

\$2.50

# REMark<sup>®</sup>

Volume 5, Issue 10 • October 1984

P/N 885-2057



Official magazine for users of  computer equipment.

# First Principles

The basic goal of business is to get and keep satisfied customers. Seems obvious, but a surprising number of companies lose sight of this basic tenet of marketing. At First Capitol Computer, our first principle is "satisfy the customer". How do you do this? By having great prices and total knowledge of what you're selling. And what we sell is Zenith computer systems.

We're the retail sales division of Software Wizardry, and our knowledge of, and support for Heath/Zenith computer systems is legendary and award winning. The same expert Software Wizardry staff that develops top-notch hardware accessories and software for the Heath/Zenith community sells computer systems, peripherals, software and supplies for First Capitol.

Now most companies that offer that kind of total support and expertise (or claim to), also charge you total list price. Not at First Capitol! We offer prices you'd have to look under a rock to beat, and who wants to buy from a source like that?

Our systems expertise extends beyond the norm, too. If you have need of a drafting assistant, we can fix you up with one of the best-value CAD systems you can find, complete with plotter. Are you thinking about a Local Area Network (LAN)? Not only do we do networks, we'll discuss with you WHICH ONE of the several we support would be best for your application.

We don't sell toys, and we aren't a microscopic-print price list in the back of every magazine. What we do offer is the kind of support you expect from a computer systems specialist, AND prices that rival the take-your-money-and-run discount house. Now *that's* a combination you can have faith in!



AUTHORIZED SALES AND SERVICE

At First Capitol, our first principle is you!

NEW TOLL-FREE NUMBER!

1-800-TO-BUY-IT



for orders and quotes

1-314-946-1968  
for questions and technical support



Available direct from First Capitol Computer. Please add \$2 minimum (or 2%, whichever is greater) for shipping and handling. If shipped to a Missouri address, please add appropriate sales tax.

First Capitol Computer is a division of Software Wizardry, Inc.

1106 First Capitol Drive St. Charles, MO 63301 (314) 946-1968

First Capitol has over 500 items of interest to Heath/Zenith users, including all Software Wizardry products. Please request our free full line catalog.

**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  
Graphics and Layout  
Assistant ..... Greg Martin  
(616) 982-3463  
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% handling and shipping. 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

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

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

HUG is provided as a service to its members for the purpose of fostering the exchange of ideas to enhance their usage of Heath equipment. As such, little or no evaluation of the programs or products advertised in REMark, the Software Catalog or other HUG publications is performed by Heath Company, in general and HUG in particular. The prospective user is hereby put on notice that the programs may contain faults the consequence of which Heath Company in general and HUG in particular cannot be held responsible. The prospective user is, by virtue of obtaining and using these programs, assuming full risk for all consequences.

REMark is a registered trademark of the Heath/Zenith Users' Group, St. Joseph, Michigan.

Copyright © 1984, Heath/Zenith Users' Group



## on the stack

<b>Buggin' HUG</b> .....	5
<b>Western Regional HUG Conference</b> .....	9
<b>Practical File Management</b> <i>David E. Warnick</i> .....	11
<b>Spreadsheet Corner - Part 4</b> <i>H.W. Bauman</i> .....	15
<b>The FORTRAN Formula-1</b> <i>Dick Stanley</i> .....	21
<b>Introduction to Data Structures - Part 3</b> <i>Emily A. Yount</i> .....	25
<b>"My Favorite Subroutines"</b> .....	33
<b>New HUG Products</b> .....	34
<b>Local HUG Club News</b> .....	38
<b>How To Make Color Lecture Slides</b> <b>By Photographing The CRT Display</b> <i>Robert J. Telepak, M.D.</i> .....	39
<b>The Computers Are Coming . . . Are Here</b> <i>Louise B. Guest</i> .....	41
<b>Z-BASIC Patch</b> <i>Pat Swayne</i> .....	43
<b>Z80 Speedfix For H8/H89</b> <i>Frederick F. Freeland, Jr.</i> .....	45
<b>CHECKSUM: A Program Proofreading Aid</b> <i>S.A. Jacob</i> .....	53
<b>Advanced Assembly Language Programming</b> <i>Pat Swayne</i> .....	57

**ON THE COVER:** A ZBASIC Halloween picture provided by Ed Byrnes of Intuitive Logic, Rochester, Michigan. Ed, thru Intuitive Logic, produces CAI programs for youngsters for use on the H/Z-100 computers.

# 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 \_\_\_\_\_

**PTP-100 (for Z100) @ \$99. or PTP-100 (Z100 Upgrade) @ \$50.\***

- CP/M-86 (replaces CP/M-85)  
 ZDOS

Qty \_\_\_\_\_ Amount \_\_\_\_\_

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

**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 \_\_\_\_\_

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

# BUGGIN' HUG



## Reseeding ZBASIC

Dear Hug,

This should satisfy Mr. Harvey's (Buggin HUG, June '84) request for a simple method for reseeding the ZBASIC random number generator.

```
10 RANDOMIZE(TIME/3)
```

TIME returns a value from zero to 86400. TIME/3 ensures a seed within the allowable limits, i.e. -32768 to +32767.

George Holt  
403 2nd St. West  
BAFB, LA 71110

## Fixn' Business Graphics

Dear HUG,

Responding to the request in the July issue from Sobantu Ndimande: There are at least two problems with the Interactive Business Graphics package on the H/Z-100 Demonstration Disk. First, the package will not accept new data, and second, the graphics produced by some parts of the package are scaled incorrectly.

To fix the data input problem, change the following in MENU.BAS:

1. Renumber line 50070 as line 50071.
2. In line 50080, change the two 50070s to 50071s.
3. Insert the new line 50070 KEYIN\$=INKEY\$.

To fix the problems in the bar chart display, make the following changes in B.BAS:

1. Replace line 170 with 170 XEND=577.
2. In line 180, change 20 to 36.
3. In line 480, change the assignment statement for EACHAREA to EACHAREA=((XPLACES/NRECORDS)\XSCAN)\*XSCAN.
4. In line 780, delete -EACHAREA-BLANKAREA.
5. In line 790, insert -YSCAN\2 after YSTART.
6. Replace line 950 with  
950 SPX=XSTART+(J-1)\*((EACHAREA+BLANKAREA)\XSCAN)\*XSCAN.

To fix the problems in the line chart display, make the same changes to L.BAS as listed above for B.BAS and, in addition, in lines 1012 and 1020, change EACHBAR to EACHAREA/2.

To fix the problems in the trend line chart display, make the same changes to T.BAS as listed above for L.BAS. Also, in line 1060, change the two occurrences of EACHBAR to EACHAREA/2 and, in line 1040, replace the assignment statement for SPX with the one given above for line 950.

The pie chart program plots an ellipse on the screen rather than a circle. However, if you do a graphics dump of the screen to a printer using the utility in the Z-UTIL package from Lindley Systems, the pie will plot correctly as a circle.

I do not have color chips in my Z-100, so I am not able to test the 3

dimensional bar chart and multiple pie chart programs. I have not been able to get the side bar chart program (S.BAS) to plot correctly.

Sincerely,

Craig W. Kirkwood  
1960 East Vaughn Street  
Tempe, AZ 85283

## Press Release

The prize for guessing the number of SUNFLOWER seeds in the bottle at the July HUG convention was \$50.00 toward the purchase of any merchandise listed in our current catalog.

There was a tie between Mr. Mike Wilson of O'Fallon, Illinois and Ms. Lori Theodore of Hammond, Indiana. They both guessed 3500 and were the closest to the correct number of 3482. They will both receive a \$25.00 gift certificate toward any purchase from SUNFLOWER.

Congratulations to Mr. Wilson and Ms. Theodore and our sincere thanks to all who visited our booth and participated in the drawing.

Sunflower Software, Inc.

## H/Z-100 Benchmark Primer

Dear HUG,

Well, you have a Z-100 with Basic and you wonder how you can increase the speed at which your programs run. Towards this end we've run a standard benchmark prime number program, the Sieve of Eratosthenes, using several versions of some of the more popular languages. In addition to testing the performance of the Z-100 using these languages, we also installed a "Z-100 Speed Module" that supposedly increases execution speed by 50%.

All of the compiled languages tested run much faster than interpreted Basic. Any one of these languages would offer a significant advantage over Basic in execution time. Even the slowest of the compiled programs, compiled using the IBM Pascal compiler, executed 26 times faster than interpreted Basic. The fastest compiled program, produced using MASM, executed 425 times faster.

We note that it may not be fair to compare code produced by the IBM compiler, since it is a 1981 product and more recent versions might produce smaller and faster code. This just happened to be the version available to us for testing.

Normally, assembly language programs produce compact code that occupies much less disk space than equivalent programs written in higher level languages. In this case, however, the assembly language version is unusually large because of the way the program is written.

Next, we want to comment on the use of the "Z-100 Speed Module." It works! The execution speeds of all versions of the prime number program increased by 50%. The cost of this module was \$49.95 at HUGCON '84 from C.D.R. Systems, Inc. (619-560-1272). Note that the device may not work with all Z-100's.

Source code listings for the C, Pascal, and Basic programs were from Gilbreath, J, and Gilbreath, G., "Eratosthenes Revisited," BYTE, January 1983, Page 283. The Basic program is given in Listing 1. The assembly language program was from Lafore, R., *Assembly Language Primer for the IBM PC & XT*, New American Library, 1984,

Richard and Janet Hirsch  
470 Belleview  
Webster Groves, MO 63119

**Listing 1.** Prime Number Program in Basic.

```
10 REM eratosthenes sieve prime number program in basic
20 DEFINT A-Z
30 DIM FLAGS(8191)
35 TIMES="0.00"
40 PRINT"10 iterations"
50 FOR M=1 TO 10
60 COUNT =0
70 FOR I=0 TO 8191
80 FLAGS(I) =1
90 NEXT I
100 FOR I=0 TO 8190
110 IF FLAGS(I)=0 GOTO 200
120 PRIME =I+I+3
130 REM print prime
140 K=I+PRIME
150 WHILE K<=8190
160 FLAGS(K) =0
170 K=K+PRIME
180 WEND
190 COUNT = COUNT+1
200 NEXT I
210 NEXT M
220 PRINT COUNT," primes"
225 PRINT TIMES;" to execute program"
230 END
```

**Gripes on FORTRAN**

Ladies and Gentlemen:

I have recently purchased a Zenith Z-100 computer. I enjoy programming in both ZBASIC and FORTRAN. However, I have some gripes about the FORTRAN software.

First of all, it does not have a text editor. This is only a slight inconvenience, since I can input and edit the source program with any word processing package. The main problem I have is that the source file lines are not numbered. If an error occurs during compilation, the compiler will give a line number. The compiler seems to assign the first line of the program the number one and increases each line number by one, except that it does not assign line numbers to the continued lines. So when an error occurs, the compiler gives the line number, but the hard-copy of the program listing is unnumbered. Hence, it is frustrating and time consuming to locate the line in error. I do make many programming errors, hence, this line referencing problem is a major source of frustration to me while debugging. I don't believe this problem is addressed in the FORTRAN manual. Can anyone suggest a method to get line numbers on my source file and have the compiler reference those same lines? Help!

Sincerely yours,

Ray Battalora  
Rt. 7, Box 40  
Covington, LA 70433

**Dumping to a MPI**

Dear HUG,

In response to Timothy Ross's article "OKIDUMP.BAS" on Page 90 of the June issue of REMark, here is the "MPIDUMP.BAS" for Z-100

and MPI-99G dot matrix printer.

MPIDUMP.BAS copies the basic idea from OKIDUMP.BAS, but with a little different approach. The result is 324 seconds for a full screen dump.

If MPI-99G has the 7-bit graphic pattern, as OKIDATA 92 does, this program would take only about 280 seconds to dump a screen.

For those MPI-99G owners, if you happen to hate the AP-PACK as I do, try this one. And please read Timothy's article along with this program, Timothy did a very good job in explaining the fundamentals.

The main difference between Timothy's and mine, is I use PEEK to get the specific byte. The bytes of the green plane start from the 7th byte from the COLARRAY(0) and every 3rd follows it (225 bytes total). These values are defined in line 1080.

If you want to introduce any variables after line 1080, please initialize them before line 1080. Otherwise, you will get incorrect base address of COLARRAY. You can find more details about this on Page 10.173 of ZBASIC manual, Vol. 2.

The way to use this program is to use

```
80 RUN"A:MPICUMP" ' or RUN"B:MPIDUMP" if B: is where you
                  ' kept the file at any place you just
                  ' finished a great picture on the screen
```

instead of

```
80 BSAVE "B:CIRCLE",0,&H0000
```

as shown on Page 93 of the June issue. This will save you 50K bytes of disk space.

```
1000 DEFINT A-Z : DIM COLARRAY(339) : ESC$=CHR$(27)
1020 ADDR=0 : COL=0 : COLSTEP=6 : BYTE=0 : BITPATTERN=0
      : TOPBYTE=0 : BOTBYTE=0
1040 SETUP$=ESC$+CHR$(30)+ESC$+CHR$(19)+
      ESC$+CHR$(22)+STRING$(4,10)
1050 ' this set up will give you the closest
      aspect ratio to the screen's
1060 GRAPH$=ESC$+CHR$(23) : LMARGIN$=STRING$(75,64) :
      GRAFCRLF$="6"
1080 ADDR=VARPTR(COLARRAY(0)) : TOPBYTE=(ADDR+6) :
      BOTBYTE=TOPBYTE+672
1100 OPEN "0",1,"1pt1:" : PRINT #1,SETUP$;
1120 ' first slice is special, we only
      care for left 4 bits
1140 GET (0,0)-( 7,224),COLARRAY
1160 PRINT #1,GRAPH$;LMARGIN$;
1180 FOR BYTE=BOTBYTE TO TOPBYTE STEP -3
1200 BITPATTERN=PEEK(BYTE)\16+64
      'right shift 4 bits, set graphic bit
1220 PRINT #1,STRING$(2,BITPATTERN);
1240 NEXT BYTE : PRINT #1,GRAFCRLF$;
1260 ' following slices are simple,
      the highest bit is ignored by MPI printer
1280 FOR COL=2 TO 632 STEP COLSTEP
1300 GET (COL,0)-(COL+7,224),COLARRAY
1320 PRINT #1,GRAPH$;LMARGIN$;
1340 FOR BYTE=BOTBYTE TO TOPBYTE STEP -3
1360 BITPATTERN=PEEK(BYTE) OR 64
      'set graphic bit
1380 PRINT #1,STRING$(2,BITPATTERN);
1400 NEXT BYTE : PRINT #1,GRAFCRLF$;
1420 NEXT COL : PRINT #1,CHR$(12);ESC$;
      CHR$(21);ESC$;CHR$(20);
1440 END
```

Again, thanks to Timothy's good job.

Dave Hoo  
547 Hawthorn Lane  
Winnetka, IL 60093

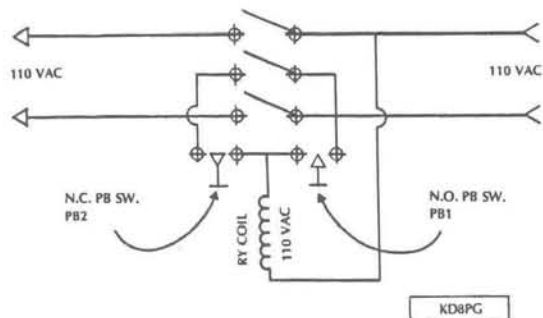
**Anti-Computer-Zapper**

Dear HUG:

Thought possibly others might save valuable programs by the installation of the FAIL SAFE control, which I have used since buying a Heath Computer in 1977.

I have enclosed a diagram of the FAIL SAFE unit, which can be easily assembled from available components. The parts required are as follows:

- A. A 3 x 5 x 7 inch aluminum chassis and cover.
- B. A 3 pole single throw relay with 110 VAC coil.
- C. A push button type switch - contacts normally open.
- D. A push button type switch - contacts normally closed.
- E. Output socket - 110 VAC cord and plug.



Operation of the device is as follows. Pressing PB1 (the normally open switch) actuates the relay feeding power to the computer, disk drives, and printer. To disconnect power to these units press PB2.

The FAIL SAFE feature is that should the power fail, the relay will drop out, thus disconnecting power to all of the units. Power will not come back on until PB1 is again pressed.

Sincerely,

D.C. French  
3396 Adaline Drive  
Stow, Ohio 44224

**FORTRAN Reader Responds**

Dear William:

This is in response to your inquiry in the August, 1984 issue of REMark regarding the problem with the line feeds.

FORTRAN uses the first character on the print line as carriage control and you have come close to solving that problem with the "X" in the FORMAT, but it should either be 50 FORMAT(' .2F10.5) or 50 FORMAT(1X,2F10.5), either will print a space as the first character. Your "X" causes the X to appear in column 1, which is ignored when it appears on the screen since FORTRAN uses that column directly. However, when printing to disk, the same form is used except that when you list the disk file, it is no longer under the control of a FORTRAN program, thus making column 1 a "normal" character.

The other thing that you noticed is that only a carriage return is written to the disk. This is normal for a CP/M system with almost any printer driver supplying the additional line feed. Under HDOS, only the line feed is saved with the driver providing the carriage return.

FORTRAN recognizes several carriage control characters, most commonly the 0, 1, and +. A 0 printed in column 1 (FORMAT('0',...)) causes double spacing (0D 0D written to disk) while a + (FOR-

MAT('+',...)) suppresses all carriage control for that FORMAT statement. This last can be handy for overprinting or concatenating strings of data, just remember, somewhere along the line, to insert your own carriage control. The 'most interesting' carriage control character is the 1 which causes a page tab (FORMAT('1',...)). The fun starts if you are trying to print a number with the first digit of 1 and you forget that column 1 is carriage control (i.e. if NUM=123 and you use FORMAT(I3)). The paper will fly to the top of the next page and 23 will be printed. This is particularly thrilling if it is in some kind of a loop. (i.e., assuming unit 5 is the printer.)

```
DO 33 J=100,199
33 WRITE(5,72)J
72 FORMAT(I3)
```

That's 99 page tabs with 01, 02, 03, ..., 99 printed at the top of each successive page. Better yet is the following:

```
DO 29 J=1,3200
29 WRITE(5,77)
77 FORMAT('1')
```

This little loop is sometimes called "skip to top of box," and can be a bit hair raising. I teach FORTRAN and Pascal programming and it is usually obvious when a student forgets about FORTRAN's carriage control characteristics.

If you, or anyone else, have other questions, feel free to drop me a note.

Sincerely,

Roy Coleman  
10549 S. Homan Ave.  
Chicago, IL 60655

Vectored to 63

**EMULATE**

Let your H89 read and write to:

- OSBORNE XEROX MORROW
  - CROMEMCO EPSON TELEVIDIO
  - DEC VT180 ACTRIX TRS-80
- and others — over 20 formats

Requires HEATH CP/M 2.2.03, 2.2.04 or CDR BIOS.

Include your CP/M serial number when ordering.

- For H37 controller ..... \$59
- For CDR controller ..... \$39

**AUTOMATIC KEY REPEAT for H89/H19**  
simple installation —

Kit . . . \$32 Assembled . . . \$40

**REAL TIME CLOCK for H89**

Kit . . . \$55 Assembled . . . \$65

Check our discount prices on products from:  
**CDR Systems and The Software Toolworks®**

Call or write for catalog. CA Residents add 6% tax.  
**WE PAY POSTAGE Specify Disk Format on Software.**

**ANALYTICAL PRODUCTS 714/929-6919**  
40793 Gibbel Road Hemet, CA 92343

# We give you our finest!



## WIN an H-151 PC Computer System!

HUGGIES everywhere, the perfect partners at your Heathkit Electronic Centers are really giving their best this time! One top-quality H-151 PC computer to be awarded at each of the two big HUG conferences in November.

We offer the kind of support you want. A friendly, trained staff that speaks your

language. Plenty of software and great hardware. Plus qualified in-store service on Heath/Zenith equipment.

So, stop by the local stores' booth at either conference. We're showing our support with an H-151 prize donation. And we hope you're one of the lucky ones to win!



### Join us at...

**Capital HUG Conference,  
November 3,  
Arlington, VA**

**Western Regional HUG Conference,  
November 10 & 11,  
Anaheim, CA**

## Heathkit®

Electronic  
Center\*

Where you get more by doing

\*Units of Veritechnology Electronics Corporation in the U.S.

VEC-856



# Western Regional HUG Conference

## November 10-11, 1984

HUGgies who attended the International HUG Conference this year know the feeling! The feeling of meeting other HUGgies from across the country. The feeling of meeting and talking to Heath/Zenith related vendors, whose products they use. The feeling of meeting the people from National HUG, and the feeling of joy after buying that longed-for product at a special low price. But many HUGgies, especially from the West, have not been to Chicago and don't know that feeling. So why not have a HUG Conference in the West?? That's what the Western Regional HUG Conference is all about.

The Western Regional HUG Conference will be held November 10-11, 1984, at the Disneyland Hotel in Anaheim, California. Patterned after the International Conference, the Western Regional HUG Conference hopes to bring that flavor to the West. The HUGgies will be there! The Heath/Zenith related vendors will be there! The "conference specials" will be there . . . and lots, lots more.

There will be a full program of speakers, some of them better known than others, covering a variety of topics. Speakers include Jim "Blackmax" Buszkiewicz on the HUG-SIG Bulletin Board, and Walt Bilofsky on quality software. Ron Johnson and Wayne Wilson of Heathkit will demonstrate and discuss HERO Jr. The Special Guest Speaker will be Adam Osborne, whose influence on the microcomputer industry and the microcomputer publishing industry is well known. He's an exciting speaker. There will be a panel discussion that includes Adam Osborne and Walt Bilofsky.

A special feature will be a presentation by Susan Hayes, of the Software Toolworks. Her talk will be addressed TO WOMEN ONLY. The subject is "Computer for Women -- How To Do It." Perhaps this is the answer for the "computer widow."

The National HUG staff will be represented at the conference. Bob Ellerton, Jim Buszkiewicz and others will be there to answer your questions and sign up new members. Here's your chance to meet them "in person."

Many vendors present at the National Conference will come to the Western Regional HUG Conference. They include Software Wizardry, Software Toolworks, Trionyx Electronics, CDR Systems, Inc., Sunflower Software, Husker Systems of Nebraska, Colorworks and many others. Look for special sales. Of course, a Heathkit booth will be open, and we can only guess at what goodies will be offered to HUGgies.

A highlight of the conference will be the dinner on Saturday night. A relaxed and congenial atmosphere will prevail at the Disneyland Hotel, where a delightful meal will be served. But after the meal the excitement will build as the prize winners are announced. Smaller prizes are awarded first, then the grand prize, an H/Z-100 Computer with Winchester drive! Those present at HUGCON III recall the generosity of Heathkit President, Bill Johnson at the prize drawings. We hope that spirit will continue at the Western Regional HUG Conference.

If you are interested in attending, write for registration information to: Conference Coordinator, 1555 North Orange Grove Ave., Pomona, CA 91767.

### Conference Schedule (Preliminary)

(Subject to last-minute changes)

#### SATURDAY, NOVEMBER 10TH

- 9:00 AM -- Vendor Area opens  
10:00 -- BOB ELLERTON, (National HUG)  
"Opening Remarks & Announcements"  
10:10 -- RAY DICK, (Heathkit Western Regional Manager)  
"Remarks"  
10:20 -- JIM BUSZKIEWICZ (National HUG)  
"The HUG-SIG Bulletin Board & HUG Program Submittals"  
11:00 -- GEORGE NAJARIAN (Najay Systems) & KENNETH ADCOCK (Programmer)  
"How to Modify the H-89 and the H-100"  
Noon -- Break for Lunch  
1:30 -- ADAM OSBORNE (Entrepreneur)  
"Looking to the Future"  
2:00 -- WALT BILOFSKY (Software Toolworks)  
"Producing Quality Software"  
2:30 -- CHARLES FLOTO (Editor/Publisher)  
On Current HUG Concerns:  
"Declining User Support from Heath/Zenith"  
3:00 -- PANEL: Osborne, Bilofski, Floto  
"Impact of User Groups in the Microcomputer Industry"  
6:00 PM -- Vendor Area Closed for the Day  
7:00 PM -- 9:00 PM HUG DINNER -  
Prizes, awards & guest speakers  
Dinner Guests include:  
WILLIAM E. JOHNSON, President of Heathkit,  
JOSEPH M. SCHULTE, President of V.E.C.,  
ADAM OSBORNE

#### SUNDAY, NOVEMBER 11TH

- 9:00 -- Vendor Area Re-opens  
9:30 -- BOB ELLERTON (Manager, National HUG)  
"Special Announcements"  
9:45 -- RON JOHNSON & WAYNE WILSON (Heathkit)  
"Capabilities of HERO Jr"  
10:45 -- JOHN STUBBE (Mt. San Antonio College)  
"Adapting Mainframe Ideas to the Micro"  
11:15 -- SUSAN HAYES (Software Toolworks)  
<<< FOR WOMEN ONLY !! >>>  
"Computing for Women -- How To Do It"  
Noon -- Break for Lunch  
Special Session on "Computers in Education"  
1:30 -- LES STAHLER, (AnaHUG)  
Introduction of Panel Participants  
2:30 -- PANEL: "Educational Software"  
(Participants to be Announced)  
4:00 -- Meeting adjourned  
5:00 PM -- Conference Closed

# MORE PRODUCTS . . . MORE QUALITY . . . FOR HEATH/ZENITH COMPUTER USERS

---



**Entertainment  
Programs**



**Data Entry  
and Database  
Utilities**



**Disk Management  
Utilities**

- ★ 19 Products for HDOS users
- ★ 30 Products for CP/M-80 users
- ★ 13 Products for ZDOS users
- ★ 14 Products for Z150/Z160 users with MS-DOS



**Communication  
Utilities**

**? ▶▶▶▶ ANSWERS**

**Decision Management  
Software**



**Business/  
Financial  
Software**

---

## QUALITY COMPUTER PRODUCTS . . . AT A FAIR PRICE

---

### **Generic Software**

P. O. Box 790 - Dept. 104R  
Marquette, MI 49855  
(906) 249-9801  
10 a.m. - 5 p.m. EST



**FREE 1984 SOFTWARE CATALOG  
FOR FASTER DELIVERY  
CALL 906-249-9801**

#### **ATTENTION SOFTWARE AUTHORS**

GENERIC SOFTWARE is interested in high quality and well documented microcomputer software. GENERIC offers professional packaging, and high royalties. If you are interested in making money from the software you develop, then ask for our FREE booklet, **Software Author's Kit**.

#### **DEALER INQUIRIES WELCOME**

If you are a Heath/Zenith Dealer and need software for your customers, call or write for more details about our dealer program.



# Practical File Management

## Part 5 - We Add the Final Modules

David E. Warnick  
RD#2 Box 2484  
Spring Grove, PA. 17362

This article puts the last of the modules in our file handler. We'll add 113 lines of programming which will let us delete records either singly, by the month, or by the whole year. Not only is this a convenient way to handle stocks and bonds, but it also shows various ways of handling options in searching the key field.

Owners of 48K systems may need to select one feature or the other from this month's work. With my 64K system I only have 15926 bytes of RAM left with the program loaded. However, my BIOS is set up for both hard and soft sectored controllers, so it's bigger than some. Anyway, next month we'll remedy the situation by putting the whole program on a diet and saving a whole bundle of RAM. This will give us the room we need to work on files.

Figure 1 is a flow chart of the first part of our deletion sub-routine. We begin with the screen setup. On line 8070 we input an option for single, monthly, or yearly deletions, or permit pressing the red key to exit the deletion mode. This time if the red key is pressed, program execution is sent to another part of the sub-routine rather than back to the menu. Line 8080 tests the input if it was not the red key, and makes sure one of the permitted options was selected. With this done, we set the screen up in the month, date, year fashion this program uses so often.

Lines 8190 and 8200 bypass unnecessary inputs if only a year or a year and a month are required, depending on the option we selected above. The inputs required are then taken on lines 8210-8340 and are displayed at the correct locations on the screen.

