088-POS6
 0090    3901E34CE340E34AE34E3901E34CE342   0092-POS7  009C-POS8
 00A0    E34AE34E3901E34CE344E34AE34E3901   00A6-POS9
 00B0    8614D32C02D38C014A26F73901010101   RIGHT HALF DIAGONAL
 00C0    8614D32C02D388014A26F73901010101   LEFT HALF DIAGONAL
 00D0    0101010101010101010101010101010101 NOT USED
 00E0    0101010101010101010101010101010101 NOT USED
 00F0    0101010101010101010101010101010101 RESERVED FOR LETTER TABLE
 0100    BD0190BD0180BD0110BD01403F3A0101   DRIVER--CALLS TO SUBS
 0110    8386FFC60C97F07C01165A2EF8C600D7   INPUT SUB, FF IN LETTER TABLE
 0120    FF8600BDF65BBDF7ADBDF79681FF270D   RECEIVE CHARACTERS TO PRINT
 0130    97F07C01317C00FF96FF81082DE33901   RECEIVE CHARACTERS TO PRINT
 0140    0101860297FD96F081FF272E81102D02   PRINT SUB-GET LETTER
 0150    6CFD4848484897FE0101013FAD5096FD   PREP JUMP ADDRESS
 0160    B7016996FEB7016ABDFDFEBD01A0AD50   GO SUB-LETTER THEN MOVE
 0170    D30C04837C01477E0140390101010101
 0180    86F0B70116B70131B70147CE00003901   INIT SUB-FIX CHANGED CODE
 0190    9601974F8B04974D3901010101010101   PEN POSITION SUBROUTINE
 01A0    B60147B701A796F0484848B701B07204   SPEAK SUBROUTINE
 01B0    003901G101010101010101010101010101 END OF SPEAK SUB

 0200    3901010101010101010101010101010101 WRITE SPACE
 0210    AD57E34AAD57E344E34AAD74E3443901   WRITE A
 0220    BD0210AD92E3443901010101010101     WRITE B
 0230    AD69E340E34AE3443901010101010101   WRITE C
 0240    AD57E34AAD57E344E34AE3403901010101 WRITE D
 0250    AD69E340E34AE344AD74E3423901010101 WRITE E
 0260    AD69E340E34AAD74E342390101010101   WRITE F
 0270    AD69E340E34AE344E348E3423901010101 WRITE G
 0280    AD57E34AAD69E34AAD74E3443901010101 WRITE H
 0290    AD60E34A3901010101010101010101010101 WRITE I
 02A0    AD57E344AD60E34AE340E3483901010101 WRITE J
 02B0    AD57E34AAD60ADC0ADB0390101010101   WRITE K
 02C0    AD57E34AE3443901010101010101010101 WRITE L
 02D0    AD92E346ADB0ADA6E346ADC039010101   WRITE M
 02E0    AD92E346ADB0AD7DADB0E3463901010101 WRITE N
 02F0    AD57E344E34AE340E3463901010101010101 WRITE O
 3000    AD92E346E344E348E3403901010101010101 WRITE P
 0310    AD57E344E348ADC0E340AD7DADB03901   WRITE Q
 0320    AD92E346E344E348E340AD7DADB03901   WRITE R
 0330    AD69E340E348E344E34AE3403901010101 WRITE S
 0340    AD60E34AAD57E3443901010101010101   WRITE T
 0350    AD57E34AE344E3463901010101010101   WRITE U
 0360    AD57E348ADB0AD69E348ADC039010101   WRITE V
 0370    AD7DADC0E346AD7DADB0E3463901010101 WRITE W
 0380    AD57ADB0AD7DADB0AD69ADC0ADC03901   WRITE X
 0390    AD57ADB0AD69ADC0AD7DADC039010101   WRITE Y
 03A0    AD57E344ADC0AD7DADC0E3443901010101 WRITE Z
 0400    03FF0000000000000006212121292903FF SPEAK SPACE A
 0410    0E3C3C3C292903FF1F3C3C3C292903FF   SPEAK B     C
 0420    1E3C3C3C292903FF3C3C3C3C292903FF   SPEAK D     E
 0430    3B3B1D1D030303FF1E1A3C3C292903FF   SPEAK F     G
 0440    0621211B2A1003FF1500090929290 3FF  SPEAK H     I
 0450    1E1A0621212903FF19190621292903FF   SPEAK J     K
 0460    0200001818030 3FF0200000C0C0303FF  SPEAK L     M
 0470    0200000D0D0303FF3535353503030 3FF  SPEAK N     O
 0480    25253C3C292903FF19192236373703FF   SPEAK P     Q
 0490    1515152B2B2B03FF0202000001F1F03FF  SPEAK R     S
 04A0    2A2A3C3C292903FF22363637373703FF   SPEAK T     U
 04B0    0F0F3C3C292903FF1E150E18223637FF   SPEAK V     W
 04C0    0202000 01C1C03FF2D2D1500092903FF  SPEAK X     Y
 04D0    2A1F1F3C3C2903FF0303030303030 3FF  SPEAK Z
```

SIG News

Bill Parrott
7010 Caenen
Shawnee, KS 66215

When the last edition of SIG news hit the stands, we were involved in a programming contest on the HUG SIG. To refresh everyone's memory, one of the members made some remarks alluding to the relative superiority of Pascal over "C." As could be predicted in this type of a situation, several C programmers took note of his opinions and a challenge of sorts followed.

Not one to pass up an opportunity to start a good war, the SYSOPs proposed a contest to settle once and for all the question of which is the better language. The contest pitted Pascal programmers against C programmers in the production of a unique application. The specifications for the contest defined a "Programmers" calculator using the popular Reverse Polish Notation (RPN). We felt that this particular problem would be equally suited for C and Pascal and that no group would have an advantage over the other by virtue of their language of choice.

As an incentive for the combatants in the battle of languages, we offered substantial prizes. The author of the program judged best to conform to the specifications and completed in the most timely manner was to receive his or her choice of any Heath/Zenith software product AND one month free operation on the SIG. The author of the second best entry was to receive his choice of either any Heath/Zenith software product OR one month free operation on the SIG.

The envelope please ...

Our first place entry was written in TURBO Pascal and was submitted by Rick Schaeffer (70120,174). His entry was the first received and included a large number of the "extra-credit" features listed in the program specifications. It did, however, contain one minor bug (which has since been eliminated).

Our second place entry was written in DeSmet C and was submitted by Dale Wilson (72155,1402). His was a more basic version which adhered closely to the program specifications but contained no bugs (at least none we could find).

Both entries were well written and are excellent examples of their respective languages. Rick and Dale have proven many times in the past that they are first rate programmers and their efforts here are indicative of the quality of their work.

This month we are publishing the source code to the winning entry by Rick Schaeffer. TURBO Pascal will be required to compile this program. If you do not have TURBO Pascal, an executable version is available in the SIG database in Section 6 under the name HUGCAL.BIN. The source code is also there under the name HUGCAL.PAS for those who don't care to type it in, but would like to have it.

TURBO Pascal is available from Borland International, 4113 Scotts Valley Drive, Scotts Valley, CA, 95066. (408) 438-8400. The price is $49.95, plus $5.00 shipping and handling ($15.00 for foreign orders). The compiler is available for both CP/M and MS-DOS.

The HUG SIG (Special Interest Group) is an electronic bulletin board which is run on the CompuServe Information Service (CIS). It incorporates, however, much more than a simple message exchange facility. It includes an extensive data base of member contributed software and real-time interaction with other members via the conference feature.

To become a member of the HUG SIG, you must first subscribe to CompuServe. The easiest way to do this is to obtain the MicroNET Connection from HUG (p/n 885-1122[-37] for HDOS, 885-1224[-37] for CP/M) which contains everything you need to join CompuServe and the HUG SIG. The package currently sells for $16.00 which includes your user number, password, and even modem software. It also includes one free hour of time on CompuServe.

```
program hugcalc(input, output);
{
   Written By Frederick C. Schaeffer
              E. 13611 26th Ave.
              Spokane, Wa.  99216
              (509) 928-3533

   This program was written for the Heath User Group
   "Great Compiler Competition" Mostly, it is self
   prompting. Exit of the program which is accomplished
   by hitting Control-X. Changing radix is accomplished
   by using the arrow keys as follows:
      Left arrow  = Binary
      Right arrow = Octal
      Up arrow    = Decimal
      Down arrow  = Hex.
}

type
   numstr = string[40];
   hugstr = string[8];
   bigstr = string[132];
   fktype = string[2];
   xltype = record
      escseq : fktype;
      opcd : hugstr;
   end;
const
   hex = 16.0;
   dec = 10.0;
   oct = 8.0;
   bin = 2.0;
   esc = #27;
   maxops = 32;
   xltbl : array[1..maxops] of xltype =
   (
    (escseq : 'A';opcd : 'DEC'),
    (escseq : 'B';opcd : 'HEX'),
    (escseq : 'C';opcd : 'OCT'),
    (escseq : 'D';opcd : 'BIN'),
    (escseq : 'J';opcd : '+'),
    (escseq : 'S';opcd : '-'),
    (escseq : 'T';opcd : '*'),
    (escseq : 'U';opcd : '/'),
    (escseq : 'V';opcd : 'DIV'),
    (escseq : 'W';opcd : 'MOD'),
    (escseq : 'P';opcd : 'EXX'),
    (escseq : 'Q';opcd : 'CHS'),
    (escseq : 'R';opcd : '!'),
    (escseq : 'OI';opcd : 'ABS'),
    (escseq : 'OJ';opcd : '&'),
    (escseq : 'OK';opcd : '|'),
    (escseq : 'OL';opcd : '↑'),
    (escseq : 'O';opcd : '>>'),
    (escseq : '@';opcd : '>>'),
    (escseq : 'L';opcd : '<<'),
    (escseq : 'E';opcd : '>'),
```

```
(escseq : '1A';opcd : 'POP'),
(escseq : '1B';opcd : 'PUSH'),
(escseq : '1C';opcd : 'K'),
(escseq : '1D';opcd : '=K'),
(escseq : '1E';opcd : 'STO'),
(escseq : '1F';opcd : 'RCL'),
(escseq : '1G';opcd : '@'),
(escseq : '1H';opcd : 'EX'),
(escseq : '1I';opcd : '='),
(escseq : '1J';opcd : 'CLS'),
(escseq : '1K';opcd : 'CLEAR')
);

var
   cstack : array[1..10] of real;
   mem : array[0..10] of real;
   entstr : numstr;
   i : integer;
   ch : char;
   valid_digit : set of char;
   wkreal : real;
   current_radix : real;
   fkstr : fktype;
   opstr : hugstr;
   overflow : boolean;

{
    The following two routines (MyConOut and init_con)
    replace the standard TURBO console handler. This was
    done in order to speed up the screen display of this
    program.  It's real impressive...try removing the
    call to init_con, which is the first instruction in
    mainline, and re_compiling to see just how much
    difference it makes!
}

procedure MyConOut(outch : char);
begin
   inline
      ($8a/$46/$04/              {mov al,outch}
       $fc/                     {cld}
       $9a/$19/$00/$01/$fe);       {long call mtr_scrt}
end;

procedure init_con;
begin
   memw[seg(ConOutPtr):ofs(ConOutPtr)] := ofs(MyConOut);
end;

function curpos(x,y : integer) : hugstr;
begin
   curpos := chr(27) + 'Y' + chr(31+x) + chr(31+y);
end;

procedure display_intro;
begin
   write(chr(27)+'E');
   write('                         ');
   writeln(esc+'p'+'HUG Programmer''s Calculator'+esc+'q');
   write('                    ');
   writeln('by: Frederick C. Schaeffer [70120,174]');
   writeln;
   writeln('        This program was written for the Heath User Group "Great Compiler');
   writeln('        Competition".  All operators described in the specification are');
   writeln('        supported as documented.  Operators are all assigned to function');
   writeln('        keys as displayed on the bottom of the screen and are executed as');
   writeln('        soon as detected.  Radix (base) change is accomplished by use of');
   writeln('        the keypad arrow keys as follows:');
   writeln('            Left  =  Binary');
   writeln('            Right =  Octal');
   writeln('            Up    =  Decimal (The default)');
   writeln('            Down  =  Hex');
   writeln;
   writeln('        Exit is accomplished with a Control-X.');
   writeln;
   writeln;
   write('               Hit any key to enter the calculator...');
   read(kbd,ch);
```

```
end;

function cvtstr(inval : real; in_radix : real) : numstr;
var
   i : integer;
   wkchr : char;
   wkstr : numstr;
   signbit : byte;
begin
   wkstr := '';
   if inval < 0.0 then signbit := 1 else signbit := 0;
   inval := abs(inval);
   if ((in_radix <> 10) and (signbit = 1)) then begin
      inval := 4294967296.0 - inval;
      signbit := 0;
   end;
   repeat
      i := round(frac(inval / in_radix) * in_radix);
      if i <= 9 then wkchr := chr(i + $30) else wkchr := chr(i + 55);
      wkstr := wkchr + wkstr;
      inval := int(inval / in_radix);
   until inval = 0.0;
   if signbit = 1 then wkstr := '-' + wkstr;
   cvtstr := wkstr;
end;

procedure chk_ovflo(var wknum : real);
begin
   overflow := false;
   if wknum > 2147483647.0 then begin
      wknum := 4294967296.0 - wknum;
      wknum := wknum * -1.0;
   end else if wknum < (-2147483648.0) then begin
      wknum := 4294967296.0 + wknum;
   end;
   if (wknum < -2147483648.0) or (wknum > 2147483647.0) then begin
      overflow := true; wknum := -1;
   end;
end;

function cvtint(in_string : numstr; in_radix : real) : real;
var
   wkint : real;
   i : integer;
   bytval : byte;
begin
   wkint := 0.0;
   for i := 1 to ord(in_string[0]) do begin
      case in_string[i] of
         '0'..'9' : bytval := ord(in_string[i]) - $30;
         'A'..'F' : bytval := ord(in_string[i]) - 55;
      end;
      wkint := wkint * in_radix;
      wkint := wkint + bytval;
   end;
   chk_ovflo(wkint);
   cvtint := wkint;
end;

procedure mem_display(regno : integer);
var
   mline : integer;
begin
   mline := (9 - regno) + 6;
   if regno = 10 then mline := 17;
   write(curpos(mline,46)+'                                    ');
   write(curpos(mline,46)+cvtstr(mem[regno], current_radix));
end;

procedure sreg_display(sregno : integer);
var
   mline : integer;
begin
   mline := (10 - sregno) + 6;
   if sregno = 2 then mline := 15;
   if sregno = 1 then mline := 17;
   write(curpos(mline,7)+'                                    ');
   write(curpos(mline,7)+cvtstr(cstack[sregno], current_radix));
```

```
end;

procedure display_allmems;
var
   i : integer;
begin
   for i := 0 to 10 do mem_display(i);
end;

procedure display_stack;
var
   i : integer;
begin
   for i := 1 to 10 do sreg_display(i);
end;

procedure init_screen;
const spc13 = '             ';
begin
   write(chr(27)+'E');
   write('                         ');
   writeln('HUG Programmer''s Calculator');
   write('                         ');
   writeln('  Current Radix is: '+esc+'p'+'DEC'+esc+'q');
   write(esc+'F');
   writeln('iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii');
   writeln('i          RPN STACK CONTENTS             i         MEMORY REGISTER CONTENTS      i');
   writeln('iaaasaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaiaaasaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaai');
   writeln('i 9 i 9 0                                i');
   writeln('i 8 i 8 0                                i');
   writeln('i 7 i 7 0                                i');
   writeln('i 6 i 6 0                                i');
   writeln('i 5 i 5 0                                i');
   writeln('i 4 i 4 0                                i');
   writeln('i 3 i 3 0                                i');
   writeln('i 2 i 2 0                                i');
   writeln('iaaabaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaai 1 0                                    i');
   writeln('i Y i 0 0                                i');
   writeln('iaaabaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaiaaabaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaai');
   writeln('i X i K 1024                             i');
   writeln('iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii');
   write(esc+'G');
   writeln(' New Value: ');
   writeln('   Last Op: ');
   write(esc+'x1'+esc+'F');
   write(esc+'p'+curpos(25,1)); {set reverse on and line 25}
   write(' +  ');
   write(' -  ');
   write(' *  ');
   write(' /  ');
   write('DIV ');
   write('MOD ');
   write('EXX ');
   write('CHS ');
   write(' !  ');
   write('ABS ');
   write(' &  ');
   write(esc+'G'+' |  '+esc+'F'+' ');
   write(esc+'G'+' ↑  '+esc+'F'+' ');
   write(' >> ');
   write(' << ');
   write(curpos(24,1));
   write(' >  ');
   write('POP ');
   write('PUSH');
   write(' K  ');
   write(' =K ');
   write(esc+'G'+'STOn'+esc+'F');
   write(esc+'G'+'RCLn'+esc+'F');
   write(esc+'G'+' @n '+esc+'F');
   write(esc+'G'+'EXn '+esc+'F');
   write(esc+'G'+' =  '+esc+'F');
   write('CLS ');
   write('CLRM');
   write(curpos(23,1)+esc+'q'+esc+'G');
   write(' F0  F1  F2  F3  F4  F5  F6  F7  F8  F9  F10 F11 F12');
   write('  ICHR INSL');
end;
```

```
procedure set_radix(inradix : real);
var
   radixint : integer;
begin
   radixint := trunc(inradix);
   case radixint of
      16 :
         begin
            valid_digit := ['0'..'9','A'..'F'];
            current_radix := hex;
            write(curpos(2,45)+esc+'p'+'HEX'+esc+'q');
         end;
      10 :
         begin
            valid_digit := ['0'..'9'];
            current_radix := dec;
            write(curpos(2,45)+esc+'p'+'DEC'+esc+'q');
         end;
      08 :
         begin
            valid_digit := ['0'..'7'];
            current_radix := oct;
            write(curpos(2,45)+esc+'p'+'OCT'+esc+'q');
         end;
      02 :
         begin
            valid_digit := ['0'..'1'];
            current_radix := bin;
            write(curpos(2,45)+esc+'p'+'BIN'+esc+'q');
         end;
   end;
end;

procedure push(num : real);
var
   i : integer;
begin
   for i := 9 downto 1 do cstack[i+1] := cstack[i];
   cstack[1] := num;
end;

function pop : real;
var
   i : integer;
   holdnum : real;
begin
   holdnum := cstack[1];
   for i := 2 to 10 do cstack[i-1] := cstack[i];
   cstack[10] := 0;
   pop := holdnum;
end;

procedure prompt_user;
begin
   write(curpos(19,13)+'                                    ');
   write(curpos(19,13));
end;

procedure getesc(var fkstr : fktype);
var ch : char;
```

```
else if opstr = '-' then begin
   op2 := pop;
   push(pop - op2);
end

else if opstr = '*' then begin
   push(pop * pop)
end

else if opstr = '/' then begin
   op2 := pop;
   if op2 <> 0 then push(myround(pop / op2))
      else write(curpos(20,13)+^G+'Error...Divide by zero!');
end

else if opstr = 'DIV' then begin
   op2 := pop;
   if op2 <> 0 then push(int(pop / op2))
      else write(curpos(20,13)+^G+'Error...Divide by zero!');
end

else if opstr = 'MOD' then begin
   op2 := pop;
   if op2 <> 0 then push(myround(frac(pop / op2) * op2))
      else write(curpos(20,13)+^G+'Error...Divide by zero!');
end

else if opstr = 'EXX' then begin
   op2 := cstack[1];
   cstack[1] := cstack[2];
   cstack[2] := op2;
end

else if opstr = 'CHS' then
   cstack[1] := cstack[1] * -1

else if opstr = '!' then
   cstack[1] := (cstack[1] + 1) * -1

else if opstr = 'ABS' then
   cstack[1] := abs(cstack[1])

else if opstr = '&' then begin
   op2 := pop;
   op1 := pop;
   mod_32768(op1,op2);
   op1sb := op1sb and op2sb;
   op1hi := op1hi and op2hi;
   op1lo := op1lo and op2lo;
   push((op1sb * 32768.0 + op1hi) * 32768.0 + op1lo);
end

else if opstr = '|' then begin
   op2 := pop;
   op1 := pop;
   mod_32768(op1,op2);
   op1sb := op1sb or op2sb;
   op1hi := op1hi or op2hi;
   op1lo := op1lo or op2lo;
   push((op1sb * 32768.0 + op1hi) * 32768.0 + op1lo);
```

```
begin
   fkstr := '';
   read(kbd,ch);
   if ch in ['?','0','1'] then begin
      fkstr := fkstr + ch;
      read(kbd,ch);
   end;
   fkstr := fkstr + ch;
end;

function xlate_op(var fkstr : fktype; var opstr : hugstr) : boolean;
var
   i : integer;
   match : boolean;
begin
   match := false;
   for i := 1 to maxops do begin
      if xltbl[i].escseq = fkstr then begin
         opstr := xltbl[i].opcd;
         match := true;
      end;
   end;
   xlate_op := match;
end;

function myround(op : real) : real;
begin
   myround := int(op + frac(op));
end;

procedure do_operation(var opstr : hugstr);
var
   op1,op2 : real;
   op1sb,op1hi,op1lo,op2sb,op2hi,op2lo : integer;
   i,j : integer;
   ch : char;
   dop_opstr : hugstr;
   dop_fkstr : fktype;
   holdx : real;
   done : boolean;
   signbit : byte;
   bitstr : numstr;

procedure mod_32768(op1,op2 : real);
begin
   op2lo := round(frac(op2 / 32768.0) * 32768.0);
   op2 := int(op2 / 32768.0);
   op2hi := round(frac(op2 / 32768.0) * 32768.0);
   op2sb := trunc(op2 / 32768.0);
   op1lo := round(frac(op1 / 32768.0) * 32768.0);
   op1 := int(op1 / 32768.0);
   op1hi := round(frac(op1 / 32768.0) * 32768.0);
   op1sb := trunc(op1 / 32768.0);
end;

begin
   if opstr = '+' then begin
      push(pop + pop);
   end

end

else if opstr = '*' then begin
   op2 := pop;
   op1 := pop;
   mod_32768(op1,op2);
   op1sb := op1sb xor op2sb;
   op1hi := op1hi xor op2hi;
   op1lo := op1lo xor op2lo;
   push((op1sb * 32768.0 + op1hi) * 32768.0 + op1lo);
end

else if opstr = '>>' then begin
   op2 := pop;
   op1 := pop;
   bitstr := cvtstr(op2, bin); {get a bit string}
   op1 := abs(op1);
   if op1 > 32.0 then op1 := 32.0;
   for i := 1 to trunc(op1) do begin
      if length(bitstr) = 32 then signbit := 1 else signbit := 0;
      for j := length(bitstr) downto 1 do
         bitstr[j] := bitstr[j-1];
      bitstr[1] := '0';
      j := 1;
      if signbit = 1 then
         while ((j <= length(bitstr)) and (bitstr[j] = '0')) do begin
            bitstr[j] := '1';
            j := j + 1;
         end;
    end;
    push(cvtint(bitstr, bin));
end