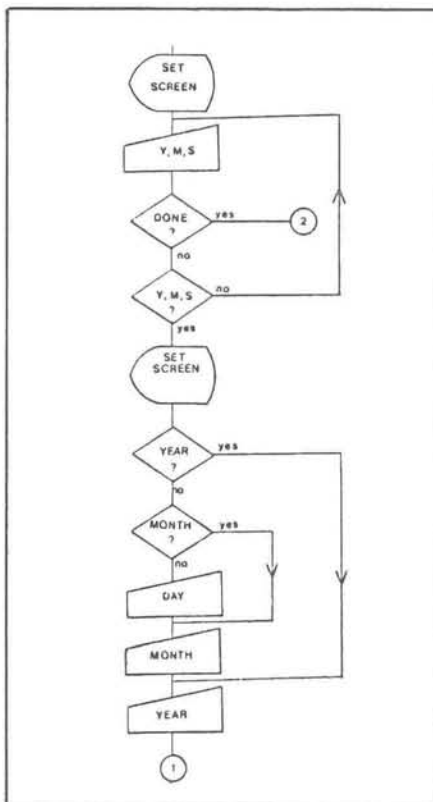


Figure 1

```

8000 PRINT FNCA$(2,1);EE$
8010 PRINT FNCA$(3,36);"DELETE MENU"
8020 PRINT FNCA$(5,21);"Y - DELETE ALL ENTRIES FOR A YEAR"
8030 PRINT FNCA$(7,21);"M - DELETE ALL ENTRIES FOR A MONTH"
8040 PRINT FNCA$(9,21);"S - DELETE A SINGLE ENTRY"
8050 PRINT FNCA$(13,21);"SELECT OPTION FROM THE LIST ABOVE"
8060 PRINT FNCA$(24,21);
      "PRESS THE RED KEY TO RETURN TO THE MENU";FNCA$(1,1)
8070 O$=INPUT$(1):IF O$=CHR$(27) THEN O$=INPUT$(1)
      :IF O$="Q" GOTO 10000
8080 IF O$<>"Y" AND O$<>"M" AND O$<>"S"
      THEN PRINT FNCA$(18,21);
      "YOU MAY ONLY SELECT Y,M, OR S. ENTER AGAIN.":
      FOR X=1 TO 500:NEXT X:PRINT FNCA$(18,1);EE$:GOTO 8060
8090 PRINT FNCA$(2,1);EE$
8100 PRINT FNCA$(3,21);"DELETE OPTION"
8110 PRINT FNCA$(9,21);"MONTH"
8120 PRINT FNCA$(11,21);"DAY"
8130 PRINT FNCA$(13,21);"YEAR"
8140 PRINT GM$
8150 PRINT FNCA$(10,27);"zz"
8160 PRINT FNCA$(12,27);"zz"
8170 PRINT FNCA$(14,27);"zz"
8180 PRINT GO$
8190 IF O$="Y" GOTO 8310
8200 IF O$="M" GOTO 8260
8210 PRINT FNCA$(18,21);
      "ENTER THE DATE TO DELETE AS 2 DIGITS (01-31)"
8220 PRINT FNCA$(11,27);
8230 D$=INPUT$(2)
8240 PRINT D$
8250 PRINT FNCA$(18,1);EL$
8260 PRINT FNCA$(18,21);
      "ENTER THE MONTH TO DELETE AS 2 DIGITS (01-12)"
8270 PRINT FNCA$(9,27);
8280 M$=INPUT$(2)
    
```

```

8290 PRINT M$
8300 PRINT FNCA$(18,1);EL$
8310 PRINT FNCA$(18,21);
      "ENTER THE LAST 2 DIGITS OF THE YEAR TO DELETE"
8320 PRINT FNCA$(13,27);
8330 Y$=INPUT$(2)
8340 PRINT Y$

```

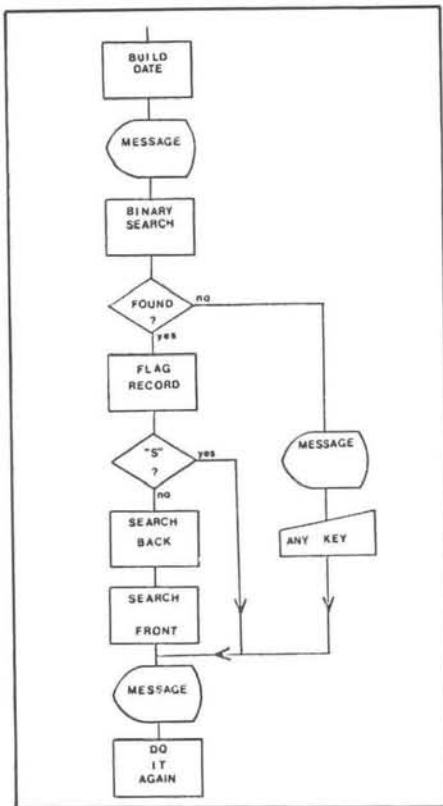


Figure 2

We continue with Figure 2. Lines 8360-8380 build the date to be searched for, depending on the option selected. We then inform the operator that searching is in progress. Any time an operation is likely to take more than a few seconds, it's a good idea to put a message on the screen to tell the operator that something's happening. We're spoiled by the speed of our computers, and a minute with no indication of action can seem like days. With this done, we go into a binary search routine. Lines 8450 and 8460 permit searching the year or the year and month columns of the date field by taking the first 2 or 4 characters from the key field. If you don't understand the MBasic function LEFT\$, you should look up LEFT\$, RIGHT\$, and MID\$ in your manual. They permit selecting part of a string as specified in the parenthesis after the function name.

If no match is found, A% becomes greater than B% and line 8420 directs execution to line 9000 where the operator is informed and permitted to continue. If a match is found, the record number is assigned to KR% (Kill Record) and the operator is informed that the deletion is being made. We don't actually delete the record. Instead, we place an asterisk in the value field as a "FLAG" so it can be skipped when recopying the file at the end of this module. This is done on lines 8530-8550. We can't make any changes to the date field. If we did and further deletions were made, selection of that record could then confuse the binary search.

The deletion we've done is fine if a single entry is to be deleted. Line 8560 directs execution forward to line 8700 if this is the case (option "S" selected). If not, lines 8570-8620 cause the computer to step backward, one record at a time, through the file and compare the key

field to the delete date input by the operator. If a match is found, another flag is inserted. The computer continues to step back through the file until a match is not found. When this happens, lines 8630-8690 do the same thing by stepping forward through the file. This method works because the sorted file has all entries for one month or one year together in the key file, and the key file is used to direct the stepping through the data file. Lines 8580 and 8650 prevent stepping outside the limits of the file.

```

8350 PRINT FNCA$(18,1);EL$
8360 IF O$="S" THEN DR$=Y$+M$+D$:GOTO 8390
8370 IF O$="M" THEN DR$=Y$+M$:GOTO 8390
8380 DR$=Y$
8390 PRINT FNCA$(18,21);
      "NOW SEARCHING FOR THE REQUESTED ENTRY"
8400 A%=1
8410 B%=NR%
8420 IF A%>B% GOTO 9000
8430 C%=INT((A%+B%)/2)
8440 GET #1,B%(C%)
8450 IF O$="Y" THEN R$=LEFT$(A$,2):GOTO 8480
8460 IF O$="M" THEN R$=LEFT$(A$,4):GOTO 8480
8470 R$=A$
8480 IF R$<DR$ THEN A%=C%+1:GOTO 8420
8490 IF R$>DR$ THEN B%=C%-1:GOTO 8420
8500 KR%=C%
8510 PRINT FNCA$(18,1);EL$
8520 PRINT FNCA$(18,21);"ENTRY FOUND - DELETION BEING MADE"
8530 RN%=KR%
8540 LSET B$="*"
8550 PUT#1,B%(RN%)
8560 IF O$="S" GOTO 8700
8570 RN%=RN%-1
8580 IF RN%<1 GOTO 8630
8590 GET #1,B%(RN%)
8600 IF O$="Y" THEN R$=LEFT$(A$,2)
8610 IF O$="M" THEN R$=LEFT$(A$,4)
8620 IF R$=DR$ GOTO 8540
8630 RN%=KR%
8640 RN%=RN%+1
8650 IF RN%>NR% GOTO 8700
8660 GET #1,B%(RN%)
8670 IF O$="Y" THEN R$=LEFT$(A$,2)
8680 IF O$="M" THEN R$=LEFT$(A$,4)
8690 IF R$=DR$ THEN LSET B$="*":PUT#1,B%(RN%):GOTO 8640
8700 PRINT FNCA$(18,1);EL$
8710 PRINT FNCA$(18,21);"DELETION COMPLETED"
8720 FOR X=1 TO 500:NEXT X
8730 GOTO 8000
9000 PRINT FNCA$(18,1);EL$
9010 PRINT FNCA$(18,21);
      "THERE'S NO ENTRY FOR THE DATE YOU REQUESTED"
9020 PRINT FNCA$(20,21);"PRESS ANY KEY TO CONTINUE"
9030 W$=INPUT$(1)
9040 GOTO 8000

```

Figure 3 completes the deletion module. We've found all the records we want to delete and have flagged them with an asterisk in the value field. Then, we pressed the red key and program execution was sent to line 10000. Lines 10000-10040 let the operator know what's going on. Then line 10050 builds a new file name. It has the same drive and file name as the data file, but the extension is .\$\$\$ . This extension is part of the convention for our operating system and indicates a temporary file. We'll read every record from the data file, and check for the flag we put there. If there's no flag, we don't want the record deleted, so we'll write it to the temporary file. The variable RN% keeps track of how many records we write in this manner. This operation is performed by lines 10090-10170. When completed, the temporary file contains all the records we didn't flag for deletion. So, we can close both the data and the temporary files on lines 10230 and 10240. We then KILL the data file. This is the same as your operating system command, which ERAses or DELETes files from the disk. It's no longer there. (Actually, it's there until we write over it. It was just removed from the disk directory.) We then use the NAME

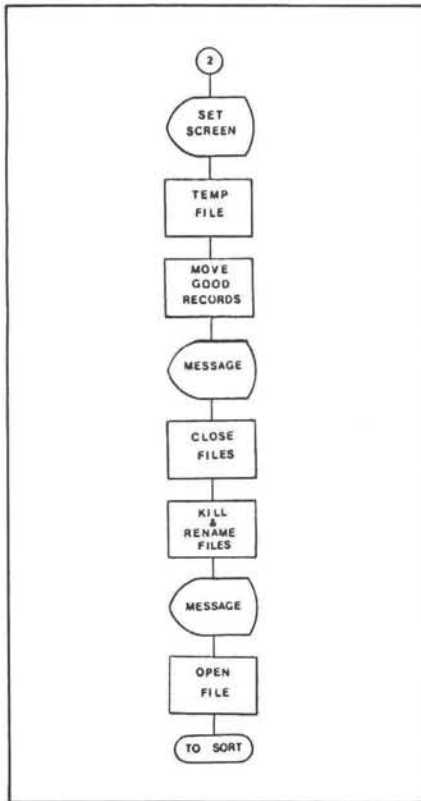


Figure 3

command to change the name of the temporary file to the name of the data file. A check of your disk directory will show that only a .DAT file exists. The .\$\$\$ file is gone because we changed its name. The original data file is gone because we KILLED it. We could have saved the old data file for archive purposes by changing line 10250 to read:

```
10250 NAME FL$ AS D1$+" "+F$+" ".BAK"
```

It would then exist on the disk as a backup file. Again, this is acceptable to our operating system and is part of the extension conventions shown in the manual.

We wrap up the module by printing an operator message and reopening the data file on lines 10310 and 10320. Then we direct execution to the sort module. There's no need to go to the menu first as a sort is always required on a file which has been rearranged this way.

```

10000 PRINT FNCA$(2,1);EE$
10010 PRINT FNCA$(3,21);
      "ALL RECORDS TO BE REMOVED HAVE BEEN MARKED"
10020 PRINT FNCA$(5,21);
      "NOW COPYING NEW FILE WITHOUT DELETED RECORDS"
10030 PRINT FNCA$(8,21);"--- PLEASE WAIT ---"
10040 PRINT FNCA$(11,21);"NOW WORKING ON RECORD NUMBER"
10050 TF$=D1$+" "+F$+" ".+"$$$"
10060 OPEN "R",#3,TF$,10
10070 FIELD #3,6 AS D$,4 AS E$
10080 RN%=1
10090 FOR X%=1 TO NR%
10100 GET #1,X%
10110 IF LEFT$(B$,1)="*" GOTO 10170
10120 LSET D$=A$
10130 LSET E$=B$
10140 PUT #3,RN%
10150 PRINT FNCA$(11,50);RN%
10160 RN%=RN%+1
10170 NEXT X%
  
```

```

10180 NR%=RN%-1
10190 PRINT FNCA$(18,1);EE$
10200 PRINT FNCA$(18,21);"ALL RECORDS TRANSFERRED"
10210 PRINT FNCA$(20,21);
      "DISK FILE OPERATIONS BEING PERFORMED"
10220 PRINT FNCA$(22,21);"--- PLEASE WAIT ---"
10230 CLOSE #1
10240 CLOSE #3
10250 KILL FL$
10260 NAME TF$ AS FL$
10270 PRINT FNCA$(18,1);EE$
10280 PRINT FNCA$(18,21);"DELETION OPERATION COMPLETE"
10290 PRINT FNCA$(20,21);"SORT OPTION SELECTED AUTOMATICALLY"
10300 FOR X=1 TO 500:NEXT X
10310 OPEN "R",#1,FL$,10
10320 FIELD #1,6 AS A$,4 AS B$
10330 GOTO 20000
  
```

We complete our program by adding the lookup module. This permits us to specify key data from the console and display its associated data on the screen. In our case, if the date requested doesn't exist in the file, the entries immediately before and after it will be displayed.

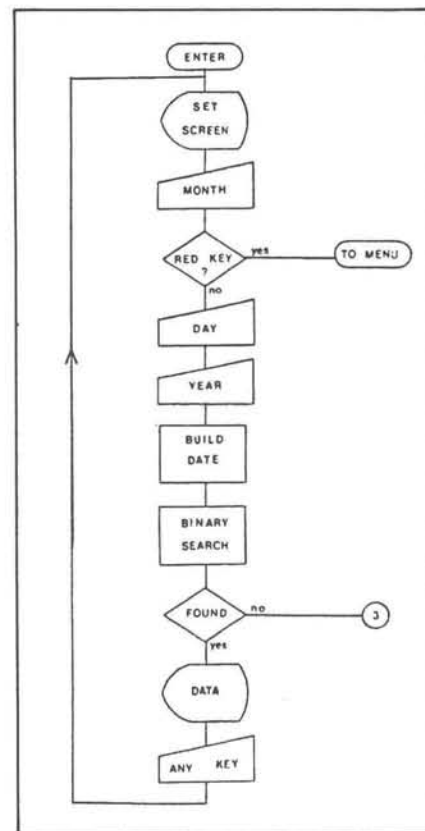


Figure 4

Figure 4 begins our lookup module. We set up the screen and take inputs for the date to be looked up. Again, the red key provides our exit to the menu. Lines 11280-11350 provide the binary search routine to find the record we want. If a match cannot be found, line 11300 directs program execution to line 12000.

If the date we selected is found, the data is displayed by lines 11360-11380. Because we don't know how long the operator will want the information displayed, we stop program execution with a "PRESS ANY KEY" message on line 11380 and the INPUT\$(1) function on line 11390. When the operator is finished with the data, pressing any key permits program execution to continue. The program then loops back to line 11000 to perform the lookup operation again.

```

11000 PRINT FNCA$(2,1);EE$
11010 PRINT FNCA$(3,21);"LOOKUP OPTION SELECTED"
11020 PRINT FNCA$(9,21);"MONTH"
11030 PRINT FNCA$(11,21);"DAY"
11040 PRINT FNCA$(13,21);"YEAR"
11050 PRINT GM$
11060 PRINT FNCA$(10,27);"zz"
11070 PRINT FNCA$(12,27);"zz"
11080 PRINT FNCA$(14,27);"zz"
11090 PRINT GO$
11100 PRINT FNCA$(24,21);
      "PRESS THE RED KEY TO RETURN TO THE MENU";FNCA$(1,1)
11110 PRINT FNCA$(18,21);
      "ENTER THE MONTH TO LOOKUP AS 2 DIGITS (01-10)"
11120 PRINT FNCA$(9,27);
11130 M$=INPUT$(2)
11140 IF M$=CHR$(27)+"Q" GOTO 1000
11150 PRINT M$
11160 PRINT FNCA$(18,1);EE$
11170 PRINT FNCA$(18,21);
      "ENTER THE DAY TO LOOK UP AS 2 DIGITS (01-31)"
11180 PRINT FNCA$(11,27);
11190 D$=INPUT$(2)
11200 PRINT D$
11210 PRINT FNCA$(18,21);EL$
11220 PRINT FNCA$(18,21);
      "ENTER THE LAST TWO DIGITS OF THE YEAR TO LOOK UP"
11230 PRINT FNCA$(13,27);
11240 Y$=INPUT$(2)
11250 PRINT Y$
11260 LU$=Y$+M$+D$
11270 PRINT FNCA$(18,1);EL$
11280 A%=1
11290 B%=NR%
11300 IF A%>B% GOTO 12000
11310 C%=INT((A%+B%)/2)
11320 GET #1,B%(C%)
11330 IF A%<LU$ THEN A%=C%+1:GOTO 11300
11340 IF A%>LU$ THEN B%=C%-1:GOTO 11300
11350 RN%=C%
11360 PRINT FNCA$(18,21);"THE VALUE OF ";F$;" ON"
11370 PRINT FNCA$(19,21);M$;"-";D$;"-";Y$;" WAS ";CVS(B%)
11380 PRINT FNCA$(22,21);"PRESS ANY KEY TO CONTINUE"
11390 W$=INPUT$(1)
11400 PRINT FNCA$(18,1);EE$;FNCA$(9,26);LE$;FNCA$(11,26);
      LE$;FNCA$(13,26);LE$
11410 GOTO 11100

```

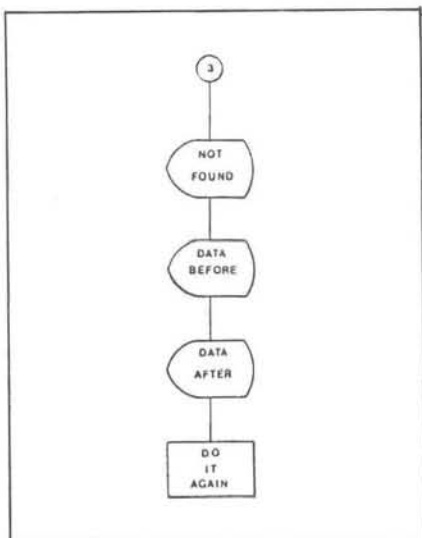


Figure 5

Figure 5 shows the action taken when the lower limit of the binary search range, A%, becomes greater than the upper limit, B%. This only happens if the data we're looking for doesn't exist in the data file. Line 12000 informs the operator that the requested information

is not in the file. Lines 12020-12050 look up the last entry before the requested date and display that information. If B% was zero, the requested date was prior to any in the file and line 12020 provides an appropriate message.

Lines 12060-12090 provide a similar function for the next entry after the requested date. Lines 12100-12110 provide a delay until the operator presses a key. The program then cycles back to the beginning of the lookup option.

```

12000 PRINT FNCA$(18,21);"NO ENTRY EXISTS FOR ";M$;"-";
      D$;"-";Y$
12010 PRINT FNCA$(19,21);"THE TWO CLOSEST ENTRIES ARE"
12020 IF B%=0 THEN PRINT FNCA$(21,21);
      "NO ENTRY EXISTS BEFORE YOUR CHOICE":GOTO 12060
12030 GET #1,B%(B%)
12040 PRINT FNCA$(21,21);MID$(A$,3,2);"-";RIGHT$(A$,2);
      LEFT$(A$,2)
12050 PRINT FNCA$(21,32);CVS(B%)
12060 IF A%>NR% THEN PRINT FNCA$(22,21);
      "NO ENTRIES AFTER YOUR CHOICE":GOTO 12100
12070 GET #1,B%(A%)
12080 PRINT FNCA$(22,21);MID$(A$,3,2);"-";RIGHT$(A$,2);
      "-" ;LEFT$(A$,2)
12090 PRINT FNCA$(22,32);CVS(B%)
12100 PRINT FNCA$(24,21);"PRESS ANY KEY TO CONTINUE";
      FNCA$(1,1)
12110 W$=INPUT$(1)
12120 GOTO 11400

```

So there you have it. A complete program to establish and maintain a set of random files. We used the value of stocks as the data we wanted to manage. However, this program will keep track of any numeric value which varies with dates. It could be rainfall, high or low temperatures for weather buffs. Or, how about the number of kilowatt-hours used for the energy conscious. Experiment and come up with applications of your own.

Next month I'll show you how to keep all the features of this program while reducing it in size. We'll also discuss modifying and customizing it to your own needs. Since we started discussing file handling 9 months ago, you have added random files and a pseudo-data-base system. Ahead, we've got sequential files and then, who knows. Your suggestions are welcome.

Please remember that these articles and the programming in them are copyrighted by the author. You're more than welcome to use them for your own personal non-commercial applications.

See you next month.



# MOVING?

Please let us know 8 weeks in advance so you won't miss a single issue of REMark!



I will show my four (4) "Spreadsheet Preparation Forms" below (PeachCalc/SuperCalc will look nearly the same, but Multiplan will require renaming the cells) for the LOTUS 1-2-3 software:

FORM #1					FORM #2				
A	B	C	D	E	F	G	H	I	
1							SPREADSHEET COR		
2							+++++		
3							PROFIT & LOSS		
4									
5	1984	JAN	FEB	MAR	1st-Q	APR	MAR	MAY	2nd-Q
6	*****								
7	SALES				S(B7..D7)	+D7*1.08	+F7*1.08	+G7*1.08	S(F7..H7)
8	COSTS				S(B8..D8)	+D8*1.04	+F8*1.04	+G8*1.04	S(F8..H8)
9									
10	G.P.	+B7-B8	+C7-C8	+D7-D8	S(B10..D10)	+F7-F8	+G7-G8	+H7-H8	S(F10..H10)
11									
12	G&A				S(B12..D12)	+D12*1.10	+F12*1.10	+G12*1.10	S(F12..H12)
13	SELLING				S(B13..D13)	+D13*1.17	+F13*1.17	+G13*1.17	S(F13..H13)
14									
15	NET	+B10-B12-B13	+C10-C12-C13	+D10-D12-D13	S(B15..D15)	+F10-F12-F13	+G10-G12-G13	+H10-H12-H13	S(F15..H15)

FORM #3				FORM #4					
J	K	L	M	N	O	P	Q	R	
1	NERCO.								
2	+++++								
3									
4									
5	JUL	AUG	SEP	3rd-Q	OCT	NOV	DEC	4th-Q	YEAR TOTALS
6									
7	+H7*1.08	+J7*1.08	+K7*1.08	S(J7..L7)	+L7*1.08	+N7*1.08	+O7*1.08	S(N7..P7)	+G7+I7+M7+Q7
8	+H8*1.04	+J8*1.04	+K8*1.04	S(J8..L7)	+L8*1.04	+N8*1.04	+O7*1.04	S(N8..P8)	+G8+I8+M8+Q8
9									
10	+J7-J8	+K7-K8	+L7-L8	S(J10..L10)	+N7-N8	+O7-O8	+P7-P8	S(N10..P10)	+G10+I10+M10+Q10
11									
12	+H12*1.10	+J12*1.10	+K12*1.10	S(J12..L12)	+L12*1.10	+N12*1.10	+O12*1.10	S(N12..P12)	+G12+I12+M12+Q12
13	+H13*1.17	+J13*1.17	+K13*1.17	S(J13..L13)	+L13*1.17	+N13*1.17	+O13*1.17	S(N13..P13)	+G13+I13+M13+Q13
14									
15	+J10-J12-J13	+K10-K12-K13	+L10-L12-L13	S(J15..L15)	+N10-N12-N13	+O10-O12-O13	+P10-P12-P13	S(N15..P15)	+G15+I15+M15+Q15

3) Type ↑1ST QUARTER and press Return.

4) Press F5, type E12 and press Return.

**NOTE:** Per our preparation form we must add the G&A Expenses Total and the Selling Costs Total to the 1st quarter column.

5) Press /WGF. type T and press Return.

6) Type @SUM(B12..D12) and press down arrow.

7) Type @SUM(B13..D13) and press Return.

8) Press F5, type E5 and press Return.

**NOTE:** We need to Move the 1st quarter column from our report area for now. We will Move it to an area of the "worksheet" that we will not be using for this assignment, as seen from our preparation form. When we are ready for it, we will Move it from the temporary storage area back to the right column in our "template". Watch for these procedures in the following steps. You will also see that we had to do this so that we could Copy our formulas to the added month's cells.

9) Press /M, type E5..E16 and press Return.

10) Type S5..S16 and press Return.

11) Press F5, type S1 and press Return.

**NOTE:** Did you find where we stored our 1st quarter column over in a unused area of the "worksheet"?

12) Press HOME key.

**NOTE:** The 1st quarter column is gone, right? Now we can add our new labels.

13) Press F5, type E5 and press Return.

14) Type ↑APRIL and press right arrow.

15) Type ↑MAY and press right arrow.

16) Type ↑JUNE and press right arrow.

17) Type ↑JULY and press right arrow.

18) Type ↑AUGUST and press right arrow.

19) Type ↑SEPTEMBER and press right arrow.

20) Type ↑OCTOBER and press right arrow.

21) Type ↑NOVEMBER and press right arrow.

22) Type ↑DECEMBER and press Return.

**NOTE:** I hope that you remember why we typed (↑) as the first character. Do you? If not, press the "HELP" key and you will find that it centers our label in the cell which is 14 characters wide because we set it for 14 for P—LTEMP1.

23) Press F5, type E7 and press Return.

24) Type +D7\*1.08 and press down arrow.

**NOTE:** Our specifications stated that Sales would increase 8% per month starting in April over March. We could use 108% also.



25) Type +D8\*1.04 and press down arrow.

**NOTE:** Again, we said Costs would increase 4%.

26) Press down arrow (3) times.

27) Type +D12\*1.10 and press down arrow.

28) Type +D13\*1.17 and press Return.

**NOTE:** Do you know where the 1.10 and 1.17 came from? Remember we stated that we would have 10% and 17% increases for these items. NOW, we are going to use something new and very important! We are using "COPY FORMULAS" which will save us a lot of typing and thus might keep us from making typing errors. To do this you must learn new "terms". I am not going to go into a detailed explanation of Relative Cell Address vs Absolute Cell Address. I will show you WHAT TO DO and you can use "HELP" key or READ about them in your manual. To use this handy method we could not have the "quarter" columns in our "template" or 1-2-3 would try to Copy the formulas into these columns as well, causing us many errors. Think about this! Can you see why? If you cannot, experiment with these columns in the "template" after we finish this "model" to see what would happen. 1-2-3 can not handle everything, but it does very well.

29) Press F5, type E7 and press Return.

30) Press /C and Press F4 key and press Return.

**NOTE:** When you press the F4 key, 1-2-3 changes the Relative Cell Address to the Absolute Cell Address, it shows this by adding the (\$) signs to the cell names.

31) Type F7..M7 and press Return.

**NOTE:** We want to Copy the formula from E7 (April) into F7 (May) through the "range" of months to M7 (December). Can you see how this works? Check some cells in this row to see how 1-2-3 put the correct formula in each cell. This sure beats typing all those formulas and it reduces the chance for a typing error. LETS give 1-2-3 a "BIG HAND"!

32) Press F5, type E8 and press Return.

33) Press/C, press F4 and Return.

34) Type F8..M8 and press Return.

**NOTE:** Do you want to checkup on 1-2-3 by checking some of the cells on this row to see if 1-2-3 put the correct formula in each cell? Before long you will believe in 1-2-3 and you will know that you KNOW the method!

35) Press F5, type D10 and press Return.

36) Press /C, press F4 and Return.

37) Type E10..M10 and press Return.

**NOTE:** Copy Formula works on this expression as well. We must go over one cell to the left for our "base" cell! Did you catch that?

38) Press F5, type E12 and press Return.

39) Press /C, press F4 and Return.

40) Type F12..M12 and press Return.

41) Press down arrow once.

42) Press /C, press F4 and Return.

43) Type F13..M13 and press Return.

44) Press F5, type D15 and press Return.

45) Press /C, press F4 and Return.

46) Type E15..M15 and press Return.

47) Press F5, type E5 and press Return.

48) Press /WI and Return.

49) Type E5..E16 and press Return.

**NOTE:** We must provide a blank column to Move our 1st quarter data back in the correct column as per our preparation form. This Move is our next step. Do you remember where we stored it on the "worksheet"? If you are not sure, you can "scroll" to the right until you find it. It will be at column T, not S, because we have added a column.