 else if opstr = '<<' then begin
    op2 := pop;
    op1 := pop;
    bitstr := cvtstr(op2, bin); {get a bit string}
    op1 := abs(op1);
    if op1 > 32 then op1 := 32;
    for i := 1 to trunc(op1) do begin
       if length(bitstr) = 32 then signbit := 1 else
          begin
             signbit := 0;
             if bitstr[1] = '1' then bitstr := '0'+bitstr;
          end;
       for j := 1 to length(bitstr)-1 do
          bitstr[j] := bitstr[j+1];
       bitstr[length(bitstr)] := '0';
       j := 1;
       if signbit = 1 then
          while ((j <= length(bitstr)) and (bitstr[j] = '0')) do begin
             bitstr[j] := '1';
             j := j + 1;
          end;
    end;
    push(cvtint(bitstr, bin));
 end

 else if opstr = '>' then begin
    op2 := pop;
```

```
   op1 := pop;
   bitstr := cvtstr(op2, bin); {get a bit string}
   op1 := abs(op1);
   if op1 > 32 then op1 := 32;
   for i := 1 to trunc(op1) do begin
      for j := length(bitstr) downto 1 do
         bitstr[j] := bitstr[j-1];
      bitstr[1] := '0';
   end;
   push(cvtint(bitstr, bin));
end

else if opstr = 'POP' then
   op2 := pop

else if opstr = 'PUSH' then
   push(cstack[1])

else if opstr = 'K' then
   push(mem[10])

else if opstr = '=K' then begin
   mem[10] := cstack[1];
   mem_display(10);
end

else if opstr = 'STO' then begin
   repeat
      read(kbd,ch);
      if not (ch in ['0'..'9']) then write(↑G);
   until ch in ['0'..'9'];
   i := ord(ch) - $30;
   mem[i] := cstack[1];
   write(ch);
   mem_display(i);
end

else if opstr = 'RCL' then begin
   repeat
      read(kbd,ch);
      if not (ch in ['0'..'9']) then write(↑G);
   until ch in ['0'..'9'];
   i := ord(ch) - $30;
   push(mem[i]);
   write(ch);
end

else if opstr = '@' then begin
   repeat
      read(kbd,ch);
      if not (ch in ['0'..'9']) then write(↑G);
   until ch in ['0'..'9'];
   i := ord(ch) - $30;
   write(ch);
   holdx := pop;
   push(holdx);
   push(mem[i]);
   done := false;
   repeat
      set_radix(hex);
      display_allmems;
   end

   else if opstr = 'OCT' then begin
      set_radix(oct);
      display_allmems;
   end

   else if opstr = 'DEC' then begin
      set_radix(dec);
      display_allmems;
   end

   else if opstr = 'BIN' then begin
      set_radix(bin);
      display_allmems;
   end

   else writeln(↑G+'BADOP');
   chk_ovflo(cstack[1]);
end;

begin {main}
   init_con; {remove this instruction to use standard console output}
   display_intro;
   init_screen;
   set_radix(dec);
   mem[10] := 1024;
   for i := 0 to 9 do mem[i] := 0.0;
   for i := 1 to 10 do cstack[i] := 0.0;
   display_stack;
   entstr := '';
   prompt_user;
   repeat
      read(kbd,ch);
      ch := upcase(ch);
      if ch = ↑X then ch := ch

      else
      if ch = ↑M then begin
         push(cvtint(entstr, current_radix));
         if overflow then write(curpos(1,72)+↑G+esc+'pOVERFLOW'+esc+'q')
            else write(curpos(1,72)+'        ');
         display_stack;
         entstr := '';
         prompt_user;
      end

      else
      if ch = ↑H then begin
         if entstr <> '' then begin
            entstr[0] := chr(ord(entstr[0]) - 1);
            write(↑H+' '+↑H);
         end;
      end

      else
      if ch in valid_digit then begin
         entstr := entstr + ch;
```

```
      repeat
         read(kbd,ch);
         if ch <> chr(27) then write(↑G);
      until ch = chr(27);
      getesc(dop_fkstr);
      if xlate_op(dop_fkstr, dop_opstr) then begin
         if dop_opstr <> '@' then begin
            write(dop_opstr);
            do_operation(dop_opstr);
            mem[i] := pop;
            mem_display(i);
            push(holdx);
            done := true;
         end;
      end;
      if not done then write(↑G);
   until done;
end

else if opstr = 'EX' then begin
   repeat
      read(kbd,ch);
      if not (ch in ['0'..'9']) then write(↑G);
   until ch in ['0'..'9'];
   i := ord(ch) - $30;
   op2 := pop;
   push(mem[i]);
   mem[i] := op2;
   write(ch);
   mem_display(i);
end

else if opstr = '=' then begin
   write(esc+'y1'+esc+'E');
   write(curpos(10,1));
   writeln('        X register  Binary = '+cvtstr(cstack[1], 2.0));
   writeln('        X register   Octal = '+cvtstr(cstack[1], 8.0));
   writeln('        X register Decimal = '+cvtstr(cstack[1], 10.0));
   writeln('        X register     Hex = '+cvtstr(cstack[1], 16.0));
   writeln; writeln;
   write('        Hit any key to continue...');
   read(kbd,ch);
   init_screen;
   set_radix(current_radix);
   display_allmems;
end

else if opstr = 'CLS' then begin
   for i := 1 to 10 do cstack[i] := 0.0;
end

else if opstr = 'CLEAR' then begin
   for i := 1 to 10 do cstack[i] := 0.0;
   for i := 0 to 9 do mem[i] := 0.0;
   mem[10] := 1024;
   display_allmems;
end

else if opstr = 'HEX' then begin
```

```
      write(ch);
   end

   else
   if ch = chr(27) then begin
      getesc(fkstr);
      if xlate_op(fkstr, opstr) then begin
         if entstr <> '' then begin
            push(cvtint(entstr, current_radix));
            if overflow then write(curpos(1,72)+↑G+esc+'pOVERFLOW'+esc+'q')
               else write(curpos(1,72)+'          ');
            entstr := '';
         end;
         write(curpos(20,13)+'                              ');
         write(curpos(20,13)+opstr);
         do_operation(opstr);
         display_stack;
         prompt_user;
      end else write(↑G)

   end else
      write(↑G);
 until ch = ↑X;
 write(esc+'y1'+esc+'E');
end.
```

The FORTRAN Formula-2

Dick Stanley
P. O. Box 9512
Alexandria, VA 22304

In the first article of this series, we looked at the comparative advantages of FORTRAN as a programming language. In this article, we are going to get comfortable with input/output (I/O) using FORTRAN and the computer console and CRT screen. For some reason, virtually every book written about this language treats I/O as if it were being done on a batch-processing system using punched cards. That's how it all started, but few of us use IBM-format cards to talk to our microcomputers. It can be more than a little confusing trying to translate that world into the one of microcomputers. Fear not! When we're done with this installment, you will be comfortably inputting and outputting data from your machine with FORTRAN and never thinking about punched cards.

First Things First

When you study any language, whether FORTRAN or French, you first need to learn some rules of grammar (the relation between words), syntax (the set of rules governing the grammatical arrangement of words), and semantics (the meanings of the words). You can study these things before trying to use the language, but I find it is easiest to learn in stages. If you can discover enough at an early stage to understand and be understood about little, but meaningful things, it then becomes much easier to learn the more intricate parts of the language that enable you to express yourself more elegantly. We'll try that approach with FORTRAN: we'll learn as we go, and polish our speech as we proceed. With that in mind, here we go!

FORTRAN's alphabet (character set) consists of the 26 letters of the English alphabet, the digits 0-9, the characters $, =, -, +, *, /, (,), blank, comma, and period. It can print any printable character, but only these have any meaning to the language. FORTRAN does not distinguish between upper- and lower-case letters; it converts everything internally to upper-case.

The language's origins in the days of cards and batch-processing show up in the line format of program statements. A FORTRAN statement must follow the following format:

Columns 1-5:	Statement label or number field
Column 6:	Continuation character field
Columns 7-72:	Statement body field
Columns 73-80:	Identification field.

Wow! That looks complicated already, and we've barely begun. Let's examine this a minute. There are 80 columns available, precisely the number in a punched card and also the number of columns on your computer screen. The correspondence is exact. Just put whatever FORTRAN wants to see in column 3 in column 3 on the CRT screen, and so forth.

The identification field is left over from the days of cards, which are VERY easy to drop or otherwise put out of order. Rather than reassemble a 5000-card program from the mess on the floor by comparing each card to the program listing, the programmer would simply insert a number into columns 73-80 when the cards were prepared. The first card was number 1, the second was 2, and so on. That way, when the cards became disordered, they could quickly be put back in order by running them through a card sorter! Very handy, if you're using cards. We won't use this feature much in microcomputing, so you can stop worrying about the identification field.

Working backwards, we find that columns 7-72 are allocated for the statement itself, e.g. A=2. After all, it has to go somewhere, and the computer needs to know where to look for it. Because the identification field begins in column 73, the computer stops reading the statement at column 72. Everything beyond that is not part of the statement. This feature has been retained in microcomputer versions of FORTRAN to maintain compatibility with code written for other machines. Remember, if the statement would extend beyond column 72, the compiler won't see it.

That being the case, there needs to be a way to tell the compiler that this statement is continued on the next line for those statements that won't fit between columns 7 and 72. That's what column 6 is for. When column 6 contains a character other than zero or blank, the compiler reads this line as a continuation of the one above. You may have as many as 19 continuation lines. Together with the 66 characters on the initial line, that allows statements up to 1320 characters in length!

We know from our work with languages, like BASIC, that there must be a way to label statements to enable the program to branch to them or refer to them. That is the function of columns 1-5. Unlike BASIC, FORTRAN does not require that each statement be numbered. When statement numbers are used, they must be integers between 1 and 99999, and each label must be used only once (you can't have two statements numbered 100 -- this hopelessly confuses the computer!). You only need to use labels when they are required for some purpose, however, and they need not be in order: 300 may precede 21, and FORTRAN won't care a bit.

One last thing. If the character in column 1 is the letter "C", that line is a comment. The compiler ignores comments, and they don't slow down program execution (as they do in BASIC, where the interpreter must determine each time it encounters a comment that it is a comment, and then ignore it). As with any programming, the liberal use of comments will help make your code readable by humans. Later versions of FORTRAN (such as FORTRAN-77) also recognize an asterisk (*) in column 1, or a totally blank line, as comments. Microsoft FORTRAN and FORTRAN-66 do not, however. As all FORTRAN versions recognize the "C" in column 1 as beginning a comment line, we will use only that syntax in this series.

A short FORTRAN program segment is shown in Figure 1 that illustrates the layout of statements according to these rules (the ruler line

is, of course, only to illustrate where columns are, and will not be found in the program). This particular program segment would be entered on your keyboard and CRT exactly as it appears. Notice that the mathematical notation is virtually identical to that used in BASIC, and that spaces can be added within the statements to enhance readability.

```
Column   1    1    2    2    3    3    4    4    5    5    6    6    7 7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0.2

C  This line is a comment line used to annotate the program
  100 A=D/3+B
C  The line above is an assignment statement, numbered 100
      VALUE =
     1    (X**2 + Y**2) -
     2    BETA/7
C  The three lines above show an equation continued on 3 program lines
```

Figure 1. FORTRAN program segment to illustrate language syntax.

Getting Things Ready

We now know almost enough to write our first FORTRAN program. To get started, prepare a SYSGEN'ed disk that contains the following files (Please remember, we are using Microsoft FORTRAN-80 as our model. If you have a different FORTRAN, load all the files your documentation says are required for compilation, linking, and loading onto this disk in place of F80, L80, and FORLIB):

* your text editor and any extra files it requires
* F80.COM (the FORTRAN-80 compiler)
* FORLIB.COM (the FORTRAN compiler system library)
* L80.COM (the linker program -- more about this later).

Hereafter, we will assume that this system disk is in Drive 0A:, and the program disk (the one on which we will write the program code) is in Drive 0B:. If you configure your disks differently, remember the differences from the above, as you will use different disk drive designators to refer to files.

Getting Into Some Output

We are ready to examine some actual FORTRAN program steps and see what they do. In this installment, you will write programs that write data onto the CRT screen, and also programs that accept data from the keyboard. First, we will prepare a simple output program to show how the language works.

The command to write something on the screen is, not surprisingly, the WRITE command. It's simplest form of syntax is as follows:

```
WRITE (u, f) k
```

where:
u = the Logical Unit Number (LUN),
f = the number of the FORMAT statement that tells the compiler how to layout, or format, the output, and
k = is a list of variable names, separated by commas, that tell the computer which data items to print.

Let's look at the easiest parts first. The variable list is exactly the same thing you would find following the command PRINT in the BASIC language statement below:

```
PRINT A$, B1, CX, XX, A2%
      |_____________|
             |
This part is the variable list, k __|
```

That was straightforward; it's just like BASIC. The Logical Unit Number is also clearcut: it simply tells the computer where to send the output. It also hails back to mainframe computers, where each tape drive has a dial on it, so that the operator can select its logical number for any program. The tape drive is always physical unit 3, for example, but it can become any logical unit number the program expects to see just by turning the dial.

BASIC tells the computer where to send output with a family of commands: PRINT sends it to the CRT, LPRINT sends it to the printer, and PRINT #n sends to disk file n. In contrast, the FORTRAN syntax is completely uniform. There is only one output command, WRITE, and the LUN tells where output goes (or, as we shall soon see, where the input comes from). As delivered, FORTRAN-80 is configured with the following equivalences of LUN's and output media:

```
LUN 1,3,4,5     =   Console CRT & keyboard
LUN 2           =   Line Printer
LUN 6,7,8,9,10 =   Disk Files.
```

If you wanted to write the variables J3 and ALEX on the CRT screen using the format specified in statement 106, the command would be:

```
WRITE (1,106) J3, ALEX.
```

We will consistently use LUN 1 to refer to the CRT, rather than jumping among the other three choices. Later, we'll investigate how to reassign LUN's, and this consistency will pay off in not having to rewrite code.

The last remaining hurdle is the FORMAT statement. FORMAT tells the computer how to print the output. It is analogous to the PRINT USING command in BASIC with one major exception: you must always use a FORMAT statement for FORTRAN I/O; there is no FORTRAN counterpart to BASIC statements like PRINT X, Y.

The FORMAT statement is a powerful feature of FORTRAN. It permits concise specification of extremely complex I/O. It can also be confusing at first blush, and much of what has been written about this feature of the language is intimidating. Rather than attempt a thorough investigation of FORMAT right now, we are going to keep to our "learn by doing" approach, and use just enough of its features to meet our needs. As we progress, we will learn more about how to exploit this command, and using it will become second nature. For now, trust me, OK?

As with anything else in computing, there is a series of codes that tells the FORMAT statement what to do where, and what kind of data to expect in that location. This statement does just what its name says -- it formats things. Each line we want printed must be specified with a format statement. For example, to print a line that says "This is a text line," the format statement (which I have arbitrarily labeled 100) is:

```
100 FORMAT (' This is a text line').
```

That looks pretty much like BASIC syntax, doesn't it? Three major points to remember are: (1) The space before the first word is not a mistake. FORTRAN uses this print position for some special things, and whatever you put there will NOT appear on the screen. We'll see what to do with it later; for now, just remember to leave it blank. (2)

Parentheses must surround the format specification, otherwise the compiler will raise a syntax error during compilation and you'll have to go back and fix it. (3) Similar to BASIC, text to be printed is surrounded by quotes, but FORTRAN uses single quotes here.

Let's Do It

We have now learned everything needed to write something on the CRT screen, so let's put together a program and do it! Using your text editor, enter the program listing shown in Figure 2 and save it in a file called SCREEN1.FOR. If you are using WordStar, use the Non-Document mode -- if you build a program file in Document mode, the compiler will get indigestion. To help you lay out the statements with the proper column alignment, this listing has the comment flags (C) in column 1. The statement number on the FORMAT statement begins in column 1, also. Because my editor has fixed tab stops every 8 columns, the program statements begin in column 8. As that is after column 7, there is no problem -- FORTRAN ignores blanks.

```
C     PROGRAM SCREEN1
C  This program demonstrates writing to the CRT
C  Set up output formats
10    FORMAT (' This prints a line of text')
C  Write the output on the CRT screen
      WRITE (1,10)
      WRITE (1,10)
      WRITE (1,10)
      WRITE (1,10)
      END
```

Figure 2. Program listing for Program SCREEN1

Notice the END statement. Each FORTRAN program must have one, and only one, of these. It is not an executable command, but rather a direction to the compiler that tells it the end of the program has been reached, and it is now time to start generating code. If there is no END statement, the compiler will stop in an error mode. If there is more than one, the compiler will read the first one as the end of the program and will not look any farther; everything after it will be ignored.

This is a basic program to demonstrate that what we have studied so far really works. It will print four lines on the CRT screen, each one saying "This prints a line of text."

Writing the program is half the battle; compiling, linking, and loading is the other half. As with the language, you needn't become expert on the compiler and linker to use them at this stage, so please follow the sample session in Figure 3 carefully. You type the information after the CP/M prompt and the compiler or linker prompt (which is *). End each line with a RETURN. The rest is printed by the computer. An explanation of what is going on has been added at the right; this will not appear on your screen, of course.

Some words of caution: F80 will not translate lower-case commands to upper-case, as will CP/M. If you enter the file name as "b:screen1," you will see the message "?Command error" appear on the next line. L80 isn't as fussy, but it is a good habit to use upper-case letters all the time, and the problem won't arise. Likewise, be careful about drive designators. If you get a "Not Found," you didn't tell the compiler where the .FOR file is. The LINK-80 program is slow. It will take 15-25 seconds to link this program on a 4-MHz H-89, so don't worry if the machine appears to be hung up -- it's just thinking. If you get any other types of errors, look up the error message in your FORTRAN User's Manual. You probably entered something incorrectly in the listing; check the source code carefully and recompile.

If you now look at the B: disk directory, you will see that besides the file SCREEN1.FOR that you wrote with the editor, there are now files named SCREEN1.REL and SCREEN1.COM. The former is the output of the compiler that was used by the linker as its input, and can be deleted. The second is a true command file representation of your FORTRAN code, and can be run just like any other .COM file, by typing its name at the CP/M prompt. Try it.

Congratulations! You have now successfully compiled and executed a FORTRAN program that generates output to your CRT, and it didn't hurt a bit. Let's now expand our capabilities to include data input.

On To The Input

The command to cause FORTRAN to read data is the READ command, and it has the same syntax as the WRITE command:

```
READ (u, f) k
```

where:
u = the Logical Unit Number (LUN),
f = the number of the FORMAT statement that tells the compiler the

```
A>F80          <<This loads and starts the compiler, F80.COM
*=B:SCREEN1    <<This tells the compiler to compile the file
$MAIN            B:SCREEN1.FOR, and place the results in a file named
                 B:SCREEN1.REL.  F80 names this program $MAIN
*↑C            <<Leave F80 with Control-C
A>L80          <<Invoke the Linker, L80.COM

Link-80  3.4  01-Dec-80  Copyright 1979,80 (C) Microsoft  <<L80 sign-on

*B:SCREEN1,B:SCREEN1/N/G   <<Tells L80 to load the file B:SCREEN1.REL,
                                create B:SCREEN1.COM on disk, and run it
Data    0103    19B7    < 6324>

37128 Bytes Free           <<Statistics about the program size and
[0104   19B7       25]       location.  We'll use these later.  For
[Begin execution]            now, just notice they are here.

This prints a line of text   <<Output of our program!!
This prints a line of text
This prints a line of text
This prints a line of text
A>                           <<All done, back in CP/M.
```

Figure 3. Sample session to compile, load, and run SCREEN1.FOR

layout, or format, of the input, and
k = is a list of variable names, separated by commas, that tell the computer which data items to fetch.

The only new thing we need to know is how to tell FORTRAN the layout of data items. For now, we will use only some of the basic capabilities of the FORMAT statement. When we are dealing with numbers, we must tell FORTRAN -- in the FORMAT statement -- what kind of numbers they are, how large they may be, and how we wish them formatted for presentation or input. It is easier to do than it is to tell about.

For now, we'll use only two kinds of numbers: integers, and real (also called floating-point) numbers. We tell FORTRAN about their characteristics in much the same way as we tell BASIC how to output things with a PRINT USING statement. For integers, we use a code of the format In, where I tells FORTRAN that this is an Integer, and n tells it how many places the integer may occupy (e.g. 1000 is a four-place number). FORTRAN insists that every number have a sign, and even though it doesn't print a + before positive numbers, we must allow a space for that sign when we tell FORTRAN about the integer. So, the number 1000 would require a specification of I5; I to identify it as an integer, and a width of 5 (four for the digits, plus one for the sign).

Floating-point numbers, like 373.78, are specified similarly, but we must also tell how many decimal places we want FORTRAN to put in. The field specification looks like this: Fn.d. F identifies a Floating-point number, n specifies the total field width (including one place for the sign and another for the decimal point), and d specifies the number of decimal places. The number above, 373.78, would require a field specification of F7.2 (F for Floating point; 5 digits + 1 sign + 1 decimal point = width of 7; 2 places after the decimal = d = 2).

It is always OK to specify a width wider than you need; FORTRAN will fill the left-hand end of the field with blanks. If you specify a field too narrow for the largest possible number in it, however, you will generate an error. This will be obvious on output, because the program will print asterisks. On input, however, it can be insidious and can cause significant errors. We will look at these aspects in detail in a later column; for now, be sure you make your fields wide enough for the largest number that can appear in them.

On input, the FORMAT specification will automatically insert the number of decimal places you specify unless you actually type the decimal point. This can be handy if you have a lot of data to enter and want to save keystrokes. It can be very perplexing if you forget about this feature and find the values of your data suddenly altered!

Figure 4 shows the output representation of some numbers with various FORMAT descriptors. Figure 5 does the same for inputs.

FORMAT Descriptor	Internal Value	Output (b=blank)
F12.3	+3.14159	bbbbbbb3.142
F8.5	+3.14159	b3.14159
F7.5	+3.14159	*.14159 (Error; number won't fit)
F5.2	-6.75	-6.75

Figure 4. The effect of FORMAT descriptors on output.

FORMAT Descriptor	Input (b=blank)	Internal Value
F12.3	3.14159	+3.14159
F12.3	314159	+314.159
F8.5	314159	+3.14159
F5.2	-1.	-1.00

Figure 5. The effect of FORMAT descriptors on input.

Notice from Figure 5 that if you enter a decimal point as part of the input, that is where the decimal is placed. If you do not enter it explicitly, the FORMAT descriptor establishes where the decimal point should be. As you can see, the difference in what is stored is large.

Let's now write and compile a program that demonstrates the effect of the FORMAT descriptors. Using your editor, enter the program listing shown in Figure 6. Save it as file B:SCREEN2.FOR.

Compile the program SCREEN2.FOR using F80 and L80, as shown in Figure 7. If you receive any error messages, check the source file (the one you created with the editor) very carefully for errors. Be particularly sure things are in the correct columns. The listings in this article are the exact ones I compiled; they were inserted into the article directly from the disk on which they were created, so you can use them to check your code.