50) Press /M, type S5..S16 and press Return.

51) Type E5..E16 and press Return.

**NOTE:** Did you get the 1st quarter data back into the right column? Check it with your "Spreadsheet Preparation Form". The next steps will provide blank columns for the 2nd and 3rd quarter data. We will enter the required labels and formulas to these columns.

52) Press F5, type I5 and press Return.

53) Press /WI and Return.

54) Type I5..I16 and press Return.

55) Type ↑2ND QUARTER and press down arrow.

56) Press down arrow once.

57) Type @SUM(F7..H7) and press down arrow.

58) Type @SUM(F8..H8) and press down arrow.

59) Press down arrow once.

60) Type @SUM(F10..H10) and press down arrow.

61) Press down arrow once.

62) Type @SUM(F12..H12) and press down arrow.

63) Type @SUM(F13..H13) and press down arrow.

64) Press down arrow once.

65) Type @SUM(F15..H15) and press Return.

66) Press F5, type M5 and press Return.

67) Press /WI and Return.

68) Type M5..M16 and press Return.

69) Type ↑3RD QUARTER and press down arrow.

70) Press down arrow once.

71) Type @SUM(J7..L7) and press down arrow.

72) Type @SUM(J8..L8) and press down arrow.

73) Press down arrow once.

74) Type @SUM(J10..L10) and press down arrow.

75) Press down arrow once.

76) Type @SUM(J12..L12) and press down arrow.

77) Type @SUM(J13..L13) and press down arrow.

78) Press down arrow once.

79) Type @SUM(J15..L15) and press Return.

**NOTE:** We still need the 4th quarter and year total column formulas and labels. The year totals math expressions are the sum of the four (4) quarter column cells for each row item.

80) Press F5, type Q5 and press Return.

81) Type ↑4TH QUARTER and press down arrow.

82) Press down arrow once.

83) Type @SUM(N7..P7) and press down arrow.

84) Type @SUM(N8..P8) and press down arrow.

85) Press down arrow once.

86) Type @SUM(N10..P10) and press down arrow.

87) Press down arrow once.

88) Type @SUM(N12..P12) and press down arrow.

89) Type @SUM(N13..P13) and press down arrow.

90) Press down arrow once.

91) Type @SUM(N15..P15) and press Return.

92) Press F5, type R5 and press Return.

93) Type ↑YEAR TOTALS and press down arrow.

94) Press down arrow once.

95) Type +E7+I7+M7+Q7 and press down arrow.

96) Type +E8+I8+M8+Q8 and press down arrow.

97) Press down arrow once.

98) Type +E10+I10+M10+Q10 and press down arrow.

99) Press down arrow once.

100) Type +E12+I12+M12+Q12 and press down arrow.

101) Type +E13+I13+M13+Q13 and press down arrow.

102) Press down arrow once.

103) Type +E15+I15+M15+Q15 and press down arrow.

**NOTE:** That completes all the formulas for the "template". Carefully check your work with that shown on your "Spreadsheet Preparation Forms". Do they match each other? If they do not, make the corrections required! Now would be a good time to SAVE! I have not suggested the times to SAVE. I left this up to you. I am not going to provide the procedure either. You should know it! Do you? Now lets "dress" up this "template".

104) Press F5, type E6 and press Return.

105) Press /C and Return.

106) Type F6..R6 and press Return.

107) Press F5, type E9 and press Return.

108) Press /C and Return.

109) Type F9..R9 and press Return.

110) Press F5, type E14 and press Return.

111) Press /C and Return.

112) Type F14..R14 and Return.

113) Press F5, type E16 and press Return.

114) Press /C and Return.

115) Type F16..R16 and press Return.

**NOTE:** Lets Move the Company name and Report name to a position nearer the center of our "template". We will use a "block move" by defining a "rectangular range". Use "HELP" key for more information or refer to your manual. See if you can recognize the "block". Can you? We specify the "block" by naming the corners of the "block".

116) Press F5, type B1 and press Return.

117) Press /M, type B1..D3 and press Return.

118) Type H1..J3 and press Return.

119) Press /WGF, type C and press Return.

**NOTE:** Did you find that 1-2-3 had your projections/forecasts already calculated? It has been doing this for you as you entered the formulas. Our "model" covers more than one (1) screen so we must "scroll" to see all parts of the "model". Lets try some new "scrolling" methods.

120) Press F5, type A5 and press Return.

121) Press F11 key (END) and the right arrow.

122) Press F11 key and the left arrow.

123) Press F11 and down arrow.

**NOTE:** Did you see how fast you can travel around on the "model"? However, there appears to be a problem! I would like to keep the right hand column of labels (1-2-3 calls them "Titles") in view on the screen all the time. It would also be nice if we did not "scroll" into these labels or the top label area. 1-2-3 can solve this problem! We will "freeze" (sometimes called "locked" or "protected") the "Titles" areas. To do this we must move the "pointer" in the "cell" one position to the right and one position below the Title areas!

124) Press F5, type B7 and press Return.

125) Press /WT and Return.

**NOTE:** Now try "scrolling" again, like we did above using the F11 key and a arrow key. You should find that we have solved our "problem"! Another problem would be when doing "WHAT IF" changes where you want to see the Year Totals at the same time as you change a particular month's data. We will solve this using the 1-2-3 "window" feature. The "window" can be used many ways, but we will use only one at this time.

126) Press F5, type D7 and press Return.

127) Press /WW and type V.

**NOTE:** We now have a "highlighted" vertical column of Row numbers. Try "scrolling" and you will find that we can "scroll" within the "window". "Scroll" using arrow keys so that the MARCH column is just to the left of the vertical divider.

128) Press F6 key.

**NOTE:** This puts the "pointer" into the right hand "window". Try "scroll" some more using arrow keys.

129) Press F11 key and the right arrow.

**NOTE:** We now have February and March in view in the left "win-

down" and 4th Quarter and Year Totals columns in the right "window" so we can see/watch these columns on the same screen. Lets do a "WHAT IF" and watch the re-calculation in both "windows"! 130) Press F6 key.

131) Press F5, type D7 and press Return.

132) Type 24678 and press Return.

**NOTE:** Did you see the recalculation? We will try another. BE SURE you watch carefully! This is one of the "BIG" features of a Spreadsheet "model"!

133) Type 14678 and press Return.

**NOTE:** Did you see it happen that time? What if we wanted to prevent changes to our original P—LTEMP1 data and allow changes only in our projection/forecast data? We would use the 1-2-3 "Protect" feature!

134) Press /WW and type C.

**NOTE:** The C "cleared" or eliminated the "windows". By-the-way have you SAVED lately? Enough said!

135) Press /WG and type P and press Return.

**NOTE:** This command "ENABLES" Protections (turns it ON)!

136) Press /R and P, type B7..D13.

**NOTE:** We used "range protection" to protect the original data. Did you recognize the "block range" that we specified? We used the corner cells!

137) Press F5, type E7 and press Return.

138) Type 22456 and press Return.

**NOTE:** We have several messages! ERROR in the upper, right corner of the screen and "Protected Cell" in the lower, right corner of the screen, PLUS a "BEEP" from the computer. Thus, we have prevented a change to a cell that contained original data. We will find this very valuable in some of our future assignments! Use "HELP" key and your manual to study this feature. We also have another valuable command -- Status Check!

139) Press ESC key.

140) Press /WS

**NOTE:** We have a Status Report in the upper work area. Check carefully what information appears in this report so that you will know when to use Status Report. In this case we want to see Protect--ON!

141) Press ESC key.

142) Press /WGP and type D.

143) Press F5, type D7 and press Return.

144) Type 22456 and press Return.

**NOTE:** Now we can change this cell because we turned Protection OFF!

145) Press /WS.

**NOTE:** The Status Report shows Protect--OFF!

146) Press ESC key.

147) Type 12456 and press Return.

**NOTE:** We have completed our second assignment--P—LTEMP2 (for 1-2- 3). SAVE and QUIT! You know how!

We have covered a lot of new 1-2-3 Spreadsheet features with this assignment. I would like you to make a "list" of them. If you did not use the 1-2-3 "HELP" screens as we used the new items, repeat the procedure and study these "HELP" screens. Also, please look each of the items up in your 1-2-3 Manual and study each!

## CLOSING

For "homework" I would like to have every 1-2-3 user repeat the above step-by-step procedure until you can do the assignment without the help of this article using only the "HELP" key and your "Spreadsheet Preparation Forms". When you can do that, you will be ready for the next assignment!

I hope that you Readers using Multiplan or PeachCalc/SuperCalc have referred to their Manuals to see how each of the "new" spreadsheet features can be done with their software. If you made the "list" that I suggested above, this will help you to find similar items for your software. Next month I will give you some "hints" of how I did this assignment using Multiplan and PeachCalc. Then you can compare how you did it with my method. BE SURE to try the assignment before next month!

I would like to restate -- "SPREADSHEET Corner" is your space in REMark! If you have any questions, suggestions, constructive criticism, other ways of doing things, etc.; please write them out in detail and send them to the author along with a SASE, business size. PLEASE, NO PHONE CALLS! The author will send you an answer or put the information into a future "SPREADSHEET Corner".



 PAUL F. HERMAN

**DATA SYSTEMS CONSULTANT**

P.O. Box 535

St. James City, FL 33956

**NEW - PERSONAL DATA MANAGEMENT SERIES**

....including ....

- Appointment Calendar
- Mailing List
- Household Inventory
- Periodical Index
- Record Index
- Audio Tape Index
- Vehicle Log
- Recipe List
- Coupon Organizer
- Library Index
- Video Tape Index
- Check Register

All on ONE disc for ONE price. . . \$ **59.95**

**DOODLER** Graphics Package \$79.95

Z100 SOFTWARE

Prepaid orders sent post-paid. Florida residents please include 5 percent sales tax.

Send SASE for additional Info on Programs.

We are an authorized ZENITH Data Systems Dealer.

This entire ad was printed actual size using DOODLER, a Z-100, and a GEMINI-10x printer.

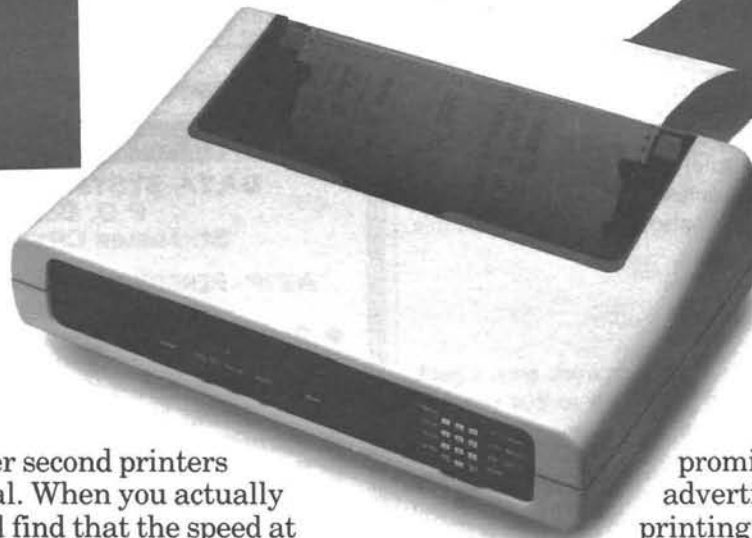
Dealer Inquiries contact - YELLOW ROSE SOFTWARE  
10208 N. 30th Street, Tampa, FL 33612

**ZENITH****data  
systems**

## PRINTERS PRINT CIRCLES AROUND THE COMPETITION

*Actual Speed Test Results*

PRINTER MODEL	STRAIGHT PRINTING	INVOICE PRINTING	BUFFERED PRINTING	RATED SPEED
Zenith Z-125	89 cps	140 cps	82 cps	150 cps
TI-810	123 cps	153 cps	90 cps	150 cps
Data South 180	80 cps	130 cps	77 cps	180 cps
Epson FX-80	82 cps	80 cps	55 cps	160 cps
Star Gemini	63 cps	77 cps	76 cps	100 cps
Okidata 84	126 cps	204 cps	100 cps	200 cps
Zenith 99	55 cps	58 cps	58 cps	100 cps
Zenith 150	113 cps	166 cps	142 cps	150 cps
"S" printer	106 cps	113 cps	111 cps	160 cps



Model shown with  
optional SoftSwitch Keypad

All 150 character per second printers are not created equal. When you actually start running, you'll find that the speed at which the characters appear on the paper depends on many things beside the published rated speed.

When actually tested, the nine printers listed gave some surprising results. See how they measured up in the different functions. "Straight Printing" is printing one full line of data after another. "Invoice Printing" is a test set up with a simulated invoice. "Buffered Printing" used the same data as the invoice test but with some disk accessing in between each invoice. "Rated Speed" is what everyone

promises in product advertising. In actual printing variables like forms advance rate, degree of logic seeking and buffer presence mean a lot.

Prove it to yourself. Visit your local Heathkit Electronics Center\* or Zenith Data Systems dealer and ask for a copy of "Q & A for the Printer Peddler." It explains why one 150 cps printer can be twice as fast as another. You might also get a copy of the Speed Testing program that was used in this testing. So you can prove these figures for yourself. For the name and address of your local Zenith Data Systems dealer call 800-842-9000, ext. 1.

\*Units of Veritechnology Electronics Corporation.

# The FORTRAN Formula-1

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

Wouldn't it be nice if you had a programming language that was similar to the BASIC you already know, but which generated fast-executing true machine code? While we are at it, how about the ability to mix program segments written in different languages? That way, you could use that neat assembly-language screen handler and the Pascal graph program you acquired in your latest effort to beat the stock market. It would help if it were low-cost, too!

Well, read on. The language is FORTRAN, and it is here; it has been for 30 years. Beginning this month, and continuing in REMark every other month, we will explore how FORTRAN can simplify and speed up the programs we write. Even if you have never used the language, stick around. You may find you want to use it. And if you are a FORTRAN veteran, you may want to read through this too; hopefully, there will be something for everyone.

## A Little History

FORTRAN is an acronym derived from the term FORMula TRANslation. It is an algebra-like language that was first developed in the mid-1950's for use primarily by scientists and engineers who were then beginning to use computers to solve complex technical problems. Because of that, the language was structured to look as much like the mathematical notation those people used daily with their slide rules as was possible. At about the same time, COBOL was developing for use within the business community. Both languages were developed to make their use relatively easy for their intended audience: COBOL is English-like, FORTRAN looks just like written algebraic equations.

At this point in computer development, there were two sorts of computers: scientific and business. Business applications, such as payroll preparation, were seen as requiring considerable amounts of input/output (I/O), but not large amounts of computational capability or memory. Scientific applications were characterized by minimal I/O, but large demands for computing power and memory. Computer runs of several days duration to solve complex mathematical problems were not uncommon. And, of course, everything was batch processed -- interactive computing was still a generation away.

Because this type of hardware was the only sort available, the languages were developed to take advantage of its capabilities. They also reflect the hardware limitations. For these reasons, both COBOL and FORTRAN have been roundly condemned by more modern data-processors as being prehistoric and useless. That is a bit of an overstatement. Both languages have limitations, as do all their more modern relatives. But both have some significant advantages, as well. In large part because the language was required to compensate for some hardware characteristics that would today be regarded as fatal. H. W. Bauman has introduced you in these pages to the abilities of COBOL; my plan is to guide you through FORTRAN. Once you have tried it, you won't go back!

## Why FORTRAN?

A good question. There are several answers.

First, because it has been around for so long, there is an enormous amount of program code written in FORTRAN. If you work in science or engineering, you will almost certainly have to work with programs written in FORTRAN. Even if you are an accountant or a bus driver, though, much of this code is now in the public domain, and available to you for the cost of copying. In either case, a knowledge of the language is essential for you to modify it to do what you want, and not spend all your time rewriting the problem solution in some other language.

Second, and also due to its long history, FORTRAN has been thoroughly tested and is virtually bug-free. Or, to put it another way, all the undocumented features are now documented!

Third, there are agreed-upon standards for FORTRAN. Thus, if you write your program in FORTRAN on your machine, you can transport the source code to another machine of totally different architecture, recompile, and run the program. Of course, it is upward-compatible: older versions work on newer versions of the compilers, but not the other way around. This portability of code can be a great asset in program development among several people or locations, especially if they do not all have the same machine. It is also a big help when you change your computer. All you need to get your old, tested programs running on the new machine is a new FORTRAN compiler and the time to recompile the old source code.

Last, FORTRAN gives you a lot of benefits you can't get anywhere else. It is easy to learn, because you have a good idea already about the language structure -- it was the model for BASIC, which is the veritable "standard" on microcomputers. FORTRAN is more powerful than BASIC, and gives you speed and capabilities simply impossible to achieve in BASIC.

## Benefits

FORTRAN is a compiled language. That means that the output of the process is a machine-language file -- a .COM file -- that is ready to run on your computer. You do not need to have any of the compiler resident in memory, or even on disk, when the program runs, as you do in BASIC. So, more memory is available for the program. Preparing a program is slightly more complex than in an interpreted language like BASIC, because you must first compile the source code (Oops! Source code is what you write in the high-level language, like FORTRAN.) Object, or machine, code is what comes out the "back end" of the compiler. It is the translation of your instructions into language the microprocessor can understand. Compilation involves passing your source code through a compiler, just as you would pass a text file through a formatter. The code will not execute immediately, as does BASIC; it must first be compiled. This extra step adds a bit of time to program preparation, but more than pays for itself in time

saved during program execution.

Not all the speed of FORTRAN comes from compilation alone. It is easy to produce slow machine code -- lots of compilers do it every day. But FORTRAN has also been around long enough to have been largely optimized. In other words, it generates efficient code, not merely object code. If you are going to do the same calculation several thousand times, and computer time costs \$100 per CPU minute, efficiency is important. Those were just the conditions that prevailed in the 1950's and 1960's, and FORTRAN evolved to minimize the cost of such calculations.

You don't repeat calculations thousands of times, though, do you? Oh? Consider the relatively simple case of calculating and printing an amortization schedule for the 30-year mortgage on your home. You will go through the calculating loop 360 times (30 years times 12 months per year), and each calculation will require the calculation of several quantities raised to a power. The computer does that by calculating the value of a mathematical series -- adding up a large number of terms to approximate the value of  $X^{**N}$ . Each payment calculation requires dozens of the same computations done repetitively, and each set done 360 times! Even this common application requires several thousand repetitive calculations. More complex applications require even more.

It is precisely at this point that BASIC starts slowing down. Because it is interpreted, each statement must be translated into machine code each time it is accessed. That takes time. The more calculations, the more time is required.

There is such a thing as a BASIC compiler, though. Changing from interpreted to compiled BASIC will greatly speed up program execution. However, it still will not be as fast as a FORTRAN program, because compiled BASIC has more overhead (code that must be compiled, but is not used in the program) than does compiled FORTRAN. More about this later. More significantly, it is still BASIC. The powerful and flexible features of FORTRAN simply are not available in BASIC, compiled or interpreted.

A word of caution at this point. Merely compiling source code is not the only thing involved in gaining speed. Like anything else, there are good compilers and there are not-so-good compilers. Slow, inefficient FORTRAN compilers can be built: the IBM PC FORTRAN has achieved a reputation for being unusably slow, to name one. We will use tested, reliable, fast compilers in this series. Why use anything else?

To get an idea of the types of speed improvements we can achieve, consider the results of some benchmarks which have been run in a variety of languages on Heath/Zenith computers. Benchmarks can fool you in some circumstances, but they can also be helpful for gross comparisons. Take a look at the data in Figure 1.

The FORTRAN compiler is written so that only those routines that actually are used by the program you write are included in the source code. If your program does not calculate the square root of anything, the code for calculating the square root will not be included in your object code. This substantially reduces program overhead, and contributes to speed in the compiled code. It means that the FORTRAN compiler must have a library of routines to refer to, so that it can select the ones it needs, and this library is usually included with the compiler. Because of the separation of the library and the compiler, it is possible to tailor particular routines to your own needs, and even to write library routines which do things you require that were not included with the compiler. The need for this library may appear on the surface to be an impediment, but it actually provides a surprising amount of flexibility to the FORTRAN programmer. Later in this

series, we will examine some public domain library routines which increase program execution speed even beyond that shown by the benchmarks below.

Machine	Operating System	Language	Run Time (seconds)	PerCent Slower than Microsoft FORTRAN
Z89	CP/M	ECMBASIC	3056	11,281%
Z89	CP/M	JRT Pascal	807	2,952%
Z89	CP/M	Compiled MBASIC	37	37%
Z89	CP/M	Microsoft FORTRAN	27	--
Z100	Z-DOS	Z-BASIC	2057	14,593%
Z100	Z-DOS	IBM FORTRAN	618	4,314%
Z100	CP/M-85	MBASIC	1710	12,114%
Z100	CP/M-85	JRT Pascal	409	2,821%
Z100	CP/M-85	Compiled MBASIC	18	29%
Z100	CP/M-85	Microsoft FORTRAN	14	--

**Figure 1.** Benchmark results of various languages using the "Sieve of Eratosthenes" as presented in BYTE, January, 1983. Data originally printed in BUSS #69, May 26, 1983.

We have already mentioned the portability of FORTRAN source code as a result of national standards. Not only can you easily move your own code to another machine (as when you upgrade your computer), you can also import other people's code (remember all that public domain FORTRAN code?) to your machine and compile and run it with minimal trouble! There is a large number of excellent solutions to common scientific and engineering problems in the public domain, and that fact alone can save you hundreds of hours of coding, debugging, and testing. If someone else has previously programmed the solution to your problem, why should you do it again?

The current "buzz-word" in modern data processing is "software engineering." A major feature of software engineering is the ability to reuse code that was originally written in another language than the one you are working in now, so that routines need not be rewritten merely to translate them when the programming language changes. Not a bad idea. FORTRAN has that capability!

True, BASIC lets you access assembly-language routines from within the BASIC program, but it is awkward, and requires not a little knowledge of specifics of the machine to do it. FORTRAN, on the other hand, lets you link in program segments written in virtually any other language when you compile the program. There are some rules, of course, to ensure that data and parameters are correctly passed between segments, but the essential fact is that you really CAN use your Pascal graph generator and your assembly-language keyboard handler with your FORTRAN program. That can be a big savings in coding time, but the real savings is in the common interfaces you work with: data can always be input from essentially the same routine, with the same command structure; graphs can routinely be prepared using the same commands and parameter routines; etc. Each time you don't have to learn a new technique or syntax, the time it takes to implement it goes down and your accuracy goes up.

FORTRAN source code is written with a text editor: any one you like. I happen to prefer the non-document mode of WordStar, others swear by PIE, VEDIT, or their particular favorite. The point is, the program doesn't care. Once you have learned the editing commands peculiar to your favorite editor, you needn't learn another set to edit program code. You can use the block move commands to your heart's content, and any other features you may care for. If you choose, you can even build a file of proper commands and run them in your spelling checker to get a first cut at possible syntax errors. I don't know about you, but I can never remember the BASIC line

editing commands for longer than ten minutes. As a result, I often re-type a line that I just edited and then inadvertently erased. And, oh yes, FORTRAN doesn't require sequential line numbers! It uses line numbers for branching references, but only the branches need to be numbered, and those don't need to be sequential. Line 10 can occur after Line 4356, and nothing untoward will happen. Try inserting 50 new lines in your BASIC program sometime when you discover a new way to make it work faster or better!

### Drawbacks:

By now your suspicions should be aroused. Isn't there anything about this language that isn't so great? Well, as a matter of fact, there are some things. They aren't devastating, but they do require some thought as you create programs. (By now, you should have determined that I think FORTRAN is a pretty good language, so you can be sure we will also talk in this column about ways to overcome the drawbacks).

FORTRAN came of age when computers were fed data on punched cards. You have all seen these in one disguise or another -- they are the stiff rectangular cards with 80 columns from left to right, and 12 rows which can have little rectangular holes punched in them. Each pattern of holes in a column distinctly identifies a letter, number, or special character. They seem antique today, but punched cards were a major improvement over punched paper tape, both in terms of speed and the fact that tape is purely sequential -- you cannot easily insert something in the data already punched in the tape. You can, however, easily insert several new data cards between the old ones.

Enough history. Because of this method of input, FORTRAN developers designed a format that was easy to put in a punched card, and which provided for some basic checks that the computer could make on the input cycle. In today's world of interactive computers, these format requirements can make for awkward I/O without a little forethought. It is largely because of the apparent difficulty of FORTRAN I/O that most of the microcomputer users I have talked to have avoided the language.

I/O with FORTRAN does require some more planning than it does with BASIC, but not much. Once you get the hang of it, it isn't any more of a problem than laying out a page using BASIC. Most of the literature out there on FORTRAN still presents I/O as if you were using punched cards. We aren't! A major aim of this column is to present some straightforward subroutines and methods for achieving input and output relatively effortlessly in FORTRAN.

For the same reasons, program statements have a format which appears rather odd at first glance. As an example, each statement must begin in column 7 and not extend beyond column 72 (yes, they can be continued on another line). Not to worry; these syntactical peculiarities will soon seem much less troublesome than the need for a statement number at the beginning of every BASIC program line.

Because of its line formatting requirements, structured programming in FORTRAN requires a little more thought than it does in Pascal. On the other hand, structured programming in BASIC is virtually impossible, so this is not a serious drawback.

There are those who would have you believe that FORTRAN cannot handle strings. Not so. No language is great at everything, though, and even die-hard supporters have to acknowledge that FORTRAN would probably not be their first choice for ease of string handling. But it would be for speed! This is another that has been glossed over in the microcomputer literature, and which we will talk about in detail in this column.

To summarize, FORTRAN is an excellent language choice for pro-

ducing fast, efficient programs. Several powerful programming features, such as linking of external subprograms, are available. The cost of all this is some attention to formatting details for I/O and program statements. For any application that does significant amounts of calculation or data manipulation, FORTRAN is likely to be a good choice as the programming language.

### What's Available?

There are three FORTRAN compilers readily available for Heath/Zenith computers: Microsoft FORTRAN, Nevada FORTRAN, and Supersoft FORTRAN. Let's examine each one separately.

Microsoft FORTRAN is the "old reliable" for the Heath/Zenith line, as well as for most other micro's. IBM FORTRAN appears to be an anomaly; the 8-bit Microsoft FORTRAN compilers have demonstrated an excellent capability to generate efficient, fast-running machine code. Microsoft 8-bit FORTRAN is available for the Heath/Zenith line from Heath (P/N H-8-20, \$175 for HDOS; P/N HMS-817-2, \$195 for CP/M hard-sector format; P/N HMS-837-2, \$195 for CP/M soft-sector format), and from a variety of independent suppliers. A quick check of some recent ads indicates prices ranging from \$239 to \$329. However, the Heath version of FORTRAN, especially in hard-sector, can often be found at substantial discounts. I was able to obtain a copy for under \$100 from a supplier who was lowering his inventory. This is an example of how the relative unpopularity of FORTRAN on micro's can work to your advantage. If you use only HDOS, this is your only choice for a FORTRAN compiler.

Microsoft FORTRAN supports the ANSI-66 standards, except for the COMPLEX data type (more about this in a later column). It has some language extensions. It requires 48K RAM and two disk drives.

If you want the 16-bit version of FORTRAN for your H/Z-100, it is P/N MS-463-2, \$195 from Heath. It requires one disk drive (two are recommended), 192K RAM, and MS-DOS or Z-DOS. A word of caution: this is a clone of the IBM FORTRAN that has been described by many users as extremely slow in execution. This compiler supports most of the ANSI-77 FORTRAN standards, which include the ANSI-66 standards as a major subset. It does not support the COMPLEX data type.

There is a bit more to Microsoft FORTRAN than meets the eye. First, of course, you get the FORTRAN compiler, which is called F80.COM. You also get the library of functions and routines, FORLIB.COM, and a library manager utility. And you get Microsoft's MACRO-80 assembler, M80.COM, and the associated linker, L80.COM. The linker is necessary for the compilation process, but both linker and assembler are also useful for assembly language work that has nothing to do with FORTRAN, so you are, in effect, getting a "bonus."

The cheapest way to get into the FORTRAN business is with Nevada FORTRAN (available from Ellis Computing, 3917 Noriega Street, San Francisco, CA 94122, (415) 753-0186, for \$39.95 in Heath hard or soft sector format). Nevada FORTRAN is a pseudo-compiled language; however, the compiler generates an intermediate code which is then interpreted at run-time. This is the same difference that exists between CBASIC and MBASIC. This process is inherently slower than running equivalently efficient native code, but it does offer the opportunity to get into FORTRAN programming at a very low cost.