```
C     PROGRAM SCREEN2
C  This program demonstrates screen/console I/O
C  Set up output formats (100-199)
100   FORMAT (' Input integer data now: ')
101   FORMAT (' Input floating point data now: ')
102   FORMAT (' Integer data is: ', I6)
103   FORMAT (' Floating point data is: ', F7.2)
104   FORMAT ('  ')
C  Set up input formats (200-299)
200   FORMAT (I6)
201   FORMAT (F7.2)
C  Print prompts and read data in specified format
      WRITE (1,100)
      READ (1,200) J
      WRITE (1,101)
      READ (1,201) F
C  Print a blank line
      WRITE (1,104)
C  Print the data you input, to verify that it is correct
      WRITE (1,102) J
      WRITE (1,103) F
      END
```

Figure 6. Listing for Program SCREEN2, to examine FORMAT characteristics.

```
A>F80

*=B:SCREEN2
MAIN$

*↑C
A>L80

[sign-on happens here]

*B:SCREEN2,B:SCREEN2/N/G

[data printed, then program runs]
```

Figure 7. Sample session to compile, link, and load SCREEN2.

SCREEN2 prompts you for input, and then prints what you gave it, using the same FORMAT descriptors for input and output. Run the program several times from the CP/M prompt, using different numbers. Deliberately use numbers that are too large or have different decimal place assignments than the FORMAT statement calls for. Then change the FORMAT descriptors in the source file, recompile and reload the program, and notice the effect this has on the I/O. After you have done this for awhile, you will have a good understanding of the FORMAT statement and how it affects what FORTRAN does.

What Next?

By the time you have reached this point, you have absorbed most of the "hard stuff." For some reason, many people are put off by the FORMAT statement. As you can see, you don't have to be. It is straightforward, and it lets you talk to the program without ever thinking about the punched cards you read about in all those FORTRAN manuals.

In the next installment, we are going to examine some of the mathematical abilities of FORTRAN, and tie them to an I/O package to form the beginning of our stock analysis system. Until then, spend some time playing with the FORMAT, READ, and WRITE statements to get comfortable with them. Try writing some short programs like SCREEN2; whether they run or not, you will learn a great deal about the language and its I/O abilities.

The Computer Time-Saver

A. E. McLaughlin, Jr.
32 West End Avenue
Brentwood, NY 11717

It is a well known fact that people buy computers to save time. To computer owners, however, it is another well known fact that computers are like jealous lovers in that they demand more and more of their owners' time. As a result, sooner or later the owner of any computer system is apt to start thinking about ways to make it work more efficiently.

In most cases, though, the real goal is not so much to make the computer run faster as it is to increase the throughput. Getting the JOB done faster, in fact, is the reason many computer owners part with their current machines and lay out great sums of money for new systems. It is also the purpose of many additions and modifications, however, and those can often postpone the replacement of a whole system.

In my case, it seemed clear that reducing the time needed to get the job done could free me for other interests, like programming to make the computer do my bidding instead of vice versa. Over the years, then, I have investigated and sometimes purchased various programs and hardware designed to give me faster throughput. What I've finally assembled seems to offer me both speed and reliability, and it may be equally valuable to other H/Z89 users.

My first purchase was a better system command processor that freed me from typing out commands in full. Then I got another processor for its additional features, and finally I bought the one I use now (SYSCMD+) for its ability to load and execute concurrent running programs, called "tasks."

Next came a software print spooler, which allowed me to get to work on the next job more quickly. Unfortunately, between the resident driver and the spool area, it also took up valuable memory space as well as CPU time, so I went on to consider the purchase of several hardware print spoolers.

SuperSpooler (SS-1000) by Compulink Corp. seemed to offer most of what I wanted. Among other features, it had its own Z80 processor, 16K (optional 62K) buffer size, and optional serial port. The only problem was its price: $349 for 16K, or $508 for 62K, and another $95 for the serial option. Given that, I decided it would be much cheaper to trade in my old printer for a new near letter-quality printer with a buffer.

I chose the Micro Peripherals, Inc., MP150G Model B1 (150 cps at $795 from Studio Computers, Inc.), partly because of the optional MemoryMate buffer ($95 for the 16K version) and partly because of the printer's ability to load a program selectable "alternate font." Any of the fonts supplied with the "Style Writer" package (or any fonts I create) can be downloaded to the printer and used instead of graphics to print individual documents or parts thereof. Since the switch can take place "on-the-fly," insertion of words or phrases in italics or any other special print is easy.

My next discovery was that, because I have both a Z89 and an H89, it was cheaper to get the MPI optional serial port ($95) than to purchase two parallel port cards and lose one expansion slot in each computer. The combination of the serial port's 19,200 baud maximum speed and the printer's buffer allows the transmission of 2400 characters (or thirty 80 character lines) of data per second. This means the computer could theoretically fill the 16K buffer with three and a third pages of sixty 80 character lines in less than seven seconds!

Granted, this speed can't actually be achieved unless all the data comes from memory with no disk accesses involved. But-- speaking of disks-- the 16K buffer can hold a complete 62 sector file at one time. No matter what I have to print, the system is usually finished and ready to do something else before I am, anyway.

As a first step, this is great for anyone who does a lot of printing. But if you are "computer-bound," you also want something to speed up the jobs that don't print. The fastest (and cheapest) method would be one of the 4 Mhz CPU clock options.

These come in various forms, from just instructions to complete hardware ready to install, but the Kres "2/4 MHz Solution" is the most complete mod in this category that I have seen. For $79.95 it provides software control of both CPU and I/O bus speed, its own oscillator for precision clock control, simple plug-in installation, and all gold-plated contacts for long, trouble-free life.

Since all the speed mods double the processor speed, their advantage is obvious. The disadvantages take a little bit of digging to see, but I learned that they're significant. First, in most cases, I/O will still be done at the slower speed unless all the device drivers are modified. But this wouldn't be too difficult if it weren't for the fact that many programs have at least one routine of time-dependent code. Unless these programs can also be modified (usually an impossible task unless you are a programmer and have the source code available), you have to discard them or have some way of switching back to the slower speed to run them. With the I/O drivers permanently modified, this would be impossible even if your mod contained software or hardware speed-switch control (and not all do).

In addition, even if you avoid the I/O problem by running your mod at the slow speed, you still have to remember which programs require that speed and then remember to switch back before starting them. Unfortunately, the program I use most is my editor (EDIT19), which uses several timed functions. I don't have the source code, so it is one of the programs I'd have to run at the slower speed. Also, forgetting to switch could be disastrous, since I could lose hours of work if the whole system got hung in one of those routines. For me, at least, this sort of speed mod apparently wasn't the answer.

A more expensive method of getting an increase in the clock speed is to replace the whole CPU board with either the Super89 from DG Electronic Developments Co. ($829) or the H1000 from Technical Micro Systems, Inc. ($1495 complete with 256K RAM, CP/M and HDOS utilities, manuals, and your choice of CP/M-86 or MS-DOS). The 64K Super89 adds up to 256K (at $125 per 64K) of bank selectable memory that can be used as an "invisible" (electronic) disk, a real time clock, three extra peripheral expansion slots, and an optional AMD9511 arithmetic processor. The H1000 has a switch-selectable 8086 CPU that can address all of the 1048K memory the

board can hold and, when used with its Z80 CPU, can also use the memory above 64K as an electronic disk.

The problem with both of these boards, however, is that with their higher 4 Mhz clock speeds, they present the same program compatibility problem as the mods which only change the CPU clock. Besides that, the Super89 presents the additional problem of not having the 2 Mhz speed option at all. These weren't my answer, either.

One more modification which adds extra memory is the Magnolia Microsystems 128K memory board (currently $495, direct from Magnolia). This board takes the place of the Heath/Zenith 16K memory board and is supplied with CP/M PLUS Version 3.101, which can use the extra storage as bank selectable memory to run larger programs. The BIOS (M348B.HEX) and utility (BOOTINV.COM) supplied with the board also enable its use as an electronic disk when running under CP/M 2.242. In addition, Ultimeth Corporation supplies HDOS support for the MMS memory board with its invisible disk driver program, AXMMS.DVD. This program is available for $50 from Quikdata Computer Services, Inc., (Ultimeth's exclusive retail dealer) and is supplied with another program, called CACHEMMS.

It's this second program that seems to be the ideal solution to the CPU speed-up problem. When CACHEMMS is loaded, it modifies the HDOS code resident in the H89 memory to allow use of the additional MMS memory (or SUPER89 memory, according to the literature) as a high speed disk buffer. It monitors all subsequent references to the system device driver (SY.DVD) and checks for the presence of the specified file in the banked memory. If it isn't found there, the file is read from the disk drive and written to both the HDOS disk buffer and the cache memory. All subsequent accesses to the same record are satisfied from banked memory, as if they had been written to an invisible disk originally.

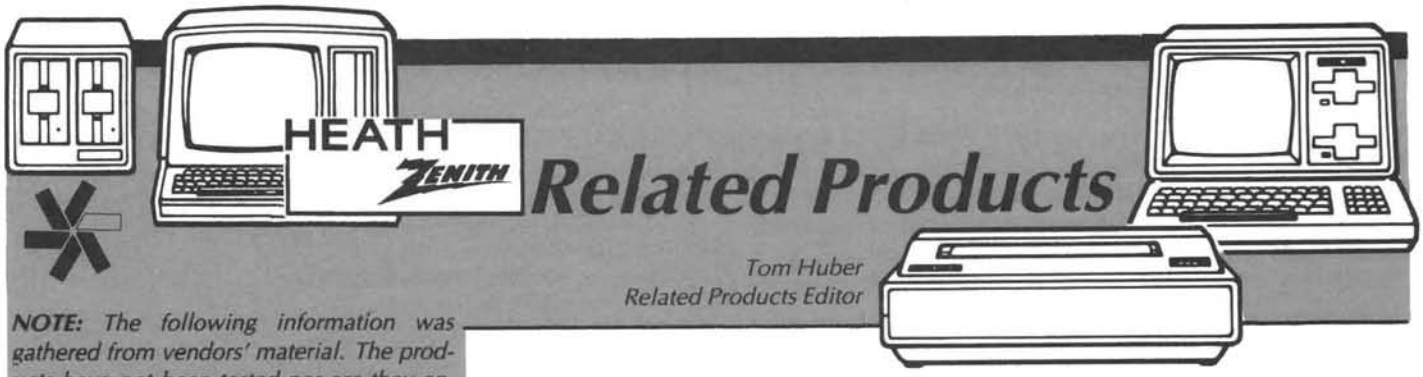
Some time is saved here, and even more is gained because files don't need to be initialized and loaded beforehand. Also, there's no need to determine in advance what files will be needed later. No matter what disk they are located on, they are loaded automatically when first needed. No space is wasted, either, since the only files loaded are those which are actually used.

When the cache memory space is filled, the next disk access for a file which is not in cache results in the new file replacing the "least used" file in the banked memory. This is the file that has been in cache longest without being accessed. Records which are modified and then written back to disk are also updated in cache, though, and this update counts as an access to keep the record "current."

The time saved with files which are accessed often is obvious. One of the less obvious advantages appears when a disk-to-disk copy of several files is done. The select light on the "TO" drive never goes off until the last file is written! Apparently, although the copy program (PIP) can only hold so much of the file at one time, the whole file is read into cache and then fed to the program as needed. This appears to reduce the copy time approximately 25% to 30%.

Another less obvious advantage is the savings in memory space possible with the elimination of large I/O buffers. For example, EDIT19 has the code for all its less frequently used commands located in overlays which are only loaded when those commands are used. It also has an option which allows an overlay to remain in core once it is loaded. This allows faster execution if the commands in an overlay are to be used repeatedly. The problem with this, however, is that as each new overlay is loaded, it takes up more memory. In the past, I would sacrifice memory for speed until I ran out of core space, and then I'd execute the DUMPO command to get the overlay memory space back. But now I've turned off the AUTLOAD option, since the overlays load into cache also and are instantly available from then on. This way, only the small area of memory reserved for the loading of temporary overlays is lost.

The combination of the MPI printer, Magnolia Microsystems' extra memory, and Ultimeth's CACHEMMS program is the answer to my search for faster throughput. If you want to work fast and print fast, try this combination. Like me, you may decide you really don't need a new system after all.



Related Products

Tom Huber
Related Products Editor

NOTE: *The following information was gathered from vendors' material. The products have not been tested nor are they endorsed by HUG. We are not responsible for errors in descriptions or prices.*

Specialized Software for dBASE II™

Custom Software Services has introduced Sandman™ Medical, Dental, Legal, and Inventory Management programs for CP/M, CP/M-86, and MS-DOS operating systems. CP/M requires 64K; the others, 128K. The programs are totally menu driven, include the dBASE II source code for customization by qualified programmers or by the vendor, Custom Software Services. Documentation is available separately or with a demo package. Specify disk and operating system format. Requires dBASE II.

Vendor: Customer Software Services Inc.
P.O. Box 65
2500 Highway 79 S.
Guntersville, Alabama 35976
(205) 582-4168

Prices: Sandman™ Medical: $1,000.00
Sandman™ Dental: $1,000.00
Sandman™ Legal: $1,000.00
Sandman™ Inventory Management: $1,000.00
Demo (includes manual): $50.00 (specify software)
Manual only: $25.00 (specify software)

QUERY!2 CALC Data Base Has Math to 12 Places

Hoyle and Hoyle Software has added QUERY!2 CALC to their line as a result of the request of QUERY!2 users for arithmetic functions. It is a new program for producing calculations and writing reports using QUERY!2 data files. Formulas may include addition, subtraction, multiplication, division, square root, summing up to ten fields, and averaging. Available in all Heath/Zenith formats.

Vendor: Hoyle and Hoyle Software
604 South Elam Avenue
Greensboro, NC 27403
(919) 378-1050

Price: $49.95

Artificial Intelligence

RACTER, a software program for CP/M and PC machines, will engage the user in an on-going English intelligent conversation from politics to pole-vaulting to "witty small talk." The program has the capability to "remember" and refer to conversations it had previously.

Vendor: John D. Owens Associates, Inc.
12 Schubert Street
Staten Island, NY 10305
(212) 448-6283 or (212) 448-2913

Price: $69.95

ZSTEMpc, VT100 Emulator for the PC

KEA Systems has announced a VT100 Emulator for the PC, which allows the computer to be connected either locally (hard-wired) or remotely (via modem) to any of the DEC minicomputers. This program makes the PC look like a VT100 with the advanced video option. It is easily configured and includes 132-columns (via windows); double-high, double-wide characters; reverse screen; smooth scrolling (user selectable rate); VT100 line graphics; programmable answerback; bidirectional file transfer (ASCII and XMODEM); concurrent serial and parallel printer support (VT102); and keyboard softkey/macro programming support. ZSTEMpc is written in 8088 assembler code and supports speeds to 38.4 Kbaud. Requires 128K, MS-DOS, and is also available for the Z-100 under MS-DOS.