Nevada FORTRAN also supports the ANSI-66 FORTRAN standards except for COMPLEX types, and has some extensions. It includes an 8080 assembler in the package, together with the pseudo-compiler

and the run-time support package. Some of the language extensions include IF-THEN-ELSE constructs and a TRACE-style debugger. If you use such extensions in any FORTRAN product, however, your code will not be especially portable. That may not matter to you, though, so look closely at your requirements before placing excessive weight on that factor.

If you can't get a "buy" on Microsoft FORTRAN, or if you aren't sure you want to pursue FORTRAN in the long term, then Nevada FORTRAN may be just the thing. The system requirements are as for Microsoft FORTRAN.

The only other compiler of which I am aware is the SuperSoft FORTRAN compiler (available from SuperSoft, P. O. Box 1628, Champaign, IL 61820, (217) 359-2112, \$425 in either 8- or 16-bit versions). This compiler is advertised as meeting the ANSI-66 standards, and also as having several important extensions. Notable amongst the claims for this compiler is the availability of 8087 support (for a \$50 added fee) and free-form string input and output. SuperSoft FORTRAN is available for CP/M-80, CP/M-86, or MS-DOS. It requires 32K memory with CP/M-80 and 128K with the other operating systems.

The SuperSoft FORTRAN compiler is the only one to provide the COMPLEX data type. It also claims a wide variety of special functions, and claims that it is over ten times faster than IBM FORTRAN. That still represents a speed 3-4 times slower than 8-bit Microsoft FORTRAN, however.

Because of its wide availability and long history, we will use Microsoft FORTRAN for program development in this column. If there is sufficient reader interest, Nevada or Supersoft FORTRAN may be included at a later date. However, remember that FORTRAN source code is portable. Only when we write code to utilize a language extension found in one manufacturer's product, but not in another's, will we likely run into any trouble when we switch machines or compilers.

### What Will We Do With FORTRAN?

I prefer articles that help me create something useful while I'm learning a new language or facility. After all, how many more sort routines or erase utilities does my system really need?

FORTRAN was built for calculation, so we are going to develop a system that takes advantage of its abilities in that area -- a technical analysis program for use with stock market data. We will construct our system in modular fashion. Particular attention will be paid to the fine points of I/O, screen and printer control, and data handling. Our goal is a system that will be useful, accurate, easy to use, and fast.

Stock market analysis involves many calculations, and round-off errors can literally cost you money, so we will deal carefully with how to ensure that what we do is not only fast, but also accurate. We'll implement some statistical routines as well, and in the process we will cover loops, array handling, file structure and management, and special features and "tricks of the trade."

Few things are more frustrating than articles which give you program listings that "ought to work," or which omit half the detail you need to make it work properly. We won't do that here. Everything you see will be an exact copy of what was done on my development system. If it didn't work, you won't read about it (except perhaps to illustrate how not to do something). We will follow a step-by-step approach to programming, compiling, and linking, so that these critical steps become second nature to you. In short, the purpose of this column is to make you comfortable with a powerful tool to solve problems that involve large amounts of calculation: FORTRAN.

The system used to develop the programs in this column is an H-89A with one 48 tpi single-sided drive and two 96 tpi double-sided drives running Heath/Zenith CP/M 2.2.03. The system has 64K RAM and is running Microsoft FORTRAN Version 3.4. If we get to the point that more than 48K RAM and two drives is required, it will be plainly identified.

To provide a common ground, we will use Microsoft BASIC as a point of departure to teach FORTRAN commands and syntax. There are two reasons for this: BASIC has become the "first language" for most microcomputer owners; and BASIC was developed from the FORTRAN language, so their similarities can aid understanding.

Next column it will be time to roll up your sleeves. We are going to begin our programming tutorial with an introduction to the format of FORTRAN commands, an orientation to several commands we will need for inputting information to the computer, and we'll actually write and compile a program to get data into the machine interactively. If you already have a FORTRAN compiler, brush up with your reference manual; if not, why not get one and join us?

## About the Author:

*Dick Stanley is a lieutenant colonel in the Army Signal Corps, and is currently the deputy director of the Ada Joint Program Office. He wrote his first FORTRAN program for an IBM 1620 in 1962, and has been "hooked" ever since. He has been working in communications and computer engineering since receiving his MS degree from Lehigh University over 20 years ago. In his spare time, he serves as vice-president of Wheeler Associates, Ltd., a consulting and publishing firm in Alexandria, Virginia, which specializes in education and computers.*



## ASSEMBLER CAN BE FUN

### Announcing the OMNICODER™ Assembler

The Omnicoder™ Assembler and Linkage Editor are designed to be used by real people with real problems. The user interface is designed to keep you in control at all times. It even has an on-screen help facility. It has many sophisticated features that you would expect to find only on larger computers: program segmentation, automatic search of up to 16 drives, a true macro language with both positional and keyword parameters, global variables, unlimited nesting, and time stamping options. The linkage editor produces a linkage map to help in debugging your programs. There are many more features, too numerous to list here.

#### System Requirements

Z80®. 64K. CP/M®. One or two disk drives. Disk formats available: H17 (hard sector), H37 (soft sector), and H47/H67 (8-inch).

#### Future Plans

We plan to add multiple cpu and mnemonic capabilities. We plan to release a source code management system in early 1985, and our PANDORA™ operating system in late 1985.

#### Ordering Information

PRICE: \$90 per site (USA only, no foreign orders). Ohio residents add 5.5% sales tax. MC/Visa welcome. Return undamaged within ten days for refund.

Send your order to:

### MOCKINGBIRD DATA SYSTEMS

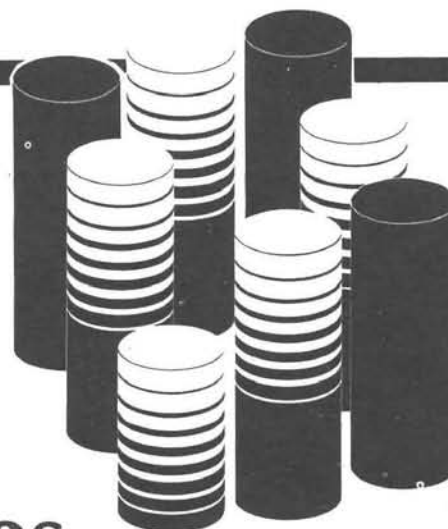
2296 Hoover Rd.  
Grove City, Ohio 43123

Trademarks: Zilog; Z80®; Digital Research; CP/M®; Heath Company; HDOS™; Mockingbird Data Systems; OMNICODER™; PANDORA™.



# Introduction to Data Structures

## Part 3 - Hash Tables



Emily A. Yount  
Rt 1, Box 408  
Danville, IN 46122

Computers are often used to store and retrieve information. In my first two articles on data structures, I described two techniques, linked lists and binary search trees, for storing information (see REMark 5(2) and 5(4)). With these methods, individual pieces of information can be retrieved as needed; also, it is easy to retrieve the stored data in sorted order. However, when searching for individual items in these structures, several memory locations may be probed before the correct item is found. Hash tables, also known as scatter tables or scatter stores, are another form of data structure used for storing and retrieving information. The term "hash", which can be used as an adjective or a verb, is of uncertain origin. It may have come into use because the data stored in a hash table seemed to be as mixed up as the meat and potatoes in hash. According to Knuth(5), the term was common jargon for several years before its first use in a scholarly journal in 1968.

The main advantage of hash tables is that usually only one or two memory locations are probed before finding the correct item (when retrieving data) or a place to insert the item (when storing data). One disadvantage is that the contents of an entire hash table cannot be retrieved in sorted order without first sorting the entire table.

A hash table is simply a sequential area of memory that has been divided into  $M$  "buckets". The buckets are numbered from 0 to  $M$ , and the number of the bucket is called its "address". Each bucket holds one or more "records". A record is a set of data to be stored together, e.g. name, address, and telephone number. Each record fills one "slot" in a bucket, and usually there is only one slot per bucket, and the terms "slot" and "bucket" will be used interchangeably. Each record is associated with a "key", and no two records in the table have the same key. The key is used to find the location of the record in the table. A "hash function",  $H$ , is used to transform the key into an address in the table, a process known as "hashing" or "key-to-address transformation".  $H(K)$  is the original hash address of the record with the key  $K$ . There are many possible hashing functions, but if  $K$  is an integer (or if the key can be converted to an integer), a reasonably good, but simple, hash function is  $H(K) = K \text{ mod } M$ , i.e. the remainder when  $K$  is divided by  $M$ . If this method is used,  $M$  should be a prime number.

Usually there are many more possible values for  $K$  than there are buckets in the table, but  $N$ , the number of keys actually entered in the table, is less than  $M$ . The term "load factor" is usually defined as  $N/M$ , and is a measure of the fullness of the table; the load factor is 1 when the table is full.

Since there are more possible values for  $K$  than there are for  $H(K)$ , it is likely that two or more different keys will have the same value of  $H(K)$ , and so map to the same address. When a new key hashes to a bucket that already contains a record, a "collision" has occurred; the new key is called a "collider". There are many techniques available for handling collisions, but some are better than others. The advantages of the better methods are most apparent in comparison to the poorer techniques. Since it is instructive to study the bad as well as the good, several techniques will be described in this article.

The "open addressing" techniques are methods for resolving collisions by inserting colliders at some point in a reproducible sequence of addresses. Every key  $K$  can be used to generate a "probe sequence", a sequence of addresses that are checked whenever the key  $K$  is inserted or retrieved. The search continues until  $K$  is found or an

SLOT	K	KEY	ORG ADDR
0	345	H89	0
1	530	ZENITH	1
2	463	COBOL	3
3	440	LISP	3
4	0		0
5	0		0
6	0		0
7	514	MARKER	8
8	514	REMARK	8
9	0		0
10	0		0
11	0		0
12	0		0
13	450	BASIC	13
14	291	C	15
15	572	FORTRAN	20
16	430	HERO	16
17	500	PASCAL	17
18	363	Z100	18
19	388	HUG	20
20	320	H8	21
21	458	HEATH	21
22	483	FORTH	0

Figure 1. Example of hashing with linear probing. In all figures, the keys were entered in the following sequence: REMARK, MARKER, HEATH, ZENITH, H8, H89, Z100, HERO, HUG, PASCAL, FORTRAN, BASIC, C, FORTH, LISP, COBOL.

empty slot is found. Finding an empty slot indicates that K is not in the table. The table is considered full when all slots, but one, are occupied.

In the simplest form of open addressing, a method called "linear probing", the collider is put in the next empty slot in the table. If the end of the table is reached, the search for an empty slot moves to the beginning of the table (i.e., if the initial hash address is  $H(K)$ , the probe sequence is  $(H(K) - 1) \bmod M, (H(K) - 2) \bmod M, \dots, (H(K) + M - 1) \bmod M$ ). The results are essentially the same if the sequence is  $(H(K) + 1) \bmod M, (H(K) + 2) \bmod M \dots$ . Probes are made sequentially through the table until an empty slot is found (an unsuccessful search), or the key is found in the table (a successful search). An example of this technique is shown in Figure 1 and Listing 1. The Listings show examples of BASIC and PASCAL subprograms used to produce such tables. In all examples, the table contains 23 slots. The keys used in these examples are terms that should be familiar to people who read REMark. The same sequence of keys was used for each figure, the differences in table arrangement resulted from the use of different algorithms. The keys are assumed to be strings of characters; only the first 8 characters are significant. Initially, all integer fields in the table contain zero, and any character fields are set to contain only blanks.

Deletions are a little more complex than insertions. Since an entry to be deleted may be one in a series of entries with the same original hash address, it is important to check for the existence of further entries in such a series. If such entries exist, they must be moved so that future find operations will be able to locate these items. For instance, if the record for "HEATH" is deleted from the table in Figure 1, the records for "C", "FORTRAN", "HUG", and "H8" must be moved. The result of such a deletion is shown in Figure 2.

SLOT	K	KEY	ORG ADDR
0	345	H89	0
1	530	ZENITH	1
2	463	COBOL	3
3	440	LISP	3
4	0		0
5	0		0
6	0		0
7	514	MARKER	8
8	514	REMARK	8
9	0		0
10	0		0
11	0		0
12	0		0
13	450	BASIC	13
14	0		15
15	291	C	20
16	430	HERO	16
17	500	PASCAL	17
18	363	Z100	18
19	572	FORTRAN	20
20	388	HUG	21
21	320	H8	21
22	483	FORTH	0

Figure 2. Contents of table shown in Figure 1 after deletion of "HEATH".

Linear probing is simple to program, but unfortunately it is one of the worst collision resolution techniques in terms of average number of probes needed per search. Entries have a tendency to form "clusters"; i.e. several entries will be grouped together in the table.

In order to find an empty slot for an entry that hashes into such a cluster, it may be necessary to probe several slots. The next available empty slot is the same for all entries that hash into such a cluster. As the table fills, it becomes probable that if a slot is full, the next slot in the sequence would also be full. This phenomenon is known as "primary clustering". In primary clustering, if two keys collide at any point in their probe sequence, they will also share the remaining probe sequence. Two different keys may share part of a probe sequence, even if they originally hash to different slots. The result is long sequences of occupied slots in the table. The quality of a hashing algorithm is usually described in terms of the average number of probes per search. The average number of probes per search is usually quite low; but in the worst case, e.g. when all keys have hashed to the same address, the number of probes per search may be very large. The mean number of probes per unsuccessful search remains low (less than 2.0) until the table is over 40% full (load factor 0.4), but beyond that point, the mean search time for unsuccessful searches increases rapidly as a function of load factor (5,10). For successful searches, the mean number of probes per search is less than 2.0 until the table is over 60% full. Again, beyond that point search times increase rapidly.

The "quadratic search" technique was proposed as a way of eliminating primary clustering(8). In this method the probe sequence is the sequence generated by computing  $H(K) + A*I + B*I*I$  for  $I = 0, 1, 2, 3, \dots$  and A and B are integer constants. One problem with this method is that when table size is prime, only half the table is searched before the search procedure returns to the original hash address. This problem can be eliminated by the use of slightly more complex probe sequences (3,9). However another problem still takes place, "secondary clustering" is said to occur when two different keys (K and K'), with the same original hash address ( $H(K) = H(K')$ ), have the same probe sequence.

The quadratic search techniques were quickly superseded by the idea of "double hashing". The basic idea of double hashing is that a one hash function ( $H1(K)$ ) is used to determine the original hash address, and a second hash function ( $H2(K)$ ) is used to determine the probe sequence. Even if two different keys share the same original hash address, with this method it is very unlikely that they will share the same probe sequence. The second hash function,  $H2(K)$ , must produce a value that is "relatively prime" to M, (i.e.  $H2(K)$  and M have no common divisors greater than one). Also, the values of  $H2(K)$  must fall between 1 and M-1. For instance, if M is prime and  $H1(K) = K \bmod M$ , then  $H2(K)$  could be  $(K \bmod (M-2)) + 1$ . In this case, it is good if M and M-2 are both primes, but M-2 need not be prime. Another possibility is  $H2(K) = ((K \text{ div } M) \bmod (M - 2)) + 1$ . Examples of subroutines used for double hashing are shown in Figure 2 and Listing 2. In this example M (23) is prime, and M-2 (21) is not prime, but the values of  $H2(K)$  will range from 1 to 21, numbers that are relatively prime to 23.

When double hashing is used, deletions are usually handled by using a special field in the record to mark the record as being "deleted" rather than being "empty" or "full". Then when searching for a key, the search sequence must continue until an "empty" slot is found, but if it is known that a key is not in the table, the entry for that key could be made in the first slot marked as "deleted". If the table is too full of "deleted" entries, it may be necessary to "rehash" the table, a procedure described in reference 8.

There are several variations on double hashing that give improved search times in one way or another. For instance, Brent's algorithm (2, 5) decreases the time needed for successful searches at the expense of longer insertion times. Also, the "ordered hashing" method of Amble and Knuth (1, 6, 9) decreases the amount of time

needed to do unsuccessful searches, again at the cost of longer insertion times. An interesting feature of ordered hashing is that there is only one possible final table arrangement for a particular set of keys. The arrangement does not depend on the order in which the keys are entered, but is the same as if the keys had been ordered prior to insertion.

Double hashing gives better search times than linear probing. The mean number of probes per unsuccessful search does not go above 2.0 until the table is 50% full, and the mean number of probes per successful search is below 2.0 until the table is 80% full. However,

SLOT	K	KEY	ORG ADDR
0	345	H89	0
1	530	ZENITH	1
2	0		3
3	440	LISP	3
4	0		0
5	0		0
6	0		0
7	0		8
8	514	REMARK	8
9	388	HUG	0
10	0		0
11	0		0
12	463	COBOL	0
13	450	BASIC	13
14	572	FORTTRAN	15
15	320	H8	20
16	430	HERO	16
17	500	PASCAL	17
18	363	Z100	18
19	291	C	20
20	514	MARKER	21
21	458	HEATH	21
22	483	FORTH	0

**Figure 3.** Example of double hashing. In all figures, the keys were entered in the following sequence: REMARK, MARKER, HEATH, ZENITH, H8, H89, Z100, HERO, HUG, PASCAL, FORTTRAN, BASIC, C, FORTH, LISP, COBOL.

mean search times still increase rapidly as the load factors increase further. An example of double hashing is shown in Listing 2 and Figure 3. Even though the mean number of probes per search is lower for double hashing than for linear probing, some sequences of keys may result in table arrangements in which the number of probes per search is greater with double hashing. As a way of testing your understanding of hashing, try the average number of probes per successful search for the keys in the tables shown in Figures 1 and 3. Then try entering the same keys, but in a different sequence, and recalculate the average number of probes per search.

Another way to resolve collisions is to include a "link" field in each record and "chain" colliding records into a linked list. There would be M linked lists, each with a list head, one for each bucket in the table. The individual lists may be kept in order so that an unsuccessful search need not go all the way to the end of the list. In this method, the colliders are stored in an area of memory outside the hash table. It may be another array, from which available nodes are taken as needed; or in PASCAL, the additional memory could be allocated dynamically with the "new" function. One disadvantage of this method is that if two blocks of storage are allocated, one for the table and one for storing colliders, it is possible that one block may become full while the other still has space left, so that "overflow" occurs before all the available space has been used. One way to, at

least, postpone the number of entries that can be made before a table overflow occurs, is to have more than one slot per bucket. If there are b slots per bucket, collisions are no problem until an attempt is made to enter a record into a bucket that already contains b records. In this case, the colliding record must be stored in an overflow area, and linked to the original bucket by a link at the end of the full bucket that points to the location of the collider.

The coalesced chaining method is one of the best collision resolution schemes available. Its only disadvantage over open addressing methods is that an additional link field is required for each record. Its advantages are that it is easy to program, the average number of probes per search is less than 2.0, even when the table is full, and overflow does not occur until all the available space has been used.

In coalesced chaining, the table is divided into two sections, the "address region" and the "cellar". The cellar is reserved for storage of colliders. The slots in the address region are numbered from one to M and the slots in the cellar are numbered from M + 1 to MT, where MT is the number of slots in the entire table. To simplify discussion, the first slot will be considered the "top" of the table and the MTth slot will be considered the "bottom". The hash function H transforms a key into a number from 1 to M. As in the hashing schemes described above, this address is probed first. If this slot is empty, the record is inserted; if the record contains the current key, the search is successful. Otherwise, a collision has occurred. As in separate chaining, the colliders that hash to the same original address are linked together with the record at the original hash address at the head of the list. Unlike separate chaining, no auxiliary storage is needed. Instead, colliders are stored in the table; a collider is stored in the vacant slot with the highest number (the bottom-most empty slot) and linked to the record at the original hash address. The first MT - M colliders will all be stored in the cellar, after the cellar becomes full, the address region can be used for storage of colliders. When the cellar is full, the next collider will be stored in the bottom-most

SLOT	K	KEY	ORG ADDR	LINK
1	0		0	0
2	514	HEATH	2	23
3	458	HEATH	3	22
4	345	H89	4	16
5	0		0	0
6	0		0	0
7	500	PASCAL	7	20
8	463	COBOL	8	0
9	388	HUG	9	19
10	0		0	0
11	0		0	0
12	0		0	0
13	430	HERO	13	0
14	450	BASIC	14	0
15	0		0	0
16	440	LISP	4	0
17	320	H8	17	0
18	530	ZENITH	18	0
19	483	FORTH	9	0
20	291	C	7	0
21	572	FORTTRAN	3	0
22	363	Z100	3	21
23	514	MARKER	2	0

**Figure 4.** Example of hashing with coalesced chaining. In all figures, the keys were entered in the following sequence: REMARK, MARKER, HEATH, ZENITH, H8, H89, Z100, HERO, HUG, PASCAL, FORTTRAN, BASIC, C, FORTH, LISP, COBOL.

empty slot in the address region. This process can continue until the table is completely full. See Figure 4 and Listing 3 for examples.

There are a number of variations on the basic coalesced chaining algorithm (10). "Standard" coalesced chaining is a special case in which the cellar size is zero and all colliders are stored in the address region. Separate chaining is actually a special case of coalesced chaining, in which the storage of colliders in the address region is forbidden. Thus, although search times may appear to be better for separate chaining (5), the differences are actually due to the way in which the term "load factor" is defined (10). The definition of "load factor" for separate chaining is usually defined as  $N/M$  and does not include the auxiliary space used for storing colliders. The definition of load factor for coalesced chaining is usually  $N/MT$ , and so does include the space used for storing colliders.

Although there is no single "best" cellar size, setting the cellar size to 14% of  $MT$  seems to be a good compromise(10). Fewer collisions occur when the address region is larger, but the cellar overflows sooner. "Early insertion coalesced chaining" is another variation. In this method, instead of inserting a collider at the end of a chain, the collider is inserted between the record at the original hash address and the next record in the chain. This technique has some disadvantages that are overcome by the "varied-insertion" method. In varied-insertion coalesced chaining, records are inserted as in early insertion, unless the cellar is full and the record at the original hash address  $s$  linked to a chain in the cellar. In this case, the record is inserted in the chain after the last cellar record in the chain. Implementations of coalesced chaining that allow deletions are described in reference 10.

In summary, if one can afford to set aside the additional space for link fields, coalesced chaining is probably one of the best collision

resolution algorithms available. If conserving space is more important than conserving time, one of the better open addressing techniques would be a good choice.

### Listing 1

```

10  REM      A PROGRAM TO DEMONSTRATE HASHING WITH LINEAR PROBING
20  REM
30  REM      TABLE INITIALIZATION : TABLE SIZE IS M
40  DEFINT I-M
50  CLEAR
60  LET M = 23
70  DIM T%(M-1)
80  DIM HK%(M-1)
90  DIM REC$(M-1)
100 DEF FNH(IX) = IX MOD M

350 REM
360 REM      KEY CONVERSION ROUTINE
370 K = ASC(KEY$)
380 IF LEN(KEY$) < 8 THEN KEY$ = KEY$ + SPACES$(8 - LEN(KEY$))
390 FOR I = 2 TO 8
400 LET K = K + ASC(MID$(KEY$,I,1))
410 NEXT I
420 RETURN

530 REM      ENTER AND FIND ROUTINE
540 GOSUB 360
550 LET I = FNH(K)
560 HG = I
570 WHILE ((REC$(I) <> KEY$) AND (T%(I) <> 0))
580 LET I = I - 1
590 IF I < 0 THEN LET I = I + M
600 WEND
610 IF REC$(I) = KEY$ THEN PRINT KEY$; " IS IN SLOT "; I; ". "
620 ELSE IF N = M - 1 THEN PRINT "TABLE IS FULL. "
630 ELSE GOTO 630
640 RETURN
650 REM      ENTER INTO TABLE
660 IF (C$ = "FIND") OR (C$ = "DELETE") THEN PRINT KEY$; " IS NOT IN TABLE. "
670 ELSE LET N = N + 1:
680 LET T%(I) = K:
690 LET REC$(I) = KEY$
700 LET HK%(I) = HG
710 RETURN

700 REM      DELETE ROUTINE
710 REM      FIRST MUST FIND
720 GOSUB 530
730 LET T%(I) = 0
740 LET REC$(I) = ""
750 LET HK%(I) = 0
760 LET J = I
770 LET I = I - 1
780 IF I < 0 THEN I = I + M
790 IF T%(I) = 0 THEN RETURN
800 LET R% = FNH(T%(I))
810 IF ((I <= R%) AND (R% < J)) OR ((R% < J) AND (J < I)) OR ((J < I) AND (I <= R%))
820 GOTO 770
830 REM      MOVE A RECORD
840 LET REC$(J) = REC$(I)
850 LET T%(J) = T%(I)

```

```

IF T[I].KEY = S THEN
  BEGIN
    SLOT := I;
    IN_TABLE := TRUE
  END
ELSE
  IF N = M - 1 THEN
    BEGIN
      SLOT := M; { TABLE IS FULL—OVERFLOW }
      IN_TABLE := FALSE;
      WRITELN ( 'TABLE OVERFLOW' )
    END
  ELSE { NOT IN TABLE BUT ROOM AT SLOT I }
    BEGIN
      SLOT := I;
      IN_TABLE := FALSE;
    END;
  { FIND }
END;

PROCEDURE ENTER(S : STRING; SLOT : INTEGER);
BEGIN
  N := N + 1;
  T[SLOT].K := C;
  T[SLOT].HK := H(C);
  T[SLOT].KEY := S;
END;

PROCEDURE DELETE(S : STRING);

```

```

850 LET HK%(J) = HK%(I)
860 GOTO 730
870 END
PROGRAM HASHL(F);
{ DEMONSTRATES HASHING USING LINEAR PROBING TO RESOLVE
  COLLISIONS }
CONST
  BLANK = 32;
  TAB = 9;
  CR = 13;
  M = 23; {TABLE SIZE }
  MAX = 20;
TYPE
  STRING = ARRAY[1..MAX] OF CHAR;
  ENTRY = RECORD
    K : INTEGER;
    HK : INTEGER;
    KEY : STRING
  END;
VAR
  T : ARRAY[0..M] OF ENTRY; { HASH TABLE }
  N : INTEGER; { NUMBER OF ENTRIES IN TABLE T }
  C : INTEGER; { CURRENT CONVERTED STRING }
  COMMAND, KEYSTRING : STRING;
  LOC : INTEGER; { LOCATION IN TABLE OF CURRENT KEY }
  IN_T : BOOLEAN; { TRUE IF KEY IN TABLE }
  I : INTEGER;
  F : FILE OF CHAR;
FUNCTION H (X : INTEGER) : INTEGER;
BEGIN
  H := X MOD M
END;
FUNCTION CONVERT(KS : STRING) : INTEGER;
VAR
  I, KI : INTEGER;
  KI := 0;
FOR I := 1 TO 8 DO
  KI := KI + ORD(KS[I]);
CONVERT := KI;
END; (* CONVERT *)
PROCEDURE FIND(S : STRING; VAR SLOT : INTEGER; VAR IN_TABLE : BOOLEAN);
VAR
  I : INTEGER;
BEGIN
  C := CONVERT(S);
  I := H(C);
  WHILE (T[I].KEY <> S) AND (T[I].K <> 0) DO
    BEGIN
      I := I + 1;
      IF I < 0 THEN I := I + M;
    END;
  END;
END;
VAR
  WHERE, I, J, R : INTEGER;
  IN_TABLE, FINISHED : BOOLEAN;
BEGIN
  FINISHED := FALSE;
  FIND(S, WHERE, IN_TABLE);
  IF NOT IN_TABLE THEN WRITELN(S, 'IS NOT IN TABLE')
  ELSE
    BEGIN
      I := WHERE;
      BEGIN
        T[I].K := 0;
        T[I].HK := 0;
        T[I].KEY := ' ';
        J := I;
        REPEAT
          I := I - 1;
          IF I < 0 THEN I := I + M;
          IF T[I].K = 0 THEN FINISHED := TRUE ELSE
            BEGIN
              R := H(T[I].K); { ORIGINAL HASH ADDRESS
                            OF KEY IN SLOT I }
              IF NOT ((I <= R) AND (R < J)) OR ((R < J) AND (J < I)) OR ((J < I) AND (I <= R))
                THEN BEGIN
                  { MOVE A RECORD }
                  T[J].K := T[I].K;
                  T[I].K := 0;
                  T[J].HK := 0;
                  T[I].HK := 0;
                  T[J].KEY := ' ';
                  T[I].KEY := ' ';
                  J := I;
                END;
              UNTIL FINISHED;
            END;
          END; { DELETE }
        END;
      END;
      PROCEDURE INIT;
      BEGIN { INIT }
        { INITIALIZE TABLE }
        N := 0;
        FOR I := 0 TO M DO
          BEGIN
            T[I].K := 0;
            T[I].HK := 0;
            T[I].KEY := ' ';
          END;
        END;
      END;
    END;
  REM A PROGRAM TO DEMONSTRATE DOUBLE HASHING
  REM TABLE INITIALIZATION : TABLE SIZE IS M
  REM DEFINT I-M

```

Listing 2

```

50 CLEAR
60 LET M = 23
70 DIM T%(M-1)
80 DIM REC$(M-1)
90 DIM HK$(M-1)
100 DEF FNH1(IX) = IX MOD M
110 DEF FNH2(IX) = (IX MOD (M-2)) + 1

360 REM KEY CONVERSION ROUTINE
370 K = AS (KEY$)
380 IF LEN(KEY$) < 8 THEN KEY$ = KEY$ + SPACES$(8 - LEN(KEY$))
390 FOR I = 2 TO 8
400 LET K = K + ASC(MID$(KEY$,I,1))
410 NEXT I
420 RETURN

530 REM ENTER AND FIND ROUTINE
540 GOSUB 360
550 REM HASH TO ORIGINAL HASH ADDRESS
560 LET I = FNH1(K)
570 IF T%(I) = 0 THEN GOTO 660
580 IF REC$(I) = KEY$ THEN RETURN
590 REM IF COLLISION DO SECOND HASH
600 J = FNH2(K)
610 LET I = I - J
620 IF I < 0 THEN LET I = I + M
630 IF T%(I) = 0 THEN GOTO 660
640 IF REC$(I) = KEY$ THEN RETURN
650 GOTO 610
660 REM ENTER INTO TABLE
670 IF (C$ = "FIND") THEN PRINT KEY$; " IS NOT IN TABLE."; RETURN
680 IF N = M - 1 THEN PRINT "OVERFLOW": RETURN
690 LET N = N + 1
700 LET T%(I) = K
710 LET REC$(I) = KEY$
720 LET HK$(I) = FNH1(K)
730 RETURN

740 REM FIND SUBROUTINE
750 GOSUB 530
760 IF REC$(I) = KEY$ THEN PRINT KEY$; " IS AT SLOT " , I , " .
770 RETURN
780 END

PROGRAM HASHD(INPUT,OUTPUT,F);
{ DEMONSTRATES HASHING USING DOUBLE HASHING TO RESOLVE
COLLISIONS }

CONST
BLANK = 32;
TAB = 9;
CR = 13;
M = 19; {TABLE SIZE }
MAX = 20;

TYPE
STRING = ARRAY[1..MAX] OF CHAR;
ENTRY = RECORD

```

```

IN_TABLE := TRUE;
SLOT := I;
END;
END; { FIND }

PROCEDURE ENTER(S : STRING; SLOT : INTEGER);
BEGIN
N := N + 1;
T[SLOT].K := C;
T[SLOT].H1K := H1(C);
T[SLOT].H2K := H2(C);
T[SLOT].KEY := S;
WRITELN(F,S);
END;

PROCEDURE INIT;
BEGIN { INIT }
{ INITIALIZE TABLE }
N := 0;
FOR I := 0 TO M DO
BEGIN
T[I].K := 0;
T[I].H1K := 0;
T[I].H2K := 0;
T[I].KEY := ' ';
END;
END; {INIT}

```

**Listing 3**

```

20 REM
30 REM TABLE INITIALIZATION : TOTAL TABLE SIZE IS MT
40 REM ADDRESS REGION SIZE IS M
50 DEFINIT I-M
60 CLEAR
70 LET MT = 23
80 LET R% = MT + 1
90 REM R% IS A POINTER USED TO FIND THE "BOTTOMMOST" EMPTY SLOT
100 LET M = 19
110 REM ADDRESS REGION = SLOTS 1..19, CELLAR = SLOTS 20..23
120 DIM T%(MT)
130 DIM REC$(MT)
140 DIM HK$(MT)
150 DIM LINK(MT)
160 DEF FNH(IX) = (IX MOD M) + 1

410 REM
420 REM KEY CONVERSION ROUTINE
430 K = ASC(KEY$)
440 IF LEN(KEY$) < 8 THEN KEY$ = KEY$ + SPACES$(8 - LEN(KEY$))
450 FOR I = 2 TO 8
460 LET K = K + ASC(MID$(KEY$,I,1))
470 NEXT I
480 RETURN

570 REM ENTER AND FIND ROUTINE
580 GOSUB 420

```

```

K : INTEGER;
H1K : INTEGER;
H2K : INTEGER;
KEY : STRING
END;

VAR
T : ARRAY[0..M] OF ENTRY; { HASH TABLE }
N : INTEGER; { NUMBER OF ENTRIES IN TABLE T }
C : INTEGER; { CURRENT CONVERTED STRING }
COMMAND, KEYSTRING : STRING;
LOC : INTEGER; { LOCATION IN TABLE OF CURRENT KEY }
IN_T : BOOLEAN; { TRUE IF KEY IN TABLE }
I : INTEGER;
F : FILE OF CHAR;

FUNCTION H1 (X : INTEGER) : INTEGER;
BEGIN
H1 := X MOD M;
END;

FUNCTION H2 (X : INTEGER) : INTEGER;
BEGIN
H2 := (X MOD (M - 2)) + 1;
END;

FUNCTION CONVERT(KS : STRING) : INTEGER;
VAR
I, KI : INTEGER;
KI := 0;
FOR I := 1 TO 8 DO
KI := KI + ORD(KS[I]);
CONVERT := KI;
END; (* CONVERT *)

PROCEDURE FIND(S : STRING; VAR SLOT : INTEGER; VAR IN_TABLE : BOOLEAN);
VAR
I, J : INTEGER;
C := CONVERT(S);
I := H1(C);
IF (T[I].K <> 0) AND (T[I].KEY <> S) THEN
BEGIN
J := H2(C);
REPEAT
I := I - J;
IF I < 0 THEN I := I + M;
UNTIL (T[I].K = 0) OR (T[I].KEY = S);
END;
IF (T[I].K = 0) THEN
BEGIN
IN_TABLE := FALSE;
IF N = M - 1 THEN SLOT := M { SIGNAL OVERFLOW }
ELSE SLOT := I { RETURN SLOT FOR USE BY ENTER }
END
ELSE
BEGIN
{ SUCCESSFUL }

```

```

590 LET I = FNH(K)
600 H% = I
610 IF T%(I) = 0 THEN GOTO 760
620 REM SLOT I MUST BE OCCUPIED
630 REM CHECK TO SEE IF SEARCH SUCCESSFUL
640 IF KEY$ = REC$(I) THEN GOTO 840
650 IF LINK(I) <> 0 THEN LET I = LINK(I);
GOTO 640
660 REM SEARCH UNSUCCESSFUL—FIND BOTTOMMOST EMPTY SLOT TO ENTER
670 IF (C$ = "FIND") THEN GOTO 820
680 LET R% = R% - 1
690 WHILE T%(R%) <> 0
700 LET R% = R% - 1
710 WEND
720 IF R% = 0 THEN PRINT "OVERFLOW";
RETURN
730 REM IF NOT OVERFLOW, ENTER KEY IN TABLE
740 LET LINK(I) = R%
750 LET I = R%
760 LET T%(I) = K
770 LET REC$(I) = KEY$
780 HK%(I) = H%
790 LET LINK(I) = 0
810 RETURN
820 PRINT KEY$; " IS NOT IN TABLE."
830 RETURN
840 PRINT KEY$; " IS IN TABLE AT SLOT ", I, "."
850 RETURN
860 END

```

{ DEMONSTRATES HASHING USING COALESCED CHAINING TO RESOLVE COLLISIONS }

```

const
blank 8 32;
TAB = 9;
CR = 13;
MT = 23; { TABLE SIZE }
M = 19; { SIZE OF ADDRESS REGION; CELLAR SIZE IS 4 }
MAX = 20;

TYPE
STRING = ARRAY[1..MAX] OF CHAR;
ENTRY = RECORD
K : INTEGER;
HK : INTEGER;
KEY : STRING;
LINK : INTEGER
END;

VAR
T : ARRAY[0..MT] OF ENTRY; { HASH TABLE }
N : INTEGER; { NUMBER OF ENTRIES IN TABLE T }
C : INTEGER; { CURRENT CONVERTED STRING }
COMMAND, KEYSTRING : STRING;
LOC : INTEGER; { LOCATION IN TABLE OF CURRENT KEY }
IN_T : BOOLEAN; { TRUE IF KEY IN TABLE }
I, R : INTEGER;
F : FILE OF CHAR;

```

## References

1. Amble, O., and Knuth, D. E., "Ordered Hash Tables", The Computer Journal 17:135-142, 1974.
2. Brent, Richard P. "Reducing the Retrieval Time of Scatter Storage Techniques.", Communications of the Association for Computing Machinery 16(2):105-109, 1973.
3. Day, A. C., "Full Table Quadratic Searching for Scatter Storage", Communications of the Association for Computing Machinery, 13(8):481-482, 1970.
4. Horowitz, E. and Sahni, S., Fundamentals of Data Structures, Computer Science Press, Rockville, Maryland, 1982.
5. Knuth, D. E., The Art of Computer Programming: Vol. 3, Sorting and Searching, Addison-Wesley, Reading, Massachusetts, 1973.
6. Knuth, D. E., "Algorithms", Scientific American, 236(4):63- 80, April, 1977.
7. Maurer, W. D., "An Improved Hash Code for Scatter Storage", Communications of the Association for Computing Machinery, 11(1):35-38, 1968.
8. Radke, C. E., "The Use of the Quadratic Residue Search", Communications of the Association for Computing Machinery, 13(2):103-105, 1970.
9. Standish, T. A., Data Structure Techniques, Addison-Wesley, Reading, Massachusetts, 1980.
10. Vitter, J. S., "Implementations for Coalesced Hashing", Communications of the Association for Computing Machinery, 25(12):911-926, 1982.

```

BEGIN
  ENTER(S,I);
  IN_TABLE := FALSE;
END;
END; { FIND }

BEGIN { INIT }
{ INITIALIZE TABLE }
N := 0;
R := MT + 1;
FOR I := 0 TO MT DO
  BEGIN
    T[I].K := 0;
    T[I].HK := 0;
    T[I].KEY := 1;
    T[I].LINK := 0;
  END;
END; {INIT}

```

```

FUNCTION H (X : INTEGER) : INTEGER;
BEGIN
  H := X MOD M + 1
END;

FUNCTION CONVERT(KS : STRING) : INTEGER;
VAR
  I, KI : INTEGER;
BEGIN
  KI := 0;
  FOR I := 1 TO 8 DO
    KI := KI + ORD(KS[I]);
  CONVERT := KI;
END; (* CONVERT *)

PROCEDURE FIND(S : STRING; VAR SLOT : INTEGER; VAR IN_TABLE : BOOLEAN);
VAR
  I : INTEGER;

PROCEDURE ENTER(S : STRING; SLOT : INTEGER);
BEGIN
  N := N + 1;
  T[SLOT].K := C;
  T[SLOT].HK := H(C);
  T[SLOT].KEY := S;
  WRITELN(F,S);
END;

BEGIN {FIND}
  C := CONVERT(S);
  I := H(C);
  IF T[I].K = 0 THEN { ORIGINAL HASH LOCATION FULL }
    BEGIN
      WHILE (T[I].KEY <> S) AND (T[I].LINK <> 0) DO
        I := T[I].LINK;
      IF T[I].KEY <> S THEN { NO SUCCESS }
        IF COMMAND = 'FIND
          THEN IN_TABLE := FALSE
        ELSE
          BEGIN { ENTER IN THE TABLE }
            IN_TABLE := FALSE;
            REPEAT
              IF R <> 0 THEN R := R - 1; { DECREMENT R }
            UNTIL T[R].K = 0; { UNTIL AN EMPTY SLOT FOUND }
            IF R = 0 THEN WRITELN('OVERFLOW')
            ELSE
              BEGIN
                T[I].LINK := R;
                I := R;
                ENTER(S,I); { ENTER STRING IN SLOT I }
                T[I].LINK := 0
              END
            END
          ELSE { SUCCESS }
            BEGIN
              IN_TABLE := TRUE;
              SLOT := I
            END
          END
        ELSE { ORIGINAL HASH LOCATION EMPTY }

```





# "My Favorite Subroutines"

Dear Hug:

I have a comment on Mr. Frank Cepulkowski's subroutine in the August 1984 issue, Page 9. ASCII characters that the capital letters are decimal values starting with 41 for "A" and ending with 90 for "Z". The lower case letters are decimal value of 97 for lower case "a" through 122 for lower case "z".

Microsoft Basic-80 has a function called ASC(X\$) which returns the decimal value for the string X\$. Now if one tests the ASC(X\$) for greater than 90. If the value is greater than 90, one only has to subtract 32 from the ASC value and convert back to ASCII character by using the CHR\$ function. Then you have a lower case letter converted into a capital letter. The coding is as follows:

```

10 LINE INPUT X$
20 FOR I=0 TO LEN(X$)
30 IF ASC(MID$(X$,I,1))>90 THEN
   MID$(X$,I,1)=CHR$(ASC(MID$(X$,I,1))-32)
40 NEXT I

```

Sincerely,  
John Pierrel

Attn: "My Favorite Subroutine"

Reference is made to the "force upper case" subroutine by Frank Cepulkowski in the "My Favorite Subroutine" section of the August 1984 issue of REMark.

Upper case forcing subroutines are very useful in keeping alpha input "constant" for comparison and ordering subroutines. If I read the referenced subroutine correctly, it forces all input to a second variable. Probably, the subroutine would work more efficiently if it were reconfigured as:

```

10 X$=INPUT$(1):IF X$>CHR$(96)AND X$<CHR$(123)THEN
   X$=CHR$(ASC(X$)-32)
20 PRINT X$,:GOTO 10

```

This configuration of the subroutine keeps boolean operations and number of variables used to a minimum. It also "forces" only lower case letters. Other input remains unaffected.

Sincerely,  
Carl Edwin Lovett, Jr.

PS: Highly flattered you printed my favorite subroutine (the julian algorithms) in same issue. It boosted the confidence of an untutored computer nerd (me).

Dear Sirs,

Enclosed please find a routine in MBASIC (and sample output) which will print out a conversion chart for Fahrenheit to Celcius, from 97 to 106.9 degrees Fahrenheit. Someone may find it useful for a medical

thermometer which reads out in the wrong units.

The program assumes that the printer is set to 80 columns. I tried to make the program reasonably short to key in.

Thank you for your attention.

Sincerely,

P.G. Manney

```

10 LPRINT TAB(28):"TEMPERATURE CONVERSION CHART" 'TITLE
20 LPRINT TAB(28):"-----":LPRINT:LPRINT
30 A$=" FAHR CENT ":B$=" ----"
100 F=97:GOSUB 500 'SECTION I
200 LPRINT:LPRINT:F=102:GOSUB 500:END 'SECTION II
500 FOR I=1 TO 5:LPRINT A$:NEXT I:LPRINT 'HEADINGS
510 FOR I=1 TO 5:LPRINT B$:NEXT I:LPRINT:LPRINT
520 FOR I=1 TO 10
530 FOR J=1 TO 5
540 C=((F-32)*5)/9 'CONVERSION
550 LPRINT USING "###.##":F:LPRINT " ";
   :LPRINT USING "###.##":C:LPRINT " ";
560 F=F+1
570 NEXT J
580 LPRINT:F=F-4.9
590 NEXT I
600 RETURN

```

```

100 ' *** SCROLL PROTECTION ROUTINE ***
110 '
120 ' PROGRAM PEEKS AT MEMORY LOCATIONS AND RETURNS
   DEC VALUE STORED THERE
130 ' IT WAS WRITTEN PRIMARILY TO 'FILL THE SCREEN'
   AND THEN DEMONSTRATE
140 ' TITLE SCROLL PROTECTION ROUTINE
150 '
160 ' TOM MUJICA HUGNJ 8/28/84
170 ' 309 HIGH ST. NORWOOD NJ 07648
180 '
190 CLS
200 PRINT"MEMORY LOCATION) VALUE STORED"
210 X=1
220 Y=119
230 FOR I=X TO X+Y
240 PRINT I;" "; PEEK(I);" ";
250 IF I MOD 6 =0 THEN PRINT
260 NEXT I
270 PRINT
280 X=X+120
290 LINE INPUT"CONTINUE <Y> OR <CTRL-C> ? ";Q$
300 ' *** TECHNIQUE FOR SCROLL PROTECTING
   CHART OR TABLE HEADINGS ****
310 ' IN THIS EXAMPLE ONE LINE OF 'TITLE'
   IS SCROLL PROTECTED
320 ' BY LOCATING CURSOR ON 2 nd LINE AND
   ERASING TO END OF SCREEN
330 LOCATE 2,1
340 PRINT CHR$(26);
350 GOTO 230
360 END

```





# 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.

The following files are included on the HUG P/N 885-1238-[37] ASCIRITY disk:

ASCIRITY	.BAS	BRAG	.DAT	RTTYSUBS	.MAC
ASCIRITY	.COM	CQ	.DAT	README	.DOC
ASCIRITY	.DOC				

**Author:** Allen Gilchrist, Jr.

**Program Content:** After ASCIRITY is executed, the top line of the display indicates the present mode, baud rate, and storage status of the program. The next two lines are in reverse video, and briefly state the functions of the control sequences. The next eight lines are the transmitted data display, and the output buffer display. The lower half of the screen is reserved for received data.

The control sequences recognized by the program are (ctl-A) through (ctl-F). The function of these control codes are as follows:

ctl-A	-Switch to transmit and send the CQ message.
ctl-B	-Set new baud rate or mode.
ctl-C	-Save memory buffer to disk and/or exit to CP/M.
ctl-D	-Storage mode toggle switch.
ctl-E	-Transmit/Receive toggle switch.
ctl-F	-Transmit an ASCII file from disk.

**Comments:** none

**TABLE C Rating:** (0), (2), (5), (9)

## P/N 885-8031-[37] CP/M MORSE CODE TRANSCIVER Ver 2.0 .. \$20.00

**Introduction:** Morse Code Transceiver Ver 2.0 is an 8080 assembly language program, which provides the operator with the ability to send or receive morse code over a wide range of code speeds, dot/dash ratios, interference, and noise conditions. In addition, the precision speed feature is intended to be used whenever extreme transmit code speed accuracy is required.

The precision morse code speed algorithms used in this program were originally developed as part of a set of custom H89 programs for the American Radio Relay League's Maxim Memorial Station 'W1AW' at A.R.R.L. Headquarters in Newington, CT.

**Requirements:** This program requires the CP/M operating system version 2.0 on the H19/H8/H17 or H/Z89 with 48k of memory. Only one drive is required, however, two are recommended.

All I/O is at RS232C levels via the DTE port. External equipment is required to interface the RS232C level I/O signals to the amateur station equipment. Design details were published in REMark Issue 33, October 1982, Page 17.

**Note:** The algorithms used for morse code decoding depend on

## P/N 885-1238-[37] CP/M

**ASCIRITY** ..... \$20.00

**Introduction:** The program ASCIRITY is written for the Microsoft Basic Compiler and Macro 80 Assembler on the Heath H-89 computer with 64k of RAM and the CP/M operating system. With the proper interface or terminal unit and appropriate transmitting and receiving equipment, this program may be used to send and receive amateur radio RTTY and ASCII at all popular baud rates or transmission speeds (wpm).

ASCIRITY features include transmission and reception of baudot RTTY at 45, 50, 54, or 74 baud; and ASCII RTTY at 110 or 300 baud. A CQ message of up to 256 characters is automatically loaded and can be sent at any time by pressing two keys. A brag file, or any ASCII file may be sent from disk. Lower case letters are converted to upper case for baudot transmission.

When text is entered from the keyboard for transmission, a line feed (LF) is inserted after each carriage return (CR). A line length of 72 characters is standard for most of the mechanical teleprinters in amateur use, and when text is entered from the keyboard, this program will insert a (CR) and (LF) after 72 characters, if none is typed. No (CR) or (LF) insertion is made when transmitting a file from disk. For baudot operation, the LTRS and FIGS characters are sent twice for each case change. This is to help insure good copy of your transmissions.

Transmitted and received data may be optionally stored to a 6000 character memory buffer, which can later be written to disk, if desired. There is a 1024 character transmit message buffer which may be filled from the keyboard while receiving or transmitting. If either of these buffers approaches being full, the terminal bell will sound with each new character.

**Requirements:** This software requires a Heath H/Z-89/90 with 64k of memory and a standard 3 port serial interface. An H/Z-19 and H-8 with 64k of memory and a standard 4 port serial interface will also work. Either system must be running the CP/M operating system and have at least one hard or soft sectored disk drive.

ASCIRITY is designed to communicate with most popular ham radio interfaces, (including the Heath HD-3030), or terminal units through the computer's RS-232 port at 330Q.

timing from the internal clock. Therefore, this program will perform properly only on a standard machine running at 2.048 MHz.

The following files are included on the HUG P/N 885-8031-[37] MORSE CODE TRANSCEIVER disk:

.MAC	RXINT	SAVMSG
RXLOP	CODETBL	RMP
U8250	TXSPEED	README
VARTBL	TXLOP	

**Author:** Robert R. Anderson K2BJG

**Program Content:** Morse Code Transceiver can receive and transmit standard morse characters, as well as special morse characters, such as (AR), (SK), (BK), (KN), (BT), and (AS). Both upper and lower case key input and screen display is allowable.

The CWDATA.DAT disk file contains eleven multi-character groups, which can at any time be read to the currently selected buffer and display screen. This file contains commonly used abbreviations and space for the station call sign and station location.

Two transmit buffers selected by the 'f2' key will transmit up to 254 characters. Ten message buffers can be loaded or cleared under control of the 'f1' key. Any selected message buffer can be transferred to the transmit buffer. These 10 buffers are saved in SAVMSG.DAT.

Disks can be changed and ASCII disk files can be loaded without leaving the program. The memory buffer extends from the end of the program to the top of available memory.

The receive program operates in three modes: LOCK, TRACK, and HOLD. The transmit program operates in two modes: NORMAL and DISK FILE. The modes can be manually switched to any of the modes.

The available precision fixed Tx speeds are: 5SP, 05, 7.5, 10, 13, 15, 18 WPM, and 20 through 70 WPM in 5 WPM steps. The morse code speed standard used in fixed speed mode is in accordance with amateur practice of one word being defined as consisting of 50 elements.

The screen display is split into three areas for viewing the receive or transmit data, the selected transmit pre-type data, and the selected message buffer data. The screen is also used as the command screen and for display of error messages. The program will not allow for improper keyboard commands to take place.

Printed documentation comes with the disk and does an excellent job of helping the beginner learn how to use Morse Code Transceiver.

**Comments:** The author has done an outstanding job with this CP/M version.

**TABLE C Rating:** (0), (1), (3), (5), (10)

---

**P/N 885-3016-37 Z-DOS**  
**ADVENTURE DISK ..... \$10.00**

---

**Introduction:** Adventure is one of the most well known and best liked computer games. It is an adventure through a giant cave to search out and find treasures. Many dangers, as well as puzzles to solve, are instore for the user who ventures into its midst.

**Requirements:** This game requires the Z-DOS operating system on an H/Z-100 computer. This game will also work properly with the

MS-DOS operating system on an H/Z-150/160. In either case, this program requires around 75k of memory. Only one 5.25" drive is required.

The following is a list of the files on the HUG P/N 885-3016-37 Z-DOS Adventure Game disk.

ADVENT	.EXE	ATAB	.DAT
AINDX	.DAT	README	.DOC

**Authors:** This program was originally developed by Willie Crowther. Most of the features of the current program were added by Don Woods (Don SU-AI). This microprocessor version was done by Brian Barnes and Dave Sandage of Zenith.

**Preparation:** To run ADVENTURE under Z-DOS or MS-DOS, simply copy the three adventure files, ADVENT, AINDX, and ATAB to a bootable disk and type ADVENT at the system prompt.

**Program Content:** Somewhere nearby is a colossal cave, where others have found fortunes in treasure and gold, though it is rumored that some who enter are never seen again. Magic is said to work in the cave. I will be your eyes and hands. Direct me with commands of one or two words. I should warn you that I look at only the first four letters of each word, so you'll have to enter 'northeast' as NE to distinguish it from 'north', 'dnstream' for 'downstream', etc. Should you get stuck, type 'help' and 'info' for some general hints.

**Help:** I know of places, actions, and things. Most of my vocabulary describes places and is used to move you there. To move, try words like forest, building, dnstream, enter, east, west, north, south, up, or down. I know about a few special objects, like a black rod hidden in the cave. These objects can be manipulated using some of the action words that I know. Usually you will need to give both the object and actionwords (in either order), but sometimes I can infer the object from the verb alone. Some objects also imply verbs; in particular, 'inventory' implies 'take inventory', which causes me to give you a list of what you're carrying. The objects have side effects; for instance, the rod scares the bird. Usually people trying unsuccessfully to manipulate an object are attempting something beyond their (or my!) capabilities and should try a completely different tack. To speed the game, you can sometimes move long distances with a single word. For example, 'building' usually gets you to the building from anywhere above ground except when lost in the forest. Also, note that cave passages turn alot, and that leaving a room to the north does not guarantee entering the next from the south.

**Suggestions:** Try 'ENTER BUILDING'. When you see an object, pick it up. Go 'DNST' (downstream) if you want to find the cave.

### Ordering Information

*For Visa and MasterCard phone orders; telephone Heath Company Parts Department at (616) 982-3571. Have the part number(s), descriptions, and quantity ready for quick processing. By mail; send order, plus 10% postage and handling (\$1.00 minimum charge, up to a maximum of \$5.00. UPS is \$1.75 minimum -- no maximum on UPS. UPS Blue Label is \$4.00 minimum.), to Heath Company Parts Department, Hilltop Road, St. Joseph, MI 49085. Visa and MasterCard require minimum \$10.00 order.*

*Any questions or problems regarding HUG software or REMark magazine should be directed to HUG at (616) 982-3463. REMEMBER - Heath Company Parts Department is NOT capable of answering questions regarding software or REMark.*

**Helpful Words:**

INVENTORY	List items you are carrying
SCORE	Show your current score
LOOK	Long description of current location
BACK	Go back the way you came
QUIT	Stop the game and give final score

**Command Style:** Remember, ADVENTURE takes one or two word commands only. Make them straight forward like:

ATTACK DRAGON	GET GOLD	DOWN or D
EAT BIRD	WEST or W	etc.
THROW AXE	UNLOCK GRATE	

**Info:** If you want to end your adventure early, say 'QUIT'. To see how well you're doing, say 'SCORE'. To get full credit for treasure, you must have left it safely in the building, though you get partial credit just for locating it. You lose points for getting killed, or for quitting, though the former costs you more. There are also points based on how much (if any) of the cave you've managed to explore; in particular, there is a large bonus just for getting in (to distinguish the beginners from the rest of the pack). I may occasionally offer hints if you seem to be having trouble.

\*\*\*\*\* GOOD LUCK \*\*\*\*\*

**Comments:** Adventure will provide many months and even years of fun trying to reach the 366 point goal.

**TABLE C Rating:** (10)

# HUG Price List

Part Number	Description of Product	Selling Price	Volume - 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	57

**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 HBASIC Utilities Disk .....	\$ 16.00	46

## CP/M

885-1210-[37]	CP/M ED (same as 885-1022) ....	\$ 20.00	
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
885-8011-[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	57
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	CP/M86 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

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

**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	

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



# Local HUG Club News

**Ft. Wayne Area HUG**

Steven Fensler is interested in starting a Ft. Wayne Area HUG. Please either write or call Steve at:

6617 Bandon Drive  
Ft. Wayne, IN 46815  
Phone # (219) 486-3738.

**Tulsa HUG**

Rt 1, Box 813  
Sperry, OK 74073  
Christian Kessler is contact person  
Tentative meeting day is the 2nd Tuesday at members homes  
5 members at present, no dues

**LSU HUG**

Baton Rouge, LA  
LSU H/Z Users' Group  
Dept. of Chemical Engineering  
Louisiana State University  
Baton Rouge, LA 70803  
Danny Reible is contact person and president  
Meet 2nd Wednesday at 4:00 pm at Center for Engineering & Business Administration  
\$5.00 dues/yr.

**LRH/ZUG**

Little Rock HUG is now the LRH/ZUG (Little Rock Heath/Zenith Users' Group). Also new address is 113 Dakota Jacksonville, AR 72076. New phone # (501) 988-5273.  
Now have 30 members and a 24 hour Bulletin Board (501) 988-5700.

**San Jose HUG**

Now only meets the 1st Wednesday each month at 7-8 pm at the Campbell Heathkit Electronics Center.

**Metro Detroit Area HUG**

Has a new address:  
35681 Hees  
Livonia, MI 48150.

New phone # (313) 427-3905.

New contact person is Neil E. Coffin, who is also Sec., Treas., and Librarian for the group. They now have 65-70 members and the president is Tom Livingstone.

**Cleveland HUG**

New contact person is Kent Currie.  
They now have 30 members and the Bulletin Board number is (216) 292-7554 during non-store hours.

**DAYHUG**

Wright Patterson HUG has changed its name to DAYHUG (Dayton HUG).  
New address:

1670 N. Laddie Ct.  
Beaver Creek, OH 45432.  
New phone # (513) 426-5014.  
New contact person is George Elwood.  
They have 160 members and meet 1st Thursday at 4:00 pm for H8/H9 users and the 3rd Thursday at 4:15 for Z-100 users.

**OKIHUG**

The new meeting date and place for OKIHUG (Okinawa HUG) is now the 2nd Friday monthly at the American Royal Office at 7:00 pm.

**TRY-STATE HUG**

2617 Country Way  
Fayetteville, AR 72701  
(501) 521-4818  
  
Contact person is Gil Hoellerich  
They meet 3rd Saturday at 1 pm at:  
  
Northwest Voc-Tech School  
Hwy 265 and Ford Road  
Springdale, AR  
  
Club just started and the group is growing!

**NHHUG**

Paul Eustace has notified us that the new contact person for the North Houston HUG is Barbara Hemmerling. The address is:  
  
20207 Cotton Glade  
Humble, TX 77338  
  
They also have a 24-hour Bulletin Board at (713) 583-1287.

**Green Bay HUG**

Green Bay area is looking for interested Heath Users' to start a HUG club. If interested send your name, address, and phone number to:  
  
David A. Ozarowicz  
505 Main Street  
Wrightstown, WI 54180





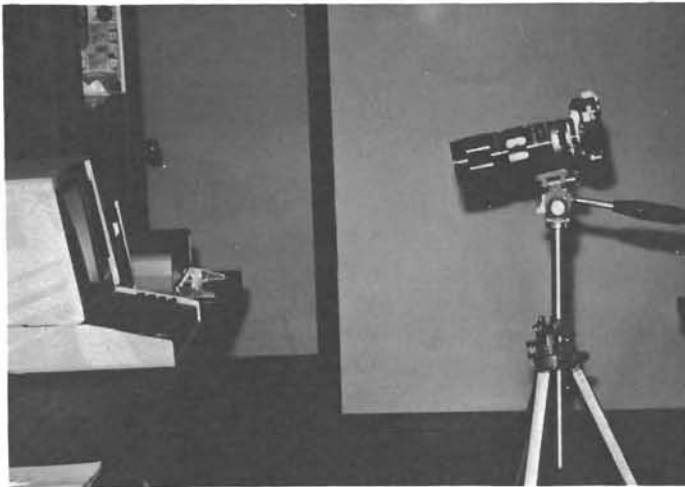


Figure 2

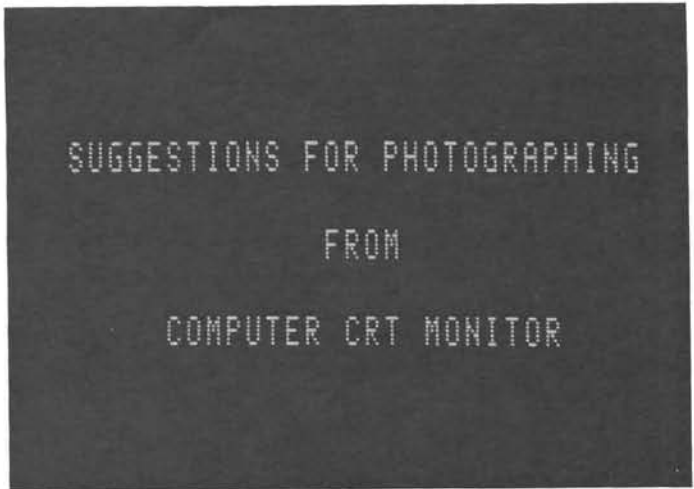


Figure 3



**About the Author:**

*Robert J. Telepak is a physician with the rank of LTC in the US Army and is stationed at Brooke Army Medical Center in San Antonio, TX. He has a BS and MS in physics, graduated in 1972 with an MD from the University of Colorado School of Medicine and Nuclear Medicine. Bob's major use of his H-89 is for record keeping, word processing (to produce lecture notes for teaching in the Department of Radiology) and preparing slides for lectures, as described above.*



**Z-GRAPH-100**  
 THE PREMIER  
 GRAPHICS SUPPORT  
 SYSTEM FOR YOUR Z-100

**Compatible With Most Popular Compilers**  
 MS-FORTRAN    MASM    Z-BASIC  
 MS-COBOL    C (Computer Innovations)    MS-Pascal  
 \*\*\*\*\*  
 Supports all Video and Memory Configurations  
 640 X 480 Interlace  
 640 X 240 Non-interlace  
 640 X 240 Non-interlace-Sync  
 Monochrome, 32K Color or 64k Color Memory

\*\*\*\*\*  
 Contains forty callable routines which provide a wide variety of high speed graphics capabilities which are based on the **TEKTRONIX** and **CORE** graphics standards. Among the capabilities are:

Plot, Draw, Circle, Arc, Move, Window, View, Newpage, Foreground and Background Color, Line Color, Absolute and Relative Vectors, Seventeen Standard Line Styles, User Definable Line Styles, Fast Polar to Rectangular Conversion, Three Area Fill Routines, Standard and User Defined Fonts in Eight Character Sizes and Four Writing Directions, ASCII and Greek Character Sets, Provides Special Drawing Modes to Support "Rubber Band" and Moving Marker or Cursor Applications.

Provides supplemental support for Random Number Generation, Pause, Direct Access to the Console and Auxiliary Ports and Runtime Error Condition Reporting.

Full user manual with table of contents and cross-reference index; all above-mentioned languages are included on 5 1/4" disk.

No restrictions or royalties on use of routines linked with your programs.  
 After sale support and phone consultation.  
 Dealer and quantity discounts available.

**Z-GRAPH-100**  
**\$89**

**NEW**  
**ORLEANS**  
**GENERAL**  
**DATA**  
**SERVICES**



Ordering Information: Products listed are available from NOGDS, 7230 Chadbourne Drive, New Orleans, LA 70126. Check, Money Order, VISA or MasterCard accepted. Phone orders call (504) 241-9495. Add \$5.00 shipping and handling.



---

# The Computers Are Coming . . . Are Here

Louise B. Guest  
S.R. 1, Box 85  
Lampe, MO 65681

"When are you going on vacation?" The secretaries who work in our school office asked me this question repeatedly. They were inquiring so that they would know when the "boss" wouldn't be around. What they really wanted to know was when they could finally get at the new computer housed in their office.

For almost three months they had seen me slowly learning how to get the computer to work for me. They saw me working, playing, being happy or discouraged -- then hitting the books (manuals) for the umpteenth time.

Before leaving for my vacation, I had formatted disks and set out a short "how to" for them to get started with the word processing program and the printer. They had seen that the computer wasn't easily damaged.

Two weeks later, I was greeted with a pile of printed computer paper and some very enthusiastic secretaries. They told me that they had "fun" while learning to use the word processor. During this last semester, we almost had to have a sign up sheet for appointments to use the computer.

Where did all this enthusiasm come from? How are computers changing the faculty, students and staff of this school? What more can we do with these computers? Some background information may help in part to answer these questions.

Several years ago it became obvious that the future of education would be dominated by the increasing use of computers. As an educational administrator, I felt that before I could ask the faculty and staff to accept computers into the school, I had to become thoroughly at ease with them myself.

With that in mind, I proceeded to take several courses; first a course in "Computers for Managers," then "Computers in Education," and finally a basic "BASIC" course. All these were given as short continuing education offerings at our local university.

The first course was by far the most informative for my purposes. The importance of getting the people who would be most involved in the operations of the computers in on the ground floor was stressed. Why we needed computers and what they could do for us were the basic questions.

As the head of this school, I have always asked for and gotten ideas, suggestions, and some detailed outlines on all facets of school administration, teaching, and functioning. So it was with the prospect of computers being introduced into this school.

Everyone was asked to submit a listing of what they would like a computer to do for them in their school job. The faculty addressed the areas of Computer Assisted Instruction (CAI), grade handling,

statistics, and report writing. The recruitment counselor wanted an easier way to keep track of the potential applicant and those newly admitted. The registrar wanted a way to know who was in the school, what classes they were taking, if they were receiving financial aid, how much they had paid, and if they needed a bill.

This left administration with wanting to schedule classroom usage in a simpler method, while having a quick way to determine who was in which class. There was also the very real need for budget control and planning. The area of inventory was also addressed. All in all -- the usual needs of an educational institution.

The next problem was to shop for the "right" computer for us. In order to make a knowledgeable decision on what type of computer and its capabilities were suited to our school, I had learned the jargon and understood the potentials of computers.

I now felt comfortable enough to visit the many computer sales representatives. I could discuss with each of them my four page outline of what we wanted the computer to do in and for our school. This outline was a compilation of all the previously mentioned input from faculty and staff.

Along the way, I did find out about several educational administration packaged programs -- especially one that would do everything on my four pages and more, but the cost was prohibitive for our size school.

What did we wind up with? The school now has two Zenith 110's (color) in our audiovisual learning center and one Zenith 120 (all-in-one) in our office. The Daisywriter printer and modem are also in the office.

The programs we have now and are using include:

For Z-DOS (MS-DOS):

Multiplan (MicroSoft) for student financial aid and the school budget

For CP/M-85:

Magic Wand (Peachtree Software, Inc.) for form letters and bills  
Personnal Pearl (Pearlsoft) a data management program -- just getting started with this

PIEZ (Software Toolworks) -- a text writer. This article was written using this program.

Quizzer (Interactive Micro Systems) -- to set up self testing modules.

Programs already written by the faculty include a grade reporting program, a spelling verifier, an annotated calendar, and a management decision aid. Most of these have been modifications, combinations, and adaptations of programs found in books and magazines.

How have the computers changed the behavior expectations of those in this school? More and more of the faculty are taking their own continuing education courses to learn about computers. Many have taken advantage of sales and now own a personal computer. Most importantly, they are not afraid to use the school computers, especially entering quiz grades and getting the sorted printout to post.

The students expect to see exam grades posted in less than two hours after finishing a test. In fact, they start to congregate near the appropriate bulletin board after one hour and are usually not disappointed. They are starting to use the text editor to write their term papers. The need for another printer has become evident.

The secretaries appreciate the ability to personalize letters without having to retype all of the letter.

As an educator, what have I learned and what have I taught? The oldest method of teaching -- by example -- has worked very well in the introduction of computers into the functioning of this school. No one would willingly go back to doing everything by hand and being bogged down with paper shuffling.

Each day, we all learn a little more, understand the capabilities better, and enlarge our list of additions needed to make the system even better.

We are on the way to becoming a computer managed school.



# ZWAR!

Tired of trying to get your latest Pascal compiler to work? Your spreadsheet won't compute? Sick of syntax errors in line 4658? It's time to take a break and indulge your fantasies of global domination with ZWAR.

ZWAR is a *two player* strategy game of world conquest. The screen displays a map of the world and the Soviet and American cities. You must decide what mix of the nine available weapons systems you will build to outwit your enemy. You can destroy his cities with ICBM's, or attempt to capture them with an invasion, thus gaining their production facilities for your own use. Each weapon has its strength: carriers and destroyers for sea superiority, ABM's for defense against missiles, jets for quick response, satellites for spying on enemy activity. The possible strategies are nearly endless.

So if you're getting tired of humdrum computing, grab a friend and try ZWAR. It's your chance to make history.

ZWAR, \$19.95 + \$1.50 shipping. For H8-H19, H/Z89 or H/Z100 systems (color graphics on the Z100), 56k, and HDOS, CP/M, or CP/M-85. Specify hard or soft sector disk. Free catalog available.

## APOGEE SOFTWARE

Box 15124  
Savannah, GA 31416

(912) 925-3765



## Controlled Data Recording Systems Inc.

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

Price \$495.00

### NEW PRODUCTS FOR THE FDC-880H

#### DM-1 DUAL BOARD MODIFICATION KIT \$29.95

Allows for both the FDC-880H and the H88-4 controller cards to interface with the same 5.25" drives. Drives will run as both hard sectored format and soft sectored format depending upon the logical drive letter.

#### CDR BIOS by Livingston Logic Labs \$60.00

Enhanced version of Heath/Zenith CP/M 2.203 BIOS with ZCPR. Supports all Heath/Zenith disk formats through the FDC-880H and the H17 controllers.

#### CDR DVD by Livingston Logic Labs. \$40.00

HDOS driver for running double density HDOS through the FDC-880H

#### Shugart Slimline 5.25" 40 track double sided drives \$275.00

#### Shugart Slimline 8" double sided drives \$525.00

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

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

## NEW Z-DOS WORDKEY™ SIMPLIFIES WORDSTAR™

Introducing WordKey. WordKey replaces **ALL** of WordStar's control-key commands with much simpler and faster keypad and function-key commands. Each function and keypad of your H/Z-100 is used two or three times over to define more than 90 key combinations. You'll never need to hold down the control key again! Look at these features:

- Keys are logically grouped so that the most often-used commands are single keystrokes. Move around the screen quickly and easily. Lesser used commands are two separate keystrokes that can be entered with one hand.
- Full onscreen help is always instantly available, including two different key diagrams and a separate explanation of each key's function. Just press the HELP key.
- Blank ruler line...Block cursor and key-click on/off...Detailed User's Manual and printed keyboard diagrams...And much more.
- Use with either WordStar version 3.21 or 3.30.

**Please send me:**

Z-DOS WordKey: \_\_\_\_\_ copies @ \$49.95 \$ \_\_\_\_\_  
 Special: C.Itoh Prowriter graphics screen  
 dump program \_\_\_\_\_ copies @ \$19.95 \$ \_\_\_\_\_  
 Calif. residents please add 6% tax \$ \_\_\_\_\_  
 Total Enclosed (includes mailing) \$ \_\_\_\_\_

Name \_\_\_\_\_

Address \_\_\_\_\_

**DelSoft**

Gary Deley, 564 Calle Anzuelo, Santa Barbara, CA 93111  
 (805) 967-9566 eves and weekends REM

## HOW MUCH FREE SOFTWARE COULD YOU USE?

FIND OUT WITH OUR GIANT PUBLIC DOMAIN DIRECTORY

- SUPPLIED ON DISK FOR EASY COMPUTER ACCESS
- MORE THAN 4,500 ENTRIES
- SUBJECT AREAS INCLUDE:

ASTRONOMY, AVIATION, BUSINESS, EDUCATION, ENGINEERING,  
 GAMES, GRAPHICS, HAM RADIO, MUSIC, PROGRAMMING, TEXT  
 EDITING, VOICE SYNTHESIS, UTILITIES AND MUCH MORE.

Yes! I need to know what free software is available. Send me the public domain directory on the Heath CP/M 5¼ format checked

ON 3 HARD RESTORED DD \$17.00  ON 2 SOFT RESTORED DD \$12.00  ON 1 DOUBLE DD \$7.50

**HEADWARE**  
 2865 AKRON STREET  
 EAST POINT, GA. 30344

TERMS: NO RISK, MONEY back guarantee. Add \$2 domestic \$4 foreign per order for S&H. Enclose your check or M.O. with your order. Sorry, no charge or phone orders.

Name \_\_\_\_\_

Address \_\_\_\_\_

City, State, Zip \_\_\_\_\_

\*CP/M Reg. TM Digital Research Corp.

# Z-BASIC PATCH

Pat Swayne  
 Software Engineer

In this article, we present a patch to Z-BASIC revision 1.1. This version of Z-BASIC does not accept the BEEP command, with the result that many HUG game programs do not run under it. The patch can be made with the DEBUG program supplied with Z-DOS. Here are the commands you need to enter to DEBUG to make the patch to ZBASIC 1.1 dated 11-May-84.

```
-NZBASIC.COM
-L
-E227
xxxx:0227 D6.69 (hit SPACE bar here, not RETURN)
xxxx:0228 92.7C
-W
Writing B500 bytes
50,
```

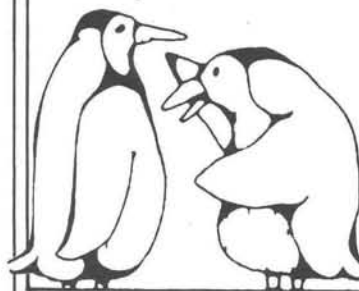
Here is the patch for ZBASIC 1.1 dated 5-Oct-84.

```
-NZBASIC.COM
-L
-E227
xxxx:0227 4A.EE (hit SPACE bar)
xxxx:0228 8F.7D
-W
Writing B700 bytes
```

This example assumes that ZBASIC.COM is on your default drive. The characters "xxxx" refer to a number which will not be the same for every application. If you are using the Z-DOS 1.x version of DEBUG, the prompt character will be ">" instead of "-" as shown. If the old data bytes at the patch address are not D6 and 92, or 4A and 8F as shown, do not make the patch. You have probably loaded the wrong version of Z-BASIC.



YOU GOTTA SEE,  
 HUG'S PRODUCTS!





# Z80 SPEEDFIX for H8/H89

Copyright © 1984  
by Frederick F. Freeland, Jr.  
Btry C, 3d BN, 7th ADA  
APO NY, NY 09139

Being stationed in Germany has its advantages, I suppose, but one of the greatest disadvantages that I can think of, is being away from what my wife describes as my "computer buddies". I really miss that warm tingly feeling that comes from a bunch of HUGGIES getting together at a local HUG meeting or a HUGCON. Despite this disadvantage, my computer and I manage to keep busy. After performing a number of modifications to my H8 system, modifying a READACTRON IBM Selectric for use with an H8 as a printer, trying to assemble an EPROM programmer, and teaching night classes in data processing, I finally found a break in my computer activities. I discovered that I had some spare time on my hands. Frankly, I was lost. It had been so long since I had seen any truly spare time that I didn't know what to do. I began having withdrawal symptoms and found myself gazing over at the computer, which sits on one wall in the living room of my government apartment. I don't know whether I was trying to justify its existence or was just itching to get on the old keyboard. There were probably a million things that I could have done, and probably should have done, but didn't. One thing that I did do, was to catch up on some long overdue correspondence to some of the aforementioned "computer buddies". With that done, I once again had some time to play with. The result of this time is the subject of this article. I simply decided that it was high time that I put something back into the Heath User's community that I have enjoyed so much for the last 6+ years.

One of the modifications that I made to my H8 system was the 3Mhz clock change described by Pat Swayne in Remark #38. Although he accurately described that feeling that I got when I ran a nice graphics game at a higher baud rate and CPU speed, I just could not bring myself to permanently modify my HDOS or CP/M systems. I like to retain maximum compatibility with anyone whose path I might cross. Permanently modifying operating systems just doesn't provide that flexibility. At last, I struck upon something useful to write. I would write some software that would allow both the H8 and the H89 user to use the small software switching routines described by Pat in his articles in REMark. And so the SPEEDFIX program was born.

The purpose of the SPEEDFIX program in all of its conditional assemblies is to provide a user with the ability to switch between two different CPU speeds under software control. This software is predicated upon the fact that the user has implemented one of the hardware modifications described by Pat Swayne, of the Heath User's Group in their monthly magazine REMark. Issue number 34 describes the appropriate modification for the H89/90 computers, and issue number 38 describes the modifications for the H8 with an HA8-6 Z80 CPU card.

Now let's describe exactly how the program works. It consists of a number of modules, each of which has a distinct purpose in the program.

The first three lines of actual code in the program define XTEXTs that will be called from the assembly process. These XTEXTs contain

definitions that are standardized in HDOS 2.0 and are provided on the 'Software Tools' disk that comes with HDOS 2.0. The user of the assembler should make sure that these XTEXTs are on the disk that contains this source code file. There are other places that they can be, but for simplicity, that is where the assembler looks for them first.

Now comes some ASCII definitions, which will be required for our assembly. They are pretty straightforward and require no further explanation.

Next we come to some basic truths...literally. In order to perform conditional assembly of any program with the HDOS assembler, we must have a means of telling the assembler whether or not we wish to assemble the various parts of our program. This is done by establishing what is TRUE and what is FALSE. The 'IF' pseudo operator of the HDOS assembler will assemble statements following an 'IF' statement if the associated argument EQUates to 0. Any other number, either negative or positive, will cause the code to not be assembled. By using this logic, we can make the label, TRUE, EQUate to 0 and the code following the 'IF' statement will be assembled. If it is EQUated to some other number, in this case 377 octal, alternate lines defined by the ELSE pseudo operator will be assembled. This is how we can use this program for two different environments, such as the application here.

The two environments, of course, are the H8 operating at 2 and 3 Mhz, and the H89 operating at 2 and 4 Mhz. Another difference in the environment is that the general purpose port (362Q) is used in a slightly different manner for each.

The two lines of code that follow the TRUE and FALSE definitions are the actual conditional assembly switches. By defining these labels appropriately as either TRUE or FALSE, we cause various parts of the code that follows to be assembled or skipped. The switch labels are H89 and PROLOG.

The program as it is listed is conditioned to assemble as an .ABS (or machine executable) program for the H8. This will allow switching of the CPU speed between 2 Mhz and 3 Mhz. Changing the code for an H89 is a very simple matter and can be done with virtually any text editor program.

By changing the H89 EQUate to TRUE, the program will, of course, assemble for an H89. One of the differences between the two versions is the mask used for the general purpose port. This is only a matter of a single different bit doing the controlling. Another difference from one computer to the other is the message that is provided to the user. The program will assemble so that there is absolutely no question in your mind about which version you are running. The H8 version states in the 'sign on' that it is for the H8 and the text describes changing CPU speeds between 2 Mhz and 3 Mhz. The H89 version signs on for an H89 and indicates speed changing between speeds of 2 Mhz and 4 Mhz.

Once the decision has been made as to what machine (H8 or H89)

you are going to assemble the program for, another option switch still remains. Changing the PROLOG EQUate to TRUE, will result in a program which can execute at bootup and simply switch the CPU to the higher CPU speed. The resultant object program can be renamed PROLOGUE.SYS and placed on a bootable disk. An undocumented, but well known HDOS feature, a prologue is a program which the HDOS operating system looks for at boot time. If a file called PROLOGUE.SYS is found on the boot disk, it will be loaded and executed. The result of this is that our program will execute at bootup and switch the CPU to the higher CPU speed without human intervention. The actual difference in the two different assemblies is simple. In the PROLOGUE version, there is no explanatory text whatsoever; the program simply signs on indicating what it is and what computer type it is for and checks the condition of the CPU speed. It will only change the speed if it is running at the low speed. If for any reason it is already running at the faster clock rate, as can occur if you boot from H37 drives which are non-clock dependent, the program will simply advise you of that fact.

Following the conditional assembly switches is the control port address EQUate. This is port 362Q, which is also known as the general purpose port. Another constant pertaining to the general purpose port needs to be defined. This constant is the address of the control byte at the label CTLBYT. The split octal address of this location is as shown. This byte contains the attributes of the general purpose port, and is the location where we store the information pertaining to the CPU speed for ready reference by the program.

After that, is the first of the conditional assembly code. The lines between the IF and the ELSE are assembled for the H89 ONLY if the H89 label has been defined as TRUE. Otherwise, the code between ELSE and ENDIF is assembled. The code in this area are the EQUates which are unique to the type of computer that the program is being assembled for. Because the H89 implementation uses a different bit of the general purpose port from the H8 to control the CPU speed, the bit masks defined by Z80.CLK are different. The two remaining differences relate to the messages that are presented in the program. The first label is HSPD, which EQUates to an ASCII character representing either a "3" or "4," depending upon the assembly. This simply insures that sign on message informs the user of the proper high speed for their computer, i.e., either 3 Mhz or 4 Mhz. The HTPP label EQUates to another ASCII character representing either a space (" ") or a "9". Depending upon the implementation, either the space or the "9" is appended to "H8" to advise the user of which computer the program was assembled for.

Finally, we get to the mainline code. The program is not written in the most memory efficient manner possible, but it is written in a structured manner, which makes it very easy to understand. The structure of the mainline code is such that each task to be performed in the program is performed by a subroutine, which is called from the mainline code. These subroutines are located following the mainline code.

We will now proceed through each task to be performed by the program and discuss its associated subroutine as we go.

The first task to be performed is the "sign on". The purpose of this part of the code is to advise the user of the title and version of the program. Additionally, a brief statement is made concerning the purpose of the program and its compatibility. The SIGNON subroutine is self-explanatory. Notice the IF..ELSE conditional assembly pseudo operators in there. The purpose of these is to eliminate all, but the title and version line, from the "sign on" if the user has decided to assemble the program as a prologue.

The next task to be performed is one that I just threw in to test for a

Z80. It does not differentiate between an H8 and H89, it simply insures that an H8 has a Z80 CPU installed. If it does not, the program will issue an error message and exit. The information at the heading of the Z80TST routine will explain the operation of the test. There is no way for the program to actually determine if the appropriate modifications have been made. However, if they have not, the only thing that will be affected is that the speed will not change. There will be no ill effects from running this program on an unmodified system.

Following the Z80 test, we want to change the terminal to the single character mode, so that we can get a response from the user at the appropriate time. This task is performed by the MODE routine. Information about the various modes can be found in the HDOS manual.

The next thing we want to do is to EXPLAIN to the user exactly what CPU speed the computer is in, and what his options are concerning that speed. The EXPLAIN subroutine evaluates the current condition of the CPU by examining the contents of the control byte. Once this has been done, the program advises the user of the current CPU speed and asks the user if he wishes to change the CPU speed. This routine solicits a response from the user regarding his desires. All responses are mapped to upper case and the default answer to any question is "YES." Once the user has made a selection, he is notified of the action resulting from that selection. If the user does not desire to change the CPU speed, the routine issues a message indicating that the CPU speed is unchanged, and the program then exits via the EXIT subroutine. It should be noted that if the PROLOG label is TRUE, only the code between the two IF and ELSE pseudos will be assembled. Conversely, if the program is not assembled as a prologue, the lines between the two ELSE and ENDIF pseudos will be assembled.

Following the EXPLAIN subroutine, we execute a routine called VERSCHK, which is used to insure that we are using a valid version of HDOS. To my knowledge, versions prior to HDOS 2.0 do not support the general purpose port. I may well be in error on this point, so if you are running HDOS 1.5 or 1.6 and everything has been working OK for you with an extended configuration, this program will work on your system. You can simply delete this subroutine and the call in the mainline code to it.

Having verified the proper version of HDOS, it is now time to get down to the business of toggling the condition of the CPU clock speed, provided of course, that the user has made that selection. The TOGGLE routine first disables the interrupts before making any changes. This was done to insure that nothing else was going to interfere with the CPU when the actual change was made. After disabling the interrupts the Z80.CLK byte, that we defined at the beginning of our listing, is exclusive or'd with the current contents of the CTLBYT. The result of this is that the bit will be changed to its opposite state, regardless of its previous condition. Once that has been done, it will be restored at the CTLBYT location for future reference and sent out the general purpose port to affect the actual change. Now that we have passed the critical stage, the interrupts can again be enabled. Once again we want to make a check on the CPU speed. This is done by AND'ing the Z80.CLK mask with the current contents. If a zero flag is set, then the CPU is operating at its low speed; if it is not zero, then it is operating at its high speed. Upon making this determination, the routine issues an appropriate message to the user indicating the current CPU operating speed.

At this point, there is only one final thing to do. That is done by the EXIT routine immediately following the mainline code. The EXIT routine clears out the type-ahead buffer, and then performs a normal HDOS exit.



```

TRUE EQU 0
FALSE EQU 377Q

** Conditional Assembly Switches

PROLOG EQU FALSE TRUE, if to assemble as PROLOGUE.SYS
H89 EQU FALSE TRUE, if to assemble for H89

** Control Port and Control Byte Address

PORT EQU 362Q General purpose port
CTLBYT EQU 040066A Control byte location

EJECT

** General Purpose Port Bit Mask and Miscellaneous
* Conditional Values for use during Assembly

IF H89
Z80.CLK EQU 00000100B Switch mask for H/Z-89/90
HSPD EQU 064Q ASCII '4' for messages
HTYP EQU 071Q ASCII '9' for H89 message

ELSE
Z80.CLK EQU 00001000B Switch mask for H8 with H8B-6
HSPD EQU 063Q ASCII '3' for messages
HTYP EQU 040Q ASCII ' ' for H8 message

ENDIF

** Program Area
ORG USERFWA

SPEEDFX EQU *
CALL SIGNON Print the program heading
CALL Z80TST Verify that there's a Z80 out there
CALL MODE Change the console to character mode
CALL EXPLAIN Tell user what we're doing
CALL VERSCHK Verify a legal version of HDOS
CALL TOGGLE Switch the Z80 to the new clock rate

***
* Clear out the type-ahead buffer,
* then exit normally to HDOS
*

EXIT EQU *
SCALL .CLRCO Clear type-ahead buffer
XRA A Normal exit
SCALL .EXIT

EJECT
SIGNON

***
* Print the program title
*

```

```

MODE
EQU
MVI A,I.CSLMD Change to character mode
MVI B,CSL.CHR
MVI C,CSL.CHR
SCALL .CONSL
RET

***
*
* Identify the current status of the
* system and the options available
*

EXPLAIN EQU
LDA CTLBYT Get the control byte
ANI Z80.CLK Check the speed
JZ EXP1 Clock is at 2 MHz

IF PROLOG
CALL $TYPTX
DB NL,'The system is operating at ',HSPD,' MHz.'
DB NL,ENL
JMP EXIT Simple exit will do

ELSE
CALL $TYPTX
DB NL,'The system is currently operating at ',HSPD,' MHz.'
DB NL,NL,'Would you like to run at 2 MHz (Y/N) <Y> ? '
DB ' '+NULL
SCALL .CLRCO Clear the type-ahead
SCALL .SCIN Get a character
JC *-2
ANI 137Q Strip parity, force upper-case
CPI NL Default is yes, also
RZ
PUSH PSW
MVI A,NL Insert newline
SCALL .SCOUT Send it
POP PSW
CPI 'Y' See if yes
RZ

CALL $TYPTX
DB NL,'The system is still operating at ',HSPD,' MHz.'
DB NL,ENL
JMP EXIT Function aborted, so exit

ENDIF

EJECT
EXP1 EQU *
IF PROLOG
RET Leave notification for later
ELSE
CALL $TYPTX
DB NL,'The system is currently operating at 2 MHz.'

```



```

SIGNON EQU *
CALL $TYPTX
DB ESC, 'E', NL
DB 'Z80 SPEEDFIX 1.0 <HDOS Version> for HB', HTYP
DB NL, '(c) Copyright 1984 by Frederick F. Freeland Jr.'
IF PROLOG
DB ENL
RET
ELSE
DB NL, NL
DB 'This program is compatible with the '
DB 'Dual Clock modifications'
DB NL
DB 'presented by the Heath Users '
DB 'Group magazine REMARK. This '
DB NL
DB 'program allows the user to toggle '
DB 'between 2 Mhz and HSPD, ' Mhz'
DB NL
DB 'CPU Speed. No other operations are '
DB 'performed and the operating '
DB NL
DB 'system is not modified in any way.'
DB ENL
RET
ENDIF
Z80TST
***
*
* Verify the presence of a Z80 CPU
*
* A 5 is moved to the A Register and then a Z80 Exchange A
* and A' instruction is performed. If the system has a Z80,
* the A Register will not contain a 5. If the system has an
* 8080 CPU the A Register will still contain a 5.
Z80TST EQU *
MVI A, 5
DB 8
CPI 5
JNZ Z80
CALL $TYPTX
DB NL, 'ERROR - A Z80 CPU is not installed.', BELL
DB NL, ENL
JMP EXIT
Z80 DB 8
Exchange back so theres no
confusion RET on the next
check of this operation
EJECT
MODE
*
* Change to character mode on the console
*
NL, NL, 'Would you like to run at ', HSPD, ' MHz (Y/N) <Y> ? '
' +NULL
.CLRCO
SCALL Clear the type-ahead
.SCIN Get a character
* -2
ANI 137q Clear parity, force upper-case
CPI NL Default is yes, also
RZ
PSW Echo a newline
MVI A, NL
.SCOUT
PSW
'Y' See if yes
RZ
CALL $TYPTX
DB NL, 'The system is still operating at 2 MHz.'
DB NL, ENL
JMP EXIT
ENDIF
VERSCHK
***
*
* Check version and verify that we're running a legal version of HDOS
*
VERSCHK EQU *
SCALL .VERS Make inquiry using SCALL
RC Error means HDOS 1.0
CPI VERS+1 Check if greater than 2.0
RM
CALL $TYPTX
DB NL, 'ERROR - Unsupported version of HDOS', BELL
DB NL, ENL
JMP EXIT
EJECT
TOGGLE
***
*
* Tell the Z80 to switch to
* the new clock rate
*
TOGGLE EQU *
DI LDA CTLBYT Disable interrupts for reliability
XRI Z80.CLK Get the present clock rate
STA CTLBYT And toggle it
OUT PORT Record it for posterity
EI Turn 'em back on else....
ANI Z80.CLK See if we're at 3 or 4 MHz now
JNZ TOGGLE1
CALL $TYPTX
DB NL, 'The system is now operating at 2 MHz.'
DB NL, ENL

```

```

RET
TOGGLE1 EQU *
CALL $TYPTX
DB NL, 'The system is now operating at ', HSPD, ' Mhz.'
DB NL, ENL
RET
END SPEEDFX
EJECT

***
Z80 SPEEDFIX 1.0 <Op/M Version> for H8/H89
(c) Copyright 1984 by Frederick F. Freeland Jr.
*
* The purpose of this program is to provide the H8 user
* with the ability to toggle back and forth between 2Mhz
* and 3Mhz while using the 3Mhz modifications to the H8-6
* Z80 CPU board presented in Remark #38 by Patrick Swayne
* of The Heath User's Group. Provision is also provided
* to toggle between 2Mhz and 4Mhz when using the H89 or H90
* Z80 CPUs per a similar modification provided by Patrick
* Swayne in Remark #34.
*
* Original: 18-Feb-84
* Latest: 5-Mar-84
*
** BDOS Definitions
BASE EQU 00H ;Start location of memory
CONIN EQU 01H ;Character input function
CONOUT EQU 02H ;Character output function
BDOS EQU 05H ;BDOS jump vector
** ASCII Definitions
BELL EQU 07H ;Bell
CR EQU 0DH ;Carriage return
LF EQU 0AH ;Line feed
ESC EQU 1BH ;Escape key
ENL EQU 80H ;End of line flag
** Definitions
TRUE EQU OFFFH
FALSE EQU NOT TRUE
** Conditional Assembly switches
PROLOG EQU FALSE
H89 EQU FALSE
H89/90 EQU TRUE, if to assemble for
command line
TRUE, if to assemble for
H89/90

```

```

DB 'presented by the Heath Users '
DB 'Group magazine Remark. This'
DB CR, LF
DB 'program allows the user to toggle '
DB 'between 2 Mhz and HSPD, ' Mhz'
DB CR, LF
DB 'CPU speed. No other operations are '
DB 'performed and the operating'
DB CR, LF
DB 'system is not modified in any way.'
DB CR, LF+ENL
RET
ENDIF

***
Z80TST
*
* Verify the presence of a Z80 CPU
*
* A 5 is moved to the A register and then a Z80 Exchange A
* and A' instruction is performed. If the system has a Z80,
* the A register will not contain a 5. If the system has an
* 8080 CPU the A register will still contain a 5.
Z80TST EQU $
MVI A, 5
DB 8 ;Z80 Instruction Exchange A and A'
CPI 5
JNZ Z80
CALL TYPTX
DB CR, LF, 'ERROR - A Z80 is not installed.', BELL
DB CR, LF+ENL
JMP EXIT
Z80 EQU $
DB 8 ;Exchange back so there's no
DB ;confusion
RET ;on the next check of this
DB ;operation
***
EXPLAIN
*
* Identify the current status of the system and options
EXPLAIN EQU $
LDA CTLBYT
ANI Z80CLK
JZ EXP1 ;Clock is at 2 MHz
IF PROLOG
CALL TYPTX
DB CR, LF, 'The system is operating at ', HSPD, ' Mhz.'
JMP EXIT
ENDIF
IF NOT PROLOG
CALL TYPTX

```

```

** Memory Definition
CTLBYT EQU 0DH ;Control byte location

** General Purpose Port Bit Mask
IF H89
EQU 00000100B ;Switch mask for H89
HSPD EQU 34H ;ASCII '4' for messages
HTYP EQU 39H ;ASCII '9' for H89 message
ENDIF

IF NOT H89
Z80CLK EQU 00001000B ;Switch mask for H8 with H8B-6
HSPD EQU 33H ;ASCII '3' for messages
HTYP EQU 20H ;ASCII ' ' for H8 message
ENDIF

* Program Area
ORG BASE+0100H ;Start 1 page above base

SPEEDFX EQU $
CALL SIGNON ;Print the program heading
CALL Z80TST ;Verify that there's a Z80
out there
CALL EXPLAIN ;Tell him what we're doing
CALL TOGGLE ;Switch the Z80 to the new
clock rate

***
* Perform a warm boot to exit CP/M normally
EXIT EQU $
JMP 0000H ;Do warm boot

***
* SIGNON
* Print the program title
*
SIGNON EQU $
CALL TYPTX
DB ESC,'E',CR,LF
DB 'Z80 SPEEDFIX 1.0 <CP/M Version> for H8',HTYP
DB CR,LF,'(c) Copyright 1984 by Frederick F. Freeleand Jr.'
DB CR,LF

IF PROLOG
DB ENL
RET
ENDIF

IF NOT PROLOG
DB ENL
ENDIF

IF 'Y'
DB ENL
ENDIF

***
* Tell the Z80 to switch to
* the new clock rate
*
*
TOGGLE EQU $
DI ;Disable interrupts for reliability

```

```

DB CR,LF,'The system is currently operating at ',HSPD,' MHz.'
DB CR,LF,LF,'Would you like to run at 2 MHz (Y/N) <Y> ? '
DB ENL

RT MVI C,CONIN
CALL BDOS
JC RT
ANI 5FH ;Strip parity, force upper-case
CR ;Default is yes, also
RZ

CPI 'Y' ;See if yes
RZ

CALL TYPTX
CR,LF
DB CR,LF,'The system is still operating at ',HSPD,' MHz.'
DB CR,LF+ENL

JMP EXIT ;Function aborted, so exit
ENDIF

EXP1 EQU $
IF PROLOG
RET
ENDIF
IF NOT PROLOG
CALL TYPTX
DB CR,LF,'The system is currently operating at 2 MHz.'
DB CR,LF,LF,'Would you like to run at ',HSPD,' MHz (Y/N) <Y> ? '
DB ENL

RT1 MVI C,CONIN
CALL BDOS
JC RT1
ANI 5FH ;Get reply
CPI CR ;Clear parity, force upper-case
RZ ;Default is yes, also

CPI 'Y' ;See if yes
RZ

CALL TYPTX
CR,LF
DB CR,LF,'The system is still operating at 2 MHz.'
DB CR,LF+ENL

JMP EXIT
ENDIF

***
*
*
*
*
TOGGLE EQU $
DI

```

```

LDA CTLBYT ;Get the present clock rate
XRI Z80CLK ;And toggle it
STA CTLBYT ;Record it for posterity
EI ;Turn 'em back on or else....
JNZ Z80CLK ;See if we're at 3 MHz now
TOGGLE1
CALL TYPTX
DB CR,LF
DB CR,LF,'The system is now operating at 2 Mhz.',CR,LF+ENL
RET

TOGGLE1 EQU $
CALL TYPTX
DB CR,LF
DB CR,LF,'The system is now operating at '
DB ',HSPD,' Mhz.',CR,LF+ENL
RET

***
*
*
*
TYPTX
Type the text following the call to this subroutine

TYPTX EQU $
XTHL ;Save HL, get address of text
MOV ;Get character
PUSH ;Save pointer
MVI C, CONOUT ;Character out function
ANI 7FH ;Strip marker bit
MOV E,A ;Send character to E Reg
CALL BDOS ;Print it
POP H ;Restore the pointer
MOV A,M ;Get another character
INX H ;Increment pointer
ORA A ;Test for end
JP TYPTX1 ;Nope, not yet
XTHL ;ok, fix stack and get HL
RET

END SPEEDFX

```



# H-1000

## A Z80/8086 UPGRADE FOR THE H89/Z89

### HARDWARE

- plug-in replacement for the H89/Z89 CPU board; no modifications required
- dual CPUs: Z80 and 8086
- 256K RAM standard; sockets for up to 1 megabyte RAM
- 5 I/O slots
- faster program execution: 2/4 MHz for Z80, 8MHz for 8086
- fully compatible with all Heath/Zenith peripherals

### SOFTWARE

- runs all Heath/Zenith software without modification
- compatible with Zenith Z100 and IBM Personal Computer
- choice of MSDOS or CP/M-86 for the 8086
- supplied with diagnostic software package
- "soft disk" feature: copies an entire disk in RAM for instant disk access
- supports multi-user and multi-task operating systems



## TMSI

Technical Micro Systems Inc.

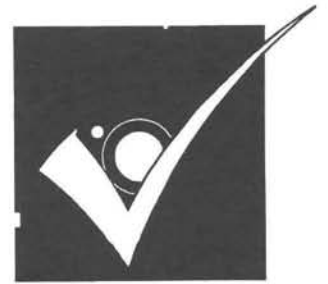
P.O. Box 7227, Dept. H  
366 Cloverdale • Ann Arbor, Michigan 48107  
(313) 994-0784

We accept MasterCard and VISA.  
Serious Dealer Inquiries Invited

The H-1000 is a quality product of Technical Micro Systems, Inc., manufacturers of innovative microcomputer systems since 1979.

H-1000 and TMSI are trademarks of Technical Micro Systems, Inc. H89, Z89, Z100 and HDOS are trademarks of Heath/Zenith Corp., Benton Harbor, Michigan. MSDOS is a trademark of Microsoft, Bellevue, Washington. CP/M and CP/M-86 are trademarks of Digital Research, Inc., Pacific Grove, California.

# CHECKSUM: A Program Proofreading Aid



Stephen A. Jacob  
1435 D Paegelow  
Scott AFB, IL 62225

It's a tedious task to key a magazine program into a computer. And once keyed, there are always those illusive Os that should be Os or 1s that should be Is. You carefully key the program and search for typos. You run it. You miss some errors. You repeat the process. The great computer cliché says: There has got to be a better--computerized--way.

CHECKSUM is one better way. It reads the program you typed and outputs a symbol for each program line. If that "checksum" symbol does not match the author's published checksum, you have isolated an error.

Suppose you run CHECKSUM on a program "TEST.BAS" and you get the following:

CHECKSUM PER LINE:

meCDE ZB

TOTAL CHECKSUM: F

The published checksum information may be as follows:

CHECKSUM PER LINE:

mEbDE ZB

TOTAL CHECKSUM: q

The TOTAL CHECKSUMS do not match (q vice F), signaling the files are not identical. The CHECKSUM PER LINE reveals the third elements differ (b vice C). The third CHECKSUM element is the third line of TEST.BAS, so that program line has an error.

That's how CHECKSUM is used. A couple of caveats are in order. First, CHECKSUM ignores all spaces. If you have an expression like `PRINT CHR$(27)+"Y4"`, CHECKSUM will not see the error if the space is left out of "Y4". Second, CHECKSUM may miss one error in 52 as it uses only 52 symbols.

Of course, CHECKSUM could check for spaces in quotes. And CHECKSUM could use more than 52 symbols--the printable ASCII set. But the first is a minor BASIC related problem and the second would complicate the display.

While CHECKSUM is written in BASIC and illustrated on a BASIC program, it works equally well checking Pascal, Assembly, you name it. I leave it to these programmers to write a version in their language. I request only that you maintain the checksum algorithm and keep all checksums standard regardless of language.

## About the algorithm:

The CHECKSUM algorithm is implemented in the "read file and checksum subroutine" (Listing 1). This subroutine takes each line character by character. It converts these characters to a number from 1 to 52. The number is then converted to a value representing an ASCII character--1 converts to 65 for A. The total for the line is stored in the array S\$( ).

Each element in the array S\$( ) is a line's checksum. The first element, S\$(1), is the checksum of the first line. Similarly for each program line. S\$(0) stores a checksum of the checksums.

To get CHECKSUM up and running:

1. Using a text editor, key the abbreviated control program (Listing 2). Save this ASCII file as SHRTCON.BAS.
2. Similarly, key the readfile and checksum subroutines (Listing 1) into an ASCII file labelled RDFILE.SUB.
3. Make a third file called SHORTCS.BAS by merging RDFILE.SUB into SHRTCON.BAS. This is the "minimal" CHECKSUM program.
4. Once you have debugged the abbreviated program SHORTCS.BAS, make a backup copy. Label the backup copy SHORTCS.BAK.
5. Run SHORTCS.BAS on SHORTCS.BAK. The results should be as follows:

CHECKSUM PER LINE:

QfpTj sBTZg nUXW hpReo tbXvt LpPHA FDPYy CTMcu QtWyu eOhLK YxMQV BF

TOTAL CHECKSUM: r

6. If necessary, debug the appropriate lines and try again.

SHORTCS.BAS may be all you will ever need to CHECKSUM published programs; however, you may want a menu, a disk save, and some error checking. The following provides these enhancements.

1. When you are satisfied SHORTCS.BAS works, set it aside.
2. Key the CHECKSUM control program (Listing 3). Notice Listing 3 has code for BASIC80 and ZBASIC. Type these as shown without modification. They will be removed/modified later. Label this CS1.BAS
3. MERGE RDFILE.SUB into CS1.BAS. Rename this CS2.BAS.
4. Key in Listing 5 and name it CSSAVE.SUB. MERGE CSSAVE.SUB

into CS2.BAS.

5. RENAME CS2.BAS to CS.BAS. Make a backup CS.BAK.

6. Use SHORTCS.BAS to CHECKSUM CS.BAS. The results should be as follows:

CHECKSUM PER LINE:

```
SFlJb MPqrp gZmNQ psrat lIVBp KoHtC mNoiQ TAuuV PXphv
xtoEw JTzgn tUXWh pReot
bXvtL pPHAF DPYyC TMcuQ tWyue OhLKY xMQVB FxmNQ
pQbvW MqnhO kXrZz ibuxu KVOCK KBVpj
```

TOTAL CHECKSUM: L

7. Debug CS.BAS and use SHORTCS.BAS one more time to CHECKSUM it.

8. Delete inappropriate BASIC80 or ZBASIC code to clean up CS.BAS, then make a backup.

9. The other files created are no longer needed and may be deleted.

10. Use the SAVE command of CHECKSUM whenever you submit a program for publication.

#### Listing 1: Read file and checksum subroutine

```
15100 'read file and checksum subroutine
15200   J=0 'storage array location for each line
        'checksum
15300   T=0 'total of all checksums (mod 52)
15400   S=0 'first line checksum (mod 52)
15500 '
15600 ' CLS
15700 PRINT CHR$(27)+"E";
15800 PRINT T$
15900 PRINT:PRINT "ENTER INPUT FILENAME.EXT: ";
16000 '
16100 LINE INPUT F$
16200 OPEN "I",1,F$
16300   PRINT:PRINT "CHECKSUM PER LINE:";
16400 '
16500 '   loop 2: read file until exhausted
16600   IF EOF(1) THEN 17500
16700 '   false
16800     J=J+1:S=0
16900     LINE INPUT #1,L$
17000     GOSUB 18500
17100     IF J MOD 5 =1 THEN PRINT " ";
17200     IF J MOD 65=1 THEN PRINT
17300     PRINT S$(J);
17400     GOTO 16600
17500 '   true
17600     CLOSE #1
17700 '   end loop 2
17800 '
17900 'write total
18000 .PRINT:PRINT:PRINT "TOTAL CHECKSUM: ";
18100 T=T+64:IF T>90 THEN T=T+6
18200 S$(0)=CHR$(T):PRINT S$(0)
18300 '
18400 RETURN
18500 'checksum sub
18600   K=0:L1=LEN(L$)
18700 '
18800 '   loop 3: determine line checksum
18900   IF L1=0 THEN 19400 ELSE K=K+1:L1=L1-1
19000     A$=MID$(L$,K,1)
19100     IF A$<>" " THEN X=ASC(A$)-32 ELSE X=0
19200     S=(S+X-1) MOD 52 +1
19300     GOTO 18800
19400 '   end loop 3
19500 '
19600 T=(T+S-1) MOD 52+1
19700 S=S+64:IF S>90 THEN S=S+6:S$(J)=CHR$(S)
19800 S$(J)=CHR$(S)
```

```
19900 '
20000 RETURN
```

#### Listing 2: Abbreviated control program

```
10 T$="CHECKSUM: PROGRAM PROOFREADING AID"
20 'abbreviated version
30 '
40 DIM S$(200)
50 GOSUB 15100
60 '
70 END
```

#### Listing 3: CHECKSUM control program

```
10000 'CHECKSUM: Program Proofreading Aid
10100 'By Stephen A. Jacob; June 20, 1984
10200 '
10300 'initialization block
10400   C=27:M=200
10500   DIM S$(M)
10600   P$=""
10700   T$="CHECKSUM: Program Proofreading Aid"+P$+"
        ver .6"
10800 '
10900 'The next two lines clear the screen for ZBASIC or
11000 'BASIC80. Run CHECKSUM on itself with both lines in
11100 'and both REMARKED. Then remove the inappropriate
11200 'lines or REMARKS throughout.
11300 ' CLS
11400   PRINT CHR$(27)+"E";
11500 '
11600   ON ERROR GOTO 22800
11700 '
11800 'loop 1: control program
11900 '
12000   GOSUB 13700 'menu
12100   IF M$="R" OR M$="r" THEN GOSUB 15100 'read file
12200   IF M$="S" OR M$="s" THEN GOSUB 20100
        'save checksum
12300   IF M$="Q" OR M$="q" THEN 13100 'quit
12400 '
12500 ' LOCATE 23,C
12600 ' PRINT CHR$(27)+"Y77";
12700 ' PRINT "Press any key to continue. ";
        :A$=INPUT$(1)
12800 '
12900   GOTO 11800
13000 '
13100 'end loop 1
13200 '
13300 'return all parameters to normal
13400 ' CLS
13500   PRINT CHR$(27)+"E";
13600 END
```

#### Listing 4: Menu subroutine

```
13700 'menu subroutine
13800 '
13900 ' CLS
14000 PRINT CHR$(27)+"E";
14100 PRINT T$:PRINT:PRINT
14200 '
14300 PRINT TAB(C);"R - Read a file from disk"
14400 PRINT TAB(C);"S - Save checksum to disk"
14500 PRINT TAB(C);"Q - Quit"
14600 '
14700 PRINT:PRINT:PRINT:PRINT TAB(C+5);
        "ENTER SELECTION: ";
14800 LINE INPUT;M$
14900 '
15000 RETURN
```

#### Listing 5: Save checksum and error checking routines

```
20100 'save checksum to disk subroutine
20200 '
20300 ' CLS
20400 PRINT CHR$(27)+"E";
20500 '
```

```

20600 PRINT T$
20700 IF J=0 THEN GOSUB 15100
'If no sums before, go to read file subroutine
20800 PRINT:PRINT
"ENTER OUTPUT FILENAME (extention will be .CKS): ";
20900 K=0
21000 LINE INPUT F$
21100 F$=F$+".CKS"
21200 '
21300 OPEN "0",1,F$
21400 PRINT #1,"CHECKSUM PER LINE: ";
21500 ' loop 4: print checksum to disk
21600 IF K=J THEN 22300
21700 ' false
21800 K=K+1
21900 IF K MOD 5=1 THEN PRINT #1," ";
22000 IF K MOD 65=1 THEN PRINT #1,
22100 PRINT #1,S$(K);
22200 GOTO 21500
22300 ' true
22400 PRINT #1,:PRINT #1,:PRINT #1,
"TOTAL CHECKSUM: ";S$(0)
22500 CLOSE #1
22600 ' end loop 4
22700 RETURN
22800 'error routine
22900 ' LOCATE 20,10
23000 ' PRINT CHR$(27)+"Y4 ";
23100 PRINT:PRINT TAB(10);
23200 PRINT
"An error occurred. Check filename and disk
space available."
23300 CLOSE
23400 RESUME 12700

```



## H/Z-100 AND 150 PC GRAPHICS SOFTWARE

MICROSERVICES continues to expand its support of Heath/Zenith computers.

For H/Z-100s:

**ZANIMATE.** Make color animation sequences using ZBASIC ..... **\$64.95**

**ZPALETTE.** Draw images in 92 colors using ZBASIC. Improved with three (3) painting modes and graphics generator ..... **\$59.75**

**ZPATTERN.** Test your color monitor ..... **\$24.95**

**Z3D** from **COLORWORKS.** Make three-dimensional objects in color ..... **\$75.00**

For H/Z-100s and 150 PCs:

**SOFTKITS** from **KERN INTERNATIONAL.** We are now carrying a full line of this excellent series of books and disks for graphics and business/scientific applications.

*We are continuing to introduce new products. Write for our catalog.*



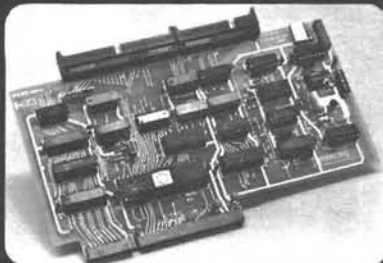
**MICROSERVICES**  
P.O. Box 7093  
Menlo Park, CA 94026  
Phone: (415) 851-3414



Include 3% p/h (\$3.00 min.). California add 6% tax.

## 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

## 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.

# *The one-stop microcomputer shopping center*

**your strong partner in  
microcomputers**

**5 NEW printers  
and plotters**

**2 NEW IBM  
PC compatible  
computers**

**NEW best-selling  
IBM PC compatible  
software**

**NEW IBM PC  
plug-compatible  
circuit boards**

*See everything that's new at your...*

**Heathkit<sup>®</sup>**

Electronic

Center

Where you get more by doing



# Advanced Assembly Language Programming:

*Long Relative Addressing for the 8080 and Z80*



Pat Swayne  
Software Engineer

Microprocessors point to locations in memory in two basic ways: by specifying the address explicitly, or by specifying it relative to another address. When the second form is used exclusively in a program, the program can be located anywhere in memory. Newer microprocessors, such as the 8088 in the Z-100, use this form, called "relative addressing", almost entirely, and the programmer is hardly ever concerned with where a program is actually located in memory when it runs.

The 8080 microprocessor (used in the H8), however, provides no relative addressing at all, and its cousin, the Z80 (used in H89's and some H8's) provides only short relative addressing (-129 to +127 bytes from the current instruction) for a few control transfer (jump) instructions. Methods have been devised to provide relocatable programs for 8080 and Z80 processors such as the Digital Research PRL (Page ReLocatable) method in which a table describes absolute addresses in a program that must be changed when it is relocated. A loader program is required to interpret the table when the program is located to its actual running address. The program is not really relocatable, it is just patched. To achieve true relocatability with the 8080 or Z80, you need a way of specifying addresses as offsets from a known point as the Z80 does with its relative jumps, but within the entire 64k of addressable memory, and you must be able to do it for data transfer as well as control transfer. In this article, I will present a way of obtaining this kind of addressing, called long relative addressing, with the 8080 and Z80 for both types of transfers.

In microprocessors that have relative addressing capability, the reference point to which the desired address is related is often (but not always) the instruction pointer. Therefore, if you could somehow obtain the value of the instruction pointer at any point in a program, you could implement relative addressing. In the 8080, as in most micros, the value of the instruction pointer is saved in memory ("pushed" onto the "stack") whenever a subroutine is called. A subroutine that obtained that value and returned it to the calling program would make relative addressing possible, and in the 8080, it can be done with only two instructions: XTHL and PCHL. The first instruction, XTHL, exchanges the value in the HL register pair with the value of the instruction pointer on the stack. The second, PCHL, transfers control to that address, which is back in the main program. The old value of the HL register pair is still on the stack, so no data has been destroyed. The program can now add the distance to a desired address to the HL register pair and calculate the absolute value of that address.

The only problem with this approach is where to put the two-byte subroutine. The best place is at an unused restart (interrupt) vector location, because then it can be called using a single byte (software interrupt) instruction, RST. The relocatable program itself can place

the two instructions XTHL and PCHL into memory at the restart address.

Listing 1 is a set of macros designed for use with Digital Research's MAC macro assembler that implements relative addressing using this technique. The function of each macro is explained in the listing. By using macros, mnemonics that provide long relative addressing versions of normal 8080 instructions can be used to make writing a relocatable program easy. The basic technique used in each macro is to first obtain the value of the instruction pointer by calling the special subroutine (RST 6). A local label (HERE) immediately following the subroutine is subtracted at assembly time from the desired address to get the distance from HERE to the address. This distance will be a positive number if the address is after HERE, and a negative number if it is before HERE. The distance is then added to the instruction pointer value to obtain the absolute value of the desired address. Note that neither your specified address nor the address of HERE are assembled into the program, but only the distance between. That means that if a program is written using only these macros, it will contain no absolute addresses except those outside the program that are referenced.

The subroutine return instruction (RET) is used in all of the control transfer macros for the actual transfer, and conditional returns are used for conditional transfers. Because of the way the MAC assembler handles instruction names, they can be used as arguments to other instructions. This makes it possible to use a conditional return instruction to specify the condition for a conditional transfer macro. The result is that only one conditional jump macro and one conditional call macro had to be written.

A practical application of this method of making programs relocatable is the SEARCH program in listing 2. This program provides a memory search extension to the DDT debugging program that is provided with CP/M. The assembled .COM file of this program can be loaded anywhere into memory with DDT's R command and then executed whenever you need to search an area of memory for a word or byte constant. For example, if you wanted to load it in at 8000H, you could enter

```
-ISEARCH.COM  
-R7F00
```

The offset 7F00 must be used because the starting point is the CP/M Transient Program Address (TPA), which is 100H. To execute the program, use the G command to jump to 8000H, and the program will display the letter S followed by a colon (:) for a prompt. The syntax for using the program is

```
S:<start addr>,<end addr>,<constant><W>
```

The letter W must be typed after the constant if you want to search for a Word (16 bits). Otherwise, the program performs a byte search (8 bits). Each time the program locates your word or byte, it prints the address where it was found until it reaches the end address, and then it re-displays its prompt. To exit the program and return to DDT, just type a period (.) at the prompt. The program preserves all registers when it is running so it can be used while you are debugging a program. The Program Counter (instruction pointer) register is, of course, not preserved, so you should note its value before you run the program. You can run the search program as a stand-alone CP/M program by removing the PUSH instructions at the beginning, removing the POP instructions below the label EXIT, and replacing the RST 7 instruction with a RET.

The search program represents only one use for fully relocatable code. Another good use would be a ROM (Read Only Memory) monitor program that could be "plugged in" to any address. This article illustrates that with proper programming a microprocessor's limitations can be overcome. It also points out that software is a major factor in determining the efficiency of any computer.

```

; LONGREL - LONG (16-BIT) RELATIVE ADDRESSING
; FOR THE 8080 PROCESSOR
;
; THESE MACROS PROVIDE 16-BIT RELATIVE ADDRESSING
; FOR 8080, Z80, AND SIMILAR PROCESSORS. CODE THAT
; IS WRITTEN USING THESE MACROS EXCLUSIVELY FOR
; JUMPS, CALLS, AND DATA STORAGE AND RETRIEVAL
; WITHIN THE PROGRAM CAN BE RUN ANYWHERE IN MEMORY.
;
; TO USE THESE MACROS, THE FOLLOWING INSTRUCTIONS
; MUST BE PLACED AT THE "RESTART 6" VECTOR:
;
; XTHL          ; GET RETURN, SAVE HL
; PCHL          ; RETURN WITH HL = PC
;
; BY P. SWAYNE, HUG 28-SEP-83
;
; NOTE: ALL OF THESE MACROS DESTROY THE CARRY FLAG,
; EXCEPT FOR THE CONDITIONAL JUMP AND CALL

```

JRL - JUMP RELATIVE LONG

```

JRL:  MACRO  ADDR
      LOCAL  HERE
      RST    6          ;; GET PC IN HL
HERE:  PUSH   D          ;; SAVE DE
      LXI   D,ADDR-HERE ;; GET OFFSET TO DESTINATION
      DAD   D          ;; ADD IT TO THE PC VALUE
      POP   D          ;; RESTORE DE
      XTHL          ;; PUT ADDR ON STACK, GET HL
      RET    2         ;; TAKE THE JUMP
      ENDM

```

JRLF - JUMP RELATIVE LONG, FASTER VERSION  
THIS MACRO DESTROYS THE DE REGISTERS.

```

JRLF:  MACRO  ADDR
      LOCAL  HERE
      RST    6          ;; GET PC IN HL
HERE:  LXI   D,ADDR-HERE ;; GET OFFSET TO DESTINATION
      DAD   D          ;; CALCULATE DESTINATION
      XTHL          ;; PUT ADDR ON STACK, GET HL
      RET    2         ;; TAKE THE JUMP
      ENDM

```

JRLC - JUMP RELATIVE LONG CONDITIONAL  
THE CONDITION FOR THE JUMP IS SPECIFIED IN THE FORM OF A RETURN INSTRUCTION OF LIKE CONDITION

```

; AFTER THE ADDRESS, AS THIS EXAMPLE OF JMP ON ZERO.
;
; JRLC  DEST,RZ          ; JUMP ON ZERO
;
; DEST:  _____    ; COME HERE IF ZERO FLAG SET

```

```

JRLC:  MACRO  ADDR,COND
      LOCAL  HERE
      RST    6          ;; GET PC ADDRESS
HERE:  PUSH   D          ;; SAVE DE
      PUSH  PSW         ;; SAVE FLAGS
      LXI   D,ADDR-HERE
      DAD   D          ;; CALCULATE DEST. ADDRESS
      POP   PSW         ;; RESTORE FLAGS
      POP   D          ;; RESTORE DE
      XTHL          ;; PUT DEST. ON STACK, GET HL
      DB    COND        ;; EXECUTE THE JUMP
      INX   SP         ;; OR FIX THE STACK
      ENDM

```

CRL - CALL RELATIVE LONG  
CALL THE ROUTINE AT THE SPECIFIED ADDRESS

```

CRL:  MACRO  ADDR
      RST    6          ;; GET PC IN HL
      PUSH  D          ;; SAVE DE
      LXI   D,16       ;; OFFSET TO CODE AFTER THIS
      DAD   D          ;; CALCULATE RETURN ADDRESS
      POP   D          ;; RESTORE DE
      XTHL          ;; PUT RETURN ON STACK, GET HL
      JRL  ADDR        ;; JUMP TO THE ROUTINE
      ENDM

```

CRLC - CALL RELATIVE LONG CONDITIONAL  
THIS MACRO WORKS LIKE JRLC, WITH THE CONDITION SPECIFIED BY A RETURN INSTRUCTION AFTER THE ADDRESS.

```

CRLC:  MACRO  ADDR,COND
      RST    6          ;; GET PC IN HL
      PUSH  D          ;; SAVE DE
      PUSH  PSW         ;; SAVE FLAGS
      LXI   D,24       ;; OFFSET TO CODE AFTER THIS
      DAD   D          ;; CALCULATE RETURN ADDRESS
      POP   PSW         ;; RESTORE FLAGS
      POP   D          ;; RESTORE DE
      XTHL          ;; PUT RETURN ON STACK, GET HL
      JRLC ADDR,COND   ;; EXECUTE THE SUBROUTINE
      INX   SP         ;; OR FIX THE STACK
      ENDM

```

SRPR - STORE REGISTER PAIR RELATIVE  
(REPLACES SHLD, AND Z80 SDED AND SBDC)  
STORE THE CONTENTS OF THE SPECIFIED REGISTER PAIR AT A RELATIVE LOCATION. THIS MACRO DESTROYS THE DE REGISTERS, IF THE DATA REGISTER PAIR IS HL.

```

SRPR:  MACRO  ADDR,REG
      LOCAL  HERE
      RST    6          ;; GET PC ADDRESS
HERE:  DS    0
      IF    REG NE H
      PUSH  REG         ;; PUT DATA REGISTER ON STACK
      ENDIF
      IF    REG EQ B
      LXI   B,ADDR-HERE
      DAD   B          ;; CALCULATE STORE ADDRESS
      POP   B          ;; GET DATA REGISTER VALUE
      MOV   M,C        ;; STORE IT
      INX   H
      MOV   M,B

```

```

ELSE
LXI D,ADDR-HERE
DAD D          ;; CALCULATE STORE ADDRESS
POP D          ;; GET DATA REGISTER VALUE
MOV M,E       ;; STORE IT
INX H
MOV M,D
ENDIF
IF REG NE H
POP H          ;; RESTORE HL
ELSE
XCHG          ;; RESTORE HL VALUE FROM DE
ENDIF
ENDM

```

```

; LRPR - LOAD REGISTER PAIR RELATIVE
; (REPLACES LHL, AND Z80 LDED AND LBCD)
;
; LOAD THE SPECIFIED REGISTER PAIR WITH THE
; DATA AT A RELATIVE LOCATION.
; DESTROYS DE IF REG = HL

```

```

LRPR: MACRO REG,ADDR
LOCAL HERE
RST 6          ;; GET PC ADDRESS
HERE: DS 0
IF REG EQ B
LXI B,ADDR-HERE
DAD B          ;; CALCULATE STORE ADDRESS
MOV C,M        ;; LOAD DATA TO BC
INX H
MOV B,M
ELSE
LXI D,ADDR-HERE
DAD D
MOV E,M        ;; LOAD DATA TO DE
INX H
MOV D,M
ENDIF
IF REG EQ H
POP H          ;; FIX STACK
XCHG          ;; PUT RESULT IN HL
ELSE
POP H          ;; RESTORE HL
ENDIF
ENDM

```

```

; LRI - LOAD RELATIVE IMMEDIATE
; (REPLACES LXI REG,DATA)
;
; LOAD THE SPECIFIED REGISTER PAIR WITH A
; RELATIVE ADDRESS. TO SET THE STACK POINTER
; TO A RELATIVE ADDRESS, USE

```

```

; LRI H,ADDR    ; PUT THE VALUE IN HL
; SPHL          ; AND THEN INTO SP
;
LRI MACRO REG,ADDR
LOCAL HERE
RST 6          ;; LOCATE PC ADDRESS
HERE: DS 0
IF REG NE D
PUSH D         ;; CONDITIONALLY SAVE REGISTERS
ENDIF
LXI D,ADDR-HERE
DAD D          ;; CALCULATE ADDRESS TO LOAD
IF REG NE D
POP D          ;; RESTORE DE
ENDIF
IF REG NE H
PUSH H         ;; PUT RESULT ON STACK
POP REG        ;; TRANSFER IT TO TARGET REGISTER
POP H          ;; RESTORE HL
ELSE
XTHL          ;; PUT RESULT ON STACK
POP H          ;; GET IT, FIX STACK

```

```

ENDIF
ENDM

```

```

; SRR - STORE 8-BIT REGISTER RELATIVE
; (REPLACES STA, AND PROVIDES B AND C REG. STORE)
;
; STORE THE CONTENTS OF THE 8-BIT REGISTER
; AT A RELATIVE LOCATION. THE REGISTER MUST
; BE A, B, OR C. DESTROYS DE

```

```

SRR MACRO ADDR,REG
LOCAL HERE
RST 6          ;; LOCATE PC
HERE: LXI D,ADDR-HERE
DAD D          ;; CALCULATE STORE ADDRESS
MOV M,REG      ;; STORE REGISTER DATA
POP H          ;; RESTORE HL
ENDM

```

```

; LRR - LOAD 8-BIT REGISTER RELATIVE
; (REPLACES LDA, AND PROVIDES B AND C REG. LOAD)

```

```

; LOAD THE SPECIFIED 8-BIT REGISTERS WITH
; THE DATA AT A RELATIVE LOCATION. THE REGISTER
; MUST BE A, B, OR C. DESTROYS DE

```

```

LRR MACRO REG,ADDR
LOCAL HERE
RST 6          ;; LOCATE PC
HERE: LXI D,ADDR-HERE
DAD D          ;; CALCULATE LOAD ADDRESS
MOV REG,M      ;; LOAD DATA
POP H          ;; RESTORE HL
ENDM

```

```

* SEARCH - MEMORY SEARCH FOR DDT

```

```

* THIS PROGRAM PROVIDES A MEMORY SEARCH
* COMMAND TO AID IN PROGRAM DEBUGGING WITH
* DIGITAL RESEARCH'S DDT.

```

```

* IT USES THE LONGREL MACRO LIBRARY TO
* MAKE IT A COMPLETELY POSITION INDEPENDENT
* PROGRAM.

```

```

* BY P. SWAYNE, HUG 28-SEP-83

```

```

MACLIB LONGREL          ;INCLUDE RELATIVE MACROS

```

```

* CP/M DEFINITIONS

```

```

BDOS EQU 5              ;BDOS VECTOR
CONOUT EQU 2            ;CONSOLE OUTPUT
PRINT EQU 9             ;PRINT FUNCTION
RCBUF EQU 10            ;READ CONSOLE BUFFER

```

```

ORG 100H                ;START HERE, FOR NOW!

```

```

START: PUSH H!PUSH D!PUSH B!PUSH PSW      ;SAVE ALL REGISTERS

```

```

LXI H,0E9E3H           ;GET "XTHL, PCHL"
SHLD 60Q               ;SET RST 6 VECTOR
LXI H,0

```

```

DAD SP                 ;LOCATE USER STACK
SRPR USRSTAK,H        ;SAVE IT

```

```

MLOOP: LRI H,STACK
SPHL                  ;SET LOCAL STACK

```

```

LRI D,PROMPT
MVI C,PRINT
CALL BDOS              ;PRINT PROMPT CHARACTER

```

```

LRI D,INBUF
PUSH D                 ;SAVE BUFFER ADDRESS

```

```

MVI C,RCBUF
CALL BDOS              ;GET USER'S INPUT

```

```

POP H ;HL = BUFFER
INX H ;POINT TO COUNT
MOV A,M ;GET IT
ORA A ;NULL ENTRY?
JRLC MLOOP,RZ ;IF SO, TRY AGAIN
MOV E,A ;ELSE, PUT COUNT IN E
MVI D,0 ;DE = COUNT
INX H ;POINT TO COMMAND
PUSH H ;SAVE THIS ADDRESS
DAD D ;MOVE TO END
MVI M,13 ;TERMINATE COMMAND
POP D ;GET COMMAND START
LDAX D ;GET FIRST COMMAND LETTER
CPI ' ' ;PERIOD?
JRLC EXIT,RZ ;IF SO, EXIT
CRL RDNUM ;READ NUMBERS ENTERED
JRLC MLOOP,RC ;BAD ENTRY
POP PSW ;GET DELIMITER
POP B ;GET SEARCH NUMBER
POP D ;GET END
POP H ;GET START
CPI 'W' ;WORD SEARCH?
JRLC SLOOP1,RNZ ;NO, BYTE SEARCH

* WORD SEARCH LOOP
SLOOP: MOV A,M ;GET A CHARACTER
CMP C ;TEST
INX H ;MOV TO NEXT CHAR
JRLC NEXT,RNZ ;NO MATCH
MOV A,M
CMP B ;TEST HIGH BYTE
NEXT: DCX H ;BACK UP TO WORD START
CRLC FOUND,RZ ;FOUND MATCH
INX H ;SKIP TO NEXT POSITION
CRL CPHD ;TEST FOR END
JRLC SLOOP,RNC ;LOOP UNTIL END
JRL MLOOP ;GET NEXT INPUT

* BYTE SEARCH LOOP
SLOOP1: MOV A,M ;GET A CHARACTER
CMP C ;TEST
CRLC FOUND,RZ ;PRINT ADDRESS IF MATCH
INX H ;SKIP TO NEXT BYTE
CRL CPHD ;TEST FOR END
JRLC SLOOP1,RNC ;LOOP UNTIL END
JRL MLOOP ;ELSE, GET NEXT INPUT

* FOUND MATCH, PRINT ADDRESS
FOUND: PUSH H ;SAVE REGISTERS
PUSH D
PUSH B
MVI E,13
MVI C,CONOUT ;PRINT CR
CALL BDOS
MVI E,10
MVI C,CONOUT ;PRINT LF
POP B ;RESTORE REGISTERS
POP D
POP H
MOV A,H ;GET HIGH BYTE
CRL PBYTE ;PRINT IT
MOV A,L ;GET LOW BYTE
PBYTE: PUSH PSW ;SAVE BYTE
RLC ;MOVE HIGH NIBBLE DOWN
RLC
RLC
RLC
ANI 0FH ;ISOLATE IT
CRL PNIB ;PRINT IT
POP PSW ;GET BYTE
ANI 0FH ;ISOLATE LOW NIBBLE
PNIB: ADI 90H ;"A" OR GREATER SETS CARRY
DAA
ACI 40H ;ADJUST

```

```

DAA
PUSH B ;SAVE REGISTERS
PUSH D
PUSH H
MOV E,A ;CHARACTER TO E
MVI C,CONOUT
CALL BDOS ;PRINT CHARACTER
POP H ;RESTORE REGISTERS
POP D
POP B
RET

* COMPARE HL TO DE, SET CARRY IF HL > DE
CPHD: MOV A,H
ORA L
STC
RZ ;DON'T ALLOW WRAP-AROUND
MOV A,E
SUB L ;SUBTRACT
MOV A,D
SBB H ;SUBTRACTION DETERMINES
RET ; RESULT

* EVALUATE INPUT NUMBERS
RDNUM: MVI C,3 ;GET THREE NUMBERS
RDNUMA: LXI H,0 ;CLEAR HL
RDNUM0: LDAX D ;GET A CHARACTER
INX D
RDNUM1: MOV B,A ;SAVE IT
CRL BIN ;CONVERT TO BINARY
JRLC RDNUM2,RC ;NOT A NUMBER
DAD H ;MOVE LAST ENTRY
DAD H ;OVER 4 PLACES
DAD H
ORA L
MOV L,A ;ADD LATEST ENTRY
JRL RDNUM0 ;GET ANOTHER ENTRY
RDNUM2: XTHL ;SWAP RETURN ADDR, HL
PUSH H ;REPLACE RETURN ADDR
MOV A,B ;GET LAST ENTRY
CPI ',' ;COMMA?
JRLC RDNUM3,RZ ;YES
RDNUMX: MOV A,B ;ELSE, GET DELIMITER
POP H ;GET RETURN ADDRESS
PUSH PSW ;PUT DELIMITER ON STACK
PUSH H ;RESTORE RETURN ADDRESS
ORA A ;CLEAR CARRY
DCR C ;ENOUGH NUMBERS?
RZ ;YES, RETURN OK
STC ;ELSE, MARK ERROR
RET ;AND RETURN
RDNUM3: DCR C ;COUNT THIS NUMBER
JRLC RDNUMA,RNZ ;LOOP IF NOT LAST
INR C ;FIX C
JRL RDNUMX ;ELSE, EXIT

* CONVERT BINARY TO ASCII
BIN: SUI '0'
RC ;LESS THAN "0"
ADI '0'-'G'
RC ;GREATER THAN "F"
ADI 6 ;PROCESS "A" - "F"
JRLC BIND,RP
ADI 7
RC ;":" THROUGH "@"
BIND: ADI 10 ;ADJUST
ORA A ;CLEAR CARRY
RET

* EXIT TO DDT
EXIT: LRPR H,USRSTAK ;GET USER'S STACK
SPHL ;SET IT
POP PSW!POP B!POP D!POP H ;RESTORE REGISTERS

```

RST 7 ;RETURN TO DDT

\* DATA AREA

PROMPT DB 13,10,'S:\$'  
USRSTAK DW 0  
INBUF DB 20,0  
DS 20+32  
STACK DB 0  
  
END START



**"C/80... the best software buy in America!"** —MICROSYSTEMS

Other technically respected publications like *Byte* and *Dr. Dobbs's* have similar praise for **The Software Toolworks' \$49.95** full featured 'C' compiler for CP/M® and HDOS with:

- I/O redirection
- command line expansion
- execution trace and profile
- initializers
- Macro-80 compatibility
- ROMable code
- and much more!

**"We bought and evaluated over \$1500 worth of 'C' compilers... C/80 is the one we use."**

—Dr. Bruce E. Wampler  
Aspen Software  
author of "Grammatik"

The optional **C/80 MATHPAK** adds 32-bit floats and longs to the C/80 3.0 compiler. Includes I/O and transcendental function library all for only **\$29.95!**

C/80 is only one of 41 great programs each **under sixty bucks**. Includes: LISP, Ratfor, assemblers and over 30 other CP/M® and MSDOS programs.

For your **free** catalog contact:

*The Software Toolworks'*  
15233 Ventura Blvd., Suite 1118,  
Sherman Oaks, CA 91403 or call 818/986-4885 today!

CP/M is a registered trademark of Digital Research.

## If you are reading a borrowed copy of REMark...

maybe now is the time to join the National Heath/Zenith Users' Group. You will receive:

- a copy of **REMark** filled with new and exciting articles and programs each month
- access to the **HUG** library filled with a large variety of programs
- discounts on a variety of Heath/Zenith computer products (see **REMark** January, 1984 issue for more details)

And remember, your local HUG is an excellent source of information, support and comradery. A membership package from the National Heath/Zenith Users' Group contains a list of current local HUG clubs as well as other interesting information.



## (LISP) for H89

Artificial Intelligence Language  
UO-LISP Programming Environment  
The Powerful Implementation of LISP  
for MICRO COMPUTERS

Excellent for developing A.I., ROBOTIC, EXPERT  
and INTELLIGENT SYSTEMS



The usual LISP Interpreter Functions  
Data Types, Structure Editor,  
Screen Editor with Mouse Support,  
Optimizing LISP Source Code  
Compiler & Assembler, Assembly &  
LISP Code Intermixing, Compiled  
Code Library Loader, Numerous Utility  
Packages, Hardware and Operating  
System Access, Session Freeze and  
Restart, Comprehensive 350 page  
Manual with Usage Examples, and  
much more is available with the  
ADVANCED INTERPRETER, your  
environment may cause constraints,  
review catalog.

Other UO-LISP products include META a translator writing system,  
RLISP a high level language, and LISPTX a text formatter.

**!!NEW!!** LEARN LISP FOR \$39.95

The "LEARN LISP" Interpreter includes a subset of the above  
description, manual includes tutorial.

The UO-LISP Programming Environment runs on the H89 with CPM.

Package Prices range from \$39.95 to \$300.00.

Manuals may be purchased separately.

TO ORDER: First send for your FREE catalog which lists technical details,  
distribution formats, and ORDER FORMS.

VISA and Mastercard accepted.

**Northwest Computer Algorithms**

P.O. Box 90995, Long Beach, CA 90809 (213) 426-1893

## FLOPPY DISK CONTROLLER

Controls Any Combination Of Up To Four  
8" and 5 1/4" Drives

This easy to install plug in board can control any  
combination of single or double sided, single or  
double density drives.

*Designed especially for H88/H89 users.*

- Fully compatible Bios supplied for your CP/M 2.2 operating system
- Easy to follow instructions
- Contains controller board with boot prom
- Order cables for connection \$15 (HFDC-110)
- **Introductory Offer \$395,**  
Order HFDC-100

**NORTH  
COAST  
INTELLIGENCE  
INC.**

1201 Cherokee Trail  
Willoughby, Ohio 44094  
Phone: 216-946-7758

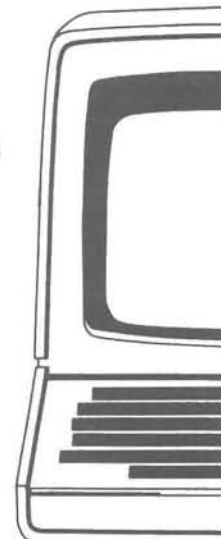
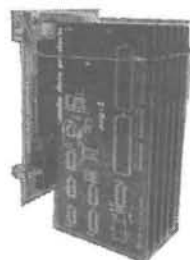
Check, COD, VISA or MC — 90 Day Warranty

## FOR YOUR LITTLE WIZARDS

**Little Wizards** is a new, menu-driven, 7-program educational package for children. The package includes Number Skills, Reading Readiness and Visual Discrimination programs. Your child will learn upper and lower case letters, numbers from 1 to 30, how to spell and recognize eight nouns, and he or she will be introduced to over 100 words in lower case letters. Each program of the set is full of new, delightful, stimulating graphics for motivated "fun" learning. **Little Wizards** is available for HDO5, CP/M and CP/M-85 with MBASIC on hard or soft sectored 5 and 1/4 inch distribution disks. 56K memory is required. (48K version available.) Can be ordered with or without color. Suggested prerequisites: "Preschool Challenge" and "Letters and Numbers". For more information on this package and others, write or call (313) 651-3859 or Send \$29.95 to:

**INTUITIVE LOGIC**

412 Taylor Street  
Rochester, MI 48063-4327



## H89/Z90's CAN NOW DEAL WITH A FULL DECK!

THE ORIGINAL ALL-IN-ONE  
ACCESSORY BUS EXPANDER.

MH89+3 doubles expansion capacity. Allow for 6 right-hand type cards instead of the usual 3. Room at last to run those neat accessory boards you've seen advertised!

Piggyback motherboard installs internally with a screwdriver in just minutes — with no modifications! 3 slots exactly duplicate the originals. The 3 added slots occupy unused addresses and eliminate previous conflicts. 100% compatible with all accessory boards!

No overheating problems! Simple design draws little power. Leaves plenty of overhead for the minimal load of most accessories. Full technical information provided.

The best news about this "No-hassle" design is the price — **ONLY \$150**. About 1/3 the price of other solutions!

Price includes assembled and tested MH89+3 expander, complete instructions and one (1) year warranty. CA residents add 6% tax. USA include \$5 shipping. Foreign add \$10. Telephone and COD orders accepted.

**mako data products**

1441-B N. RED GUM, ANAHEIM, CA 92806  
PHONE (714) 632-8583

↳ Vektored from 7

### Toggling Along With WordStar

Dear HUG,

In the July, 1984 issue of REMark, Gary Deley described his set of patches for WordStar which implement some of the features of the C.Itoh 8510A printer. His principal goal seemed to be the printing of sub and superscripts, which he implemented with a combination of escape sequences requiring ten keystrokes to enter the sub or superscript mode and six to leave it.

I found a simpler way to print sub and superscripts and to use several other features of the 8510A. The superscript mode is toggled with the normal WordStar command ^T, and the subscript mode with ^V. (Three keystrokes each.) I will not repeat Deley's very clear directions on the use of DEBUG to install patches or his discussion of the various WordStar printer control commands.

Apparently, when WordStar encounters a subscript or superscript toggle it looks to see if ROLUP or ROLDOW have been installed. If they have *not*, then it uses a half line space if PSHALF has been installed. To get a superscript it moves down a half space, prints a whole line consisting of blanks everywhere except for the superscript, and then goes down another half space to print the remainder of the line, substituting blanks for the superscripts. Subscripts are handled in an analogous way. Thus, you can get super and subscripts by installing *only* PSHALF. The major disadvantage is that the superscripts are a half line up, so they overlap the preceding line in single-spaced text. Another disadvantage comes from the fact that code 20H in the 8510's Greek font is not a space but an  $\sigma$ . You will get occasional unwanted  $\alpha$ 's if you use Greek characters in sub or superscripts, and the only sure-fire remedy I have found is white paint!

I found that I prefer the looks of the Elite font on the 8510, and using it

I have ample room for margins, so I installed it as the default, using PSINIT. This routine also resets various other options and sets the left margin to the first column, since WordStar takes care of margins by itself. The four user defined functions were then available for font changes. ^Q switches to the Greek font, ^W to the Pica, ^E back to Elite, and ^R issues an ESC so that other printer commands may be constructed (at the cost of messing around with the line length, since the remainder of the ESC command will not print but will be counted by WordStar as part of the line). Elongated characters are called forth with ^A and terminated with ^N.

Finally, I found that the underscore character on the 8510A is placed in line with the bottom edge of the capital letters. Thus, if you use WordStar's underscore feature, which overprints with this character, you change all F's into E's and produce a rather ugly result. So I used the ribbon toggle, ^Y to turn on and off the underscore feature