Vendor: KEA Systems Ltd.
Dept ZR-6
#311-811 Beach Avenue
Vancouver, BC V6Z 2B5
Canada
(604) 687-2744

Price: (ZSTEMpc - VT100): $150.00

MPI Announces "X"Printer and SwitchMate Option for the PrintMate 150

The "X"printer is a new data processing and executive quality dot matrix printer from MPI. It is a bidirectional, 80-column printer that provides draft quality at 180 cps, correspondence quality at 120 cps, and executive quality ("near letter quality") at 50 cps. The printer supports downloadable characters and pitches of 10, 12, 14.4, 15, or 17 cpi. It prints international characters, superscripts, subscripts, and images. Friction and tractor feed are standard. Options include the SideWinder (requires the optional 64K buffer) which can rotate the printed text 90 degrees (useful for spreadsheet applications), allowing an almost unlimited printing width without changing computer software.

The SwitchMate is an option for the PrintMate 150 wide carriage printer that allows two printers to be "daisy chained" together. The select/deselect switch on the front of the PrintMate 150 selects which printer will receive data from the computer. The option is factory installed and consists of a plate on the back of the PrinterMate 150 with two DB25 connectors and an internal electronic switch. (Note: The price of this option was not given in the news release provided by MPI.)

Vendor: Micro Peripherals, Inc.
4426 South Century Drive
Salt Lake City, UT 84123
(901) 263-3081

Price: $695.00 ("X"printer)

Friendliware Offerings Include Atlantis Adventure

A number of programs are being offered in the CP/M format for Heath/Zenith computers that run that software. **Atlantis 2.0** (requires 64K) is a scientifically/historically accurate four-dimensional (time is important) puzzle with functional maps that can culminate in the finding of lost Atlantis. It is generally non-antagonistic, but one shouldn't vex the gods, either. **Cells 3.1** (requires 32K) is a graphics puzzle where the player attempts to reduce 224 cells into one by jumping pieces, similar to checkers. The puzzle is divided into 7 parts of 32 cells each, resulting in 7 different challenges during the course of the game. **Cubes 3.1** (48K) is similar to Rubic's cube. It is based on the 30 MacMahon cubes where each cube has 6 faces marked in a unique order. The challenge is to stack 8 cubes to form a larger 32 cube, also a MacMahon Cube. **Disarm 3.1** (64K) is a graphic adventure puzzle where you command a robot to scan, analyze, and manipulate a large object and render it safe. The robot's vocabulary is limited at first, but will "learn" additional words that are peculiar to the object. **Dots 3.1** (48K) is an implementation of the maximum-no-three-in-line problem on n-by-n grids. The user can select any number from 2 to 20 for the grid matrix. All three-in-line conflicts are trapped and flagged. **Firefly 3.1** (48K) is a two-part dynamic game/puzzle. The first part involves capturing 20 fireflies in a jar to use their light to read clues for solving the second part, which is naming a familiar saying. **Match 3.1** (48K) is a collage puzzle with 38 pairs of objects and one odd one. The goal is to locate the odd object. A strategy that minimizes wasted effort will be needed to achieve a high rating. **Pid 3.1** (48K) is an educational program introducing proportional integral derivative control, and uses a graphic example of a simple motor-speed-control problem under a variable load. Prior knowledge of calculus is not needed. **Slabs 3.1** (32K) is a variation of the Towers of Hanoi game/puzzle.

Vendor: Friendliware
P.O. Box 21206
Lansing, MI 48909
(517) 882-1675

Prices: Atlantis 2.0 - $40.00
Cells 3.1 - $12.00
Cubes 3.1 - $15.00
Disarm 3.1 - $35.00
Dots 3.1 - $15.00
Firefly 3.1 - $20.00
Match 3.1 - $18.00
Pid 3.1 - $23.00
Slabs 3.1 - $9.00
(add $2.00 handling to total order;
Michigan residents add 4% sales tax)

Z-100-PC Software Package

Stellation Two has announced the INVISIBLE OPTIMIZER, a disk buffering software package for PC computers, which makes use of memory expansion inside PCs. The software keeps track of disk activity and automatically duplicates the most needed disk data in unused memory, allowing the user to obtain this data from memory instead of disk, reducing the amount of disk I/O (and time needed).

Vendor: Stellation Two
P.O. Box 2342
26 W. Mission St. #3
Santa Barbara, CA 93120
(805) 966-1140

Price: $69.00

HDOS, CP/M, Z-DOS, and MS-DOS Trivia Game

Husker Systems' TRIVIA WIT comes with over 500 questions in four categories (General Interest, Entertainment, History, and Athletics) and a utility so that you can add your own questions to the game. Join the thousands that play trivia with this computerized version of the game. Specify operating system and disk format.

Vendor: Husker Systems of Nebraska
6657 Redick Ave.
Omaha, Nebraska 68152
(402) 572-6290

Price: $30.00

Chocolate Bits For Chocoholics

The Software Toolworks has introduced the fourth computer cookbook in the popular Computer Chef series, titled, CHOCOLATE BYTES. Recipes range from Grasshopper Cheese Cake to Chocolate Zucchini Bread (I always wanted to know what to do with all that extra Zucchini.), all culled from the chocolate newsletter, Chocolate News. Unlike conventional, printed cookbooks, this program can adjust any recipe and its ingredients to the number of servings needed. It is formatted for HDOS, CP/M, Z-DOS, and MS-DOS.

Vendor: The Software Toolworks
15233 Ventura Blvd., Suite 1118
Sherman Oaks, CA 91403
(818) 986-4885

Price: $29.95

DriveLiner Checks 8" Disk Drives

Chandler Software has developed a CP/M program for verifying alignment of 8" floppy disk drives, and includes a Dysan Diagnostic Disk. Drive speed timing is performed on systems that use a Western Digital controller (Editorial note: the PC family does not use WD controllers). Head centering, radial and azimuth alignment tests are performed automatically on any CP/M 2.2 8" floppy system, without having to remove the disk drive from the cabinet.

Vendor: Chandler Software
273 West Shore Dr.
Marblehead, MA 01945
(617) 631-4685

Price: $65.00 (includes Dysan Diagnostic Disk)

Improved Transcendental Functions for FORTRAN-80 and BASIC-80 Compilers

Chandler Software is also offering a library of improved transcendental functions for Microsoft's FORTRAN-80 and BASIC-80 compilers. SUBMIT files are provided to facilitate incorporation into the compiler libraries. ALl standard FORTRAN REAL and DOUBLE PRECISION functions are provided in source code, as well as .REL format (excepting those that are satisfactory as they come in the Microsoft library). The same accuracy improvements are obtained with the BASIC compiler. Faster arithmetic subroutines are provided for Z80 systems (25% increase in single-precision transcendental functions). The Z80 square root function is 400% faster and more accurate than the 8080 code.

Vendor: Chandler Software
273 West Shore Dr.
Marblehead, MA 01945
(617) 631-4685

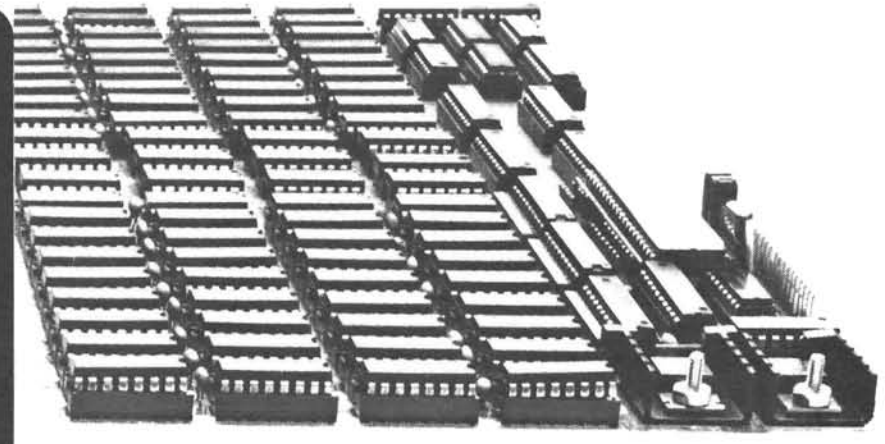
Price: $89.00

"Hey UCI, we heard you have memory boards for ZENITH and CompuPro and Lomas and..."

"This board is unique. It is the last RAM board you'll ever need. The board is a very excellent board and has been up and running in our Z110 winchester system under MP/M for some time now. No problems. We are very satisfied with that board."
Henry E. Fale
QUIKDATA, Inc.
Sheboygan, WI

"UCI boards provide better value than other memory boards. They have excellent quality, are easy to install, and have expansion features other boards don't have. We recommend this board to our customers."
Ray Massa
Studio Computers
Birmingham, MI

"For anyone interested in a very good Z100/S100 memory card, they should have their dealer contact UCI Corporation. A nice thing about the board is that is comes already configured for the Z100. UCI deserves excellent marks for support and user assistance."
Chuck Ballinger
Acme Computers
Spokane, WA



EXPANDABLE RAM *FOR S-100 SYSTEMS*

Look at these features...

- 256K up to 2MB of memory *without piggybacking* (9 bits wide)
- 8 bit or 16 bit systems (automatically selected)
- Operates with up to 8 MHz CPUs (with hidden refresh)
- On-board parity generation/detection
- Z-DOS Ram Drive Software
- Zenith System Memory and Ram Drive operational on same board
- Unique Switching Scheme — Zenith System can access entire 2MB memory

Better yet, look at these amazing prices...!

256K	$795 list	1.5MB	$3049 list
512K	$1145 list	2MB	$3995 list
1MB	$2095 list		

*Contact These UCI Dealers**

Beaverton, OR
WRAM Corporation
8243 SW Cirrus Dr.
(503) 641-3550

Birmingham, MI
Studio Computers
999 S. Adams Rd.
(313) 645-5365

Detroit, MI
Merit Data Systems
19622 Plymouth
(313) 838-6026

Florence, MA
The Computer Farm
30 N. Maple
(413) 584-2038

Ft. Myers, FL
Computer Tech
3049 McGregor Blvd.
(813) 337-1888

Ft. Walton Beach, FL
Compustation
99 Elgin Pkwy.
(904) 243-1501

Houston, TX
Payload Computer Services
15006 Sun Harbor
(713) 486-0687

Huntsville, AL
Dudley-Wales Ent.
2109 Bob Wallace Ave.
(205) 534-6868

Raleigh, NC
Computopia
1275 Buck Jones Rd.
(919) 469-0267

Sheboygan, WI
QuikData, Inc.
2618 Penn Circle
(414) 452-4172

Spokane, WA
Acme Computers
1727 E. Sprague
(509) 535-4122

St. Charles, MO
First Capital Computer
1106 First Capital
(314) 946-1968

Summerville, SC
The Computer Terminal
908 Bacons Bridge Rd.
(803) 875-4727

Winston-Salem, NC
ITS Computer Center
2070 Beach St.
(919) 727-0290

***Or All Heathkit Electronic Centers**

Also from UCI...

ZENITH 8 MHZ SPEED UP MODULE .. $49.95 list
Increase your system speed from 5MHz to 8MHz

UCI CORPORATION
an affiliate of ACME-CLEVELAND CORPORATION

CALL TOLL FREE
1-800-824-2667

IN OHIO CALL
216-673-5155

948 CHERRY STREET KENT, OHIO 44240

☜ Vectored from 6
However, if the TAB is sent to the system console using the DOS-F_DRCIO function, a special character which looks like a zero, is displayed instead of positioning the cursor as the TAB should do. I'm told by the Zenith System Consultation Group that this character is an IBM graphics character and this is consistent with PC-DOS.

This caused me some problems when a program worked fine on ZDOS and MS-DOS on the my H100, but would not work correctly on the Z100-PC series. The way around this inconsistency is to use a different DOS function or trap the TAB character and send out an appropriate number of spaces instead. This latter method is what I used in the program that I developed originally for the H/Z100, but converted it to also run on the Z100-PC series.

Richard L. Mueller, Ph.D. (Rich)
11890-65th Avenue N.
Maple Grove, MN 55369

Peachtree: Wouldn't Use Anything Else

Dear Walt:

First, congratulations on the fine job you and your people are doing on REMark. It is a real treat every time it comes out.

I am living overseas and communications are tenuous at time. Mail gets lost and delayed. Calls to the states are easy enough, but expensive. With this in mind, I would like to comment on the Peachtree software package PT5000, which I bought along with my Heath All-In-One at the end of my vacation last year on my way back to my overseas assignment in Saudi Arabia.

The PT5000 package being my first encounter with microcomputer software, I had my share of beginners problems. Back home a quick phone call would have solved them very quickly. Over here, letters would have to do. Despite the fact that it is more expensive to answer letters, despite the fact that some of our mail got delayed which caused confusion, and despite the fact that I asked some pretty basic (it's hard to admit that they were dumb) questions, Peachtree answered them promptly and in detail.

At this time, I have numerous letters from Peachtree Technical Support with very useful tips on how to use some of the more advanced features of PeachText along with a list of fixes to a number of glitches in the original software.

While, admittedly, PT5000 is no longer state-of-the-art, I recommend it as a Best Buy.

Since then I have tried a lot of other programs including Condor, the Cadillac of relational Data Base Management programs, I am using Lotus 1-2-3, I have tried WordStar and Multimate, etc. They all have great features. But they have their bugs. My mainstay still is PT5000.

Now that one can buy the Peachtree Business Graphics package, all I need to stay happy with PT5000 is a driver for the Star Gemini 10/15X and the time-between-dates function for Peachcalc.

A similar letter will go to Buss and Sextant.

Again, I appreciate REMark.

Very truly yours,

Peter W. Guenther
c/o Aramco
Box 5551
Dhahran, Saudi Arabia

Z-100 PC Hardware Mod To Improve Compatibility In Video Performance

Question: When I run the new Microsoft Flight Simulator version 2, the sky is a green color instead of a blue color. How can I correct this?

Answer: A simple change to the video board is needed to correct for this minor incompatibility with the industry standard computer. Remove the video board from the computer and locate the chip at the location on the board marked U347. Remove the chip from the socket and carefully bend pin 13 out a little bit so that when the chip is put back in the socket it goes outside the hole instead of into it. Restore the chip back to its socket. This will work for most Z-100 PC series computers.

If this simple change does not work, there is a slightly more difficult change which is sure to work. Remove the chip at U347 and bend pin 13 back in so that it goes back into the socket hole. Put a small blob of solder on pin 13 to short it to pin 14 which is right next to it. Remove the chip at the location U302 and bend pin 4 out and restore the chip.

This change will be incorporated into the video circuit boards of all future production for the Z-100 PC.

Zenith Data Systems
Software Consultation

CROSSREF - August 16

Dear HUG:

Reference my letter dated 15 August '84 (November '84 Buggin' HUG), concerning the CROSSREF program listing in the August issue of REMark, the enclosed listing and the comments herein should be considered in editing that letter for publication, if that is your intent.

When using CROSSREF on a different program from the one I used to test my original revision of CROSSREF, I noted three items that deserve additional comment.

1. To facilitate RUNning CROSSREF, it is better, as the author originally did, to have the program begin at line 65000 and have the END line as 64999. The enclosed listing makes that change, while still keeping the line numbering consistent. (Note that whenever lines are RENUMered in CROSSREF, the "IF LN=64999 . . ." statement near the beginning may have to be changed.)

2. I have amended CROSSREF to permit listing all lines which contain a REM or the ' symbol for a REM. One reason is to adapt CROSSREF for programs that are generic (i.e. that can serve different purposes by simply changing some of the program's statements or adding/deleting statements), and which use REMarks to cite the line numbers requiring changes and/or alternate statements. Another reason is discussed below (deleting and merging REMarks). If line references to REMarks are not desired, the following two lines should be substituted for those in the enclosed listing.

```
65020 IF MM=143 OR (MM=58 AND PEEK(MP+1)=143)
      THEN MP=NL-1
65026 AV$=""
```

3. When CROSSREF works on a line containing a FIELD statement, it merges the word AS with any variable that might precede it and with the string variable that follows it, and the combined form is treated as a unique variable. Hence, if the variables that appear in the FIELD statement are listed in CROSSREF's output, it is only because they are found elsewhere in the program, and there is no reference at that point to the line containing the FIELD statement. For example, if

micro/VERSAL™

MS-DOS††

H-DOS†† | CP /M†† | TRS-DOS††*

Use your IBM-PC/compatible or Z-100 to READ, WRITE, or FORMAT disks for the following systems:

- CROMEMCO C-DOS
- DATAVUE
- DEC VT180
- EPSON QX-10
- H-DOS
- HEATH/Magnolia
- HP125
- IBM CP/M-86
- KAYPRO II
- LOBO MAX-80
- MORROW MD2/MD3
- NEC PC
- OSBORNE (DD)
- OSBORNE (SD)†
- SANYO
- TELEVIDEO
- TI Professional CP/M-86 (48 tpi soft-sector)
- TRS 80 DOS
- TRS 80 III
- TRS 80 I†
- XEROX 820†
- XEROX 820 (DD)
- ZENITH Z-89†
- ZENITH Z-100/CPM85
- ZENITH Z-90

micro/VERSAL™ is easy to use—menu driven, and requires only 64K and 2 floppy drives or 1 floppy and a hard disk.

micro/VERSAL™ runs on: IBM-PC, IBM-PC/XT, Zenith Z-100, Zenith Z-150, Chameleon, Columbia, COMPAQ, CORONA, Eagle, Panasonic, Otorona, NCR, Sanyo MPC (no formatting on Sanyo,) Televideo 1605 and others.

$79.99
plus $4.00 shipping & handling

to order send check, money order, or MC or VISA numbers to:

ADVANCED SOFTWARE TECHNOLOGIES

417 BROAD STREET
BLOOMFIELD, NJ 07003
201-783-7298

† single density formats are available on the Z-100 only
†† MS-DOS, CP/M, H-DOS, and TRS-DOS are trademarks of Microsoft, Digital Research, Zenith, and Tandy Corps, respectively.
* Z-100 only.

The Software Toolworks presents: THE FUTURE

Languages, games, productivity software, computer cookbooks and more are formatted for IBM PC, PCjr, most CP/M systems and Heath/Zenith HDOS. They range in price from $19.95 to $59.95. For a free 41 product catalog contact The Software Toolworks, 15233 Ventura Blvd., Sherman Oaks, CA 91403, (818) 986-4885.

HERO 1 ASSEMBLER

Say GOODBYE to tedious machine language programming of HERO.

ROBOSOFT introduces a new assembler especially designed for owners of HERO and HEATH/ZENITH computers. It features beautifully simplified motor commands and a special pseudo to define word speech phonemes with inflections added as easily as +1,+2 or+3. The assembler output can be entered into HERO either manually or automatically through the new serial interface accessory.

The assembler is available for the H8, H89 or H-100 with either HDOS or CP/M operating systems, hard or soft sector. Please specify.

Get yours now for the special price of $59.95 complete with instruction manual and sample programs. Or, send for our free brochure illustrating all the features of this powerful assembler.

Please send check or money order to

The ROBOSOFT Company
P.O. BOX 1273
FALLS CHURCH, VIRGINIA 22041

SANTA DROID'S SOFTWARE SHOP

HERO PATROL $25.

Sophisticated program has HERO patrol the halls of your home or office, on guard and alert to intrusion.

- Adaptive & Self Correcting
- Verbal Reports & Interactions
- Easy to Use

HERO BUTLER $25.

HERO performs a variety of household chores easily addressable from his keyboard. Amuses your guests.

- Delivers Announcements
- Serves Food & Drinks
- Tends the Door

HERO PET $25.

The most amicable pet ever. Fully house broken. As user-friendly as a Cairn Terrier. Sleeps a lot.

- Mills Around
- Mutters Appropriate Comments
- Likes People

HERO GAMES

Fun for kids of all ages. Educational. Verbal responses.

Voice Control
Rudimentary voice recognition algorithm allows you to direct HERO through a maze while HERO keeps track of your time. Recognizes go, starboard, port, go backward, and allay-allay-ocean-free.

Hop-To-It
Sophisticated version of popular distance measurement game. HERO gives player a distance and tells him to hop-to-it. Distances given in feet/inches and meters/centimeters. HERO corrects misestimates and reacts delightfully to right answers.

Math Quiz
HERO poses arithmetic problems and waits for verbal response. Score based on speed and accuracy. Builds fundamental skills.

Games $15. each. Buy all three on one tape for $35.

HERO 1 is a tradename of Heath Co.

Check - Visa - MasterCard

Add $3. for shipping and handling. NM residents add tax.

Robotronix, Inc. Box 1125 Los Alamos, NM 87544

CROSSREF worked on the following lines:

```
10 FIELD #1,25*Z% AS A$,25 AS B$
20 A1$=A$:B1$=B$
```

the output would show:

A$ - 20
A1$ - 20
ASB$ - 10
B$ - 20
B1$ - 20
Z%ASA$ - 10

Finally, I liked the author's idea for saving memory by eliminating REMarks from the program used for execution and adding them to the hardcopy listing, but he was not clear as to the method of doing this - presumably with a text editor. However, since it is also desirable to have the REMarks available when listing the program on the screen, I developed a different method of implementing the idea, still using the text editor. The following steps will divide the target program into separate files, one for the program without REMarks and one for the REMarks only, which can then be MERGEd for printout either on the screen or printer.

1. LOAD MBASIC, then LOAD the target program and MERGE the enclosed version of CROSSREF. Type RUN65000 and press RETURN. In addition to variables and line number references, CROSSREF will find all REMarks and the ' symbol also, but the output listing will reference all lines with the ' symbol as if REM were used.

2. Refer to the output listing for lines containing a REM (which will include lines with the ' symbol), and then check the line-reference part of the listing to see if the REM line is referenced elsewhere in the program. If it is, and if the only thing on that referenced line is a REM, those lines containing a reference to a REM line must be amended to reference instead the next executable line after the REM line. In other words, using the CROSSREF output listing as a guide, edit the target program so that no GOTO or GOSUB statement directs program execution to a line that contains only a REM, since those lines are going to be deleted.

3. From MBASIC, dump to disk an ASCII copy of the program with REMarks (e.g. "SY1:TEST.BAS",A).

4. Exit MBASIC and load your text editor, then read in the saved program. (When dumping a file to disk from your text editor in the steps below, be sure to include the .BAS extension.)

NOTE: If you are certain that all of your program's REMarks are on lines by themselves and none are at the end of otherwise executable lines, you can skip Steps 5 and 6.

5. Using your editor's search function, find all REMarks that are not on a line by themselves, rewrite them to a new numbered line by themselves and delete them from the end of the lines where they originally appeared.

6. Dump the revised ASCII listing back to disk, using the same filename as you used to read it, and then read it back into your text editor again.

7. Delete all REMark lines, including the line number, and any blank lines that are left after those deletions. (The ease with which you can accomplish this will depend on the search commands available in your text editor. For example, mine has a command combination that permits executing a given series of commands an unlimited number of times by pressing two keys. Thus, after having found a REM line and deleting it once, the editor can then take over and find and delete all the other REM lines and any resulting blank lines.) Do not renumber any lines.

8. Dump the remaining file to disk, this time using a different filename, e.g. TEST1.BAS.

9. Read into your text editor the original target file from Step 6 (or Step 3, if you skipped 6).

10. Delete all lines except the REM lines. Be sure to delete any blank lines that remain, and do not renumber any lines.

11. Dump the remaining file to disk, this time using a file name similar to, but different in at least one character from, the file dumped in Step 8 above, e.g. TEST2.BAS.

12. Optional: You may now delete from the disk the original file from Step 3 and/or Step 6, and any backup files that your text editor may have left. You probably will also want to load the file without the REM lines and SAVE it in non-ASCII format (i.e. without using the ,A flag), and leave the file with only the REM lines in ASCII.

Your original target program is now divided between two similarly named files, one without the REM lines and one with only the REM lines. The one without the REM lines is, of course, the one you will RUN. If you want to see the whole program with the REM lines, simply load the program file to MBASIC, MERGE the file with the REM lines, and use LIST. If you want a hardcopy, just SAVE the MERGEd file to disk in ASCII (using the ,A flag) then copy that file to the printer.

Sincerely,

C.F. Mowery, Jr.
406 Van Reed Manor Drive
Brandon, FL 33511

```
64999 END
65000 REM *** Program to MERGE with another
            to find Variables/Line #s ***
65001 CLEAR 5000:LV=0:MP=29596:ED$=CHR$(27)+"E":
      DIM V$(99),V(99)
65002 PRINT ED$;"Variable/Line # Cross-Reference Utility"
65003 PRINT "1. List ALL Variables/Line # References":
      PRINT
65004 PRINT "2. List ALL Variable References Only":PRINT
65005 PRINT "3. List ALL Line # References Only":PRINT
65006 PRINT "4. Search for ONE Variable Only":PRINT:@
      PRINT "5. Search for ONE Line # Only":PRINT
65007 INPUT "Enter the # of your choice: ";CH:
      IF CH<1 OR CH>5 THEN 65002
65008 IF CH>3 THEN LV=1:PRINT:INPUT
      "Enter the Var/Line # to search for: ";@V$(1)
65009 V$(1)=V$(1)+" -":J=1:V$(2)="ZZZ"
65010 PRINT:INPUT
      "Do you want output to go to a printer? ";LP$:@
      LP$=LEFT$(LP$,1):IF LP$="Y" THEN OPEN "O",2,"LP:"
65011 GOSUB 65016:IF LN=64999! THEN 65035 ELSE GOSUB 65017:
      GOTO 65011
65012 X=INSTR(V$(J),"-"):X=X-2:IF AV$=LEFT$(V$(J),X)
      THEN 65013 ELSE RETURN
65013 IF STR$(LN)=RIGHT$(V$(J),LEN(STR$(LN))) THEN F=1:
      RETURN
65014 V$(J)=V$(J)+STR$(LN):J=LV:F=1:PRINT AV$"=":RETURN
65015 MP=MP+1:MM=PEEK(MP):IF MM=34 THEN RETURN ELSE 65015
65016 NL=PEEK(MP)+256*PEEK(MP+1):
      LN=PEEK(MP+2)+256*PEEK(MP+3):PRINT:@
      PRINT "Working on Line #";LN:MP=MP+4:RETURN
65017 FOR MP=MP TO NL-1:MM=PEEK(MP)
65018 IF MM=255 OR MM=15 THEN MP=MP+1
65019 IF MM=132 THEN MP=NL-1
65020 IF MM=143 OR (MM=58 AND PEEK(MP+1)=143) THEN QQ=1:
      GOSUB 65026:MP=NL-1
65021 IF MM=28 OR MM=11 OR MM=12 THEN MP=MP+2
65022 IF MM=29 THEN MP=MP+4 ELSE IF MM=31 THEN MP=MP+8
65023 IF MM=34 THEN GOSUB 65015
```

```
65024 IF MM=14 THEN GOSUB 65033 ELSE IF MM>64 AND MM<123
      THEN GOSUB 65026
65025 NEXT:RETURN
65026 AV$="":IF QQ=1 THEN QQ=0:AV$="REM":GOTO 65031
65027 AV$=AV$+CHR$(MM)
65028 MM=PEEK(MP+1):IF (MM>47 AND MM<58) OR
      (MM>64 AND MM<123) OR MM=33 @ OR (MM>34 AND MM<38)
      THEN MP=MP+1:GOTO 65027
65029 IF MM=32 THEN MP=MP+1:GOTO 65028
65030 IF MM=40 OR MM=91 OR MM=123 THEN AV$=AV$+CHR$(MM)
65031 IF CH=3 OR CH=5 THEN RETURN ELSE IF CH=4 THEN J=1:
      GOTO 65012
65032 F=0:FOR J=1TO LV:GOSUB 65012:NEXT:IF F=1
      THEN RETURN ELSE @ V$(J)=AV$+" -":LV=LV+1:
      GOTO 65014
65033 AV$="":X=PEEK(MP+1)+256*PEEK(MP+2):MP+2:
      AV$=MID$(STR$(X),2)
65034 IF CH=2 OR CH=4 THEN RETURN ELSE IF CH=5
      THEN 65012 ELSE 65032
65035 PRINT ED$;"Sorting":IF CH>3 THEN LV=1:
      GOTO 65042 ELSE SW=LV-1
65036 X=SW:SW=0:FOR I=1TO X:IF V$(I)>V$(I+1) THEN SW=I:
      V$(0)=V$(I):@ V$(I)=V$(I+1):V$(I+1)=V$(0)
65037 NEXT:IF SW>0 THEN 65036
65038 FOR I=1TO LV:IF VAL(LEFT$(V$(I),1))>0 THEN SW=I-1
65039 NEXT
65040 X=SW:SW=0:FOR I=1TO X:X1=INSTR(V$(I),"-"):
      X2=INSTR(V$(I+1),"-"):@
      IF VAL(LEFT$(V$(I),X1))>VAL(LEFT$(V$(I+1),X2))
      THEN SW=I:V$(0)=V$(I):@ V$(I)=V$(I+1):V$(I+1)=V$(0)
65041 NEXT:IF SW>0 THEN 65040
65042 FOR I=1TO LV:PRINT V$(I):IF LP$="Y"
      THEN PRINT #2,V$(I)
65043 NEXT:CLOSE:END
```


Index of Advertisers

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

CUT ALONG THIS LINE

1984 ARTICLE OF THE YEAR — BALLOT —

My Choice for "Article of the Year" is:

Author: _____ Issue: _____ Page: _____

The winner of the "1984 Article of the Year" Award will be announced in the March, 1985 issue of REMark.

NOTE: Your ballot must be received at the HUG office before FEB. 1, 1985 to be counted in the judging.

Mail your ballot to:
Article of the Year
Heath/Zenith Users' Group
Hilltop Road
St. Joseph, Michigan 49085

Heath/Zenith Users' Group

Hilltop Road
Saint Joseph, Michigan 49085

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

Volume 5, Issue 12
POSTMASTER: If undeliverable, please do not return.

P/N 885-2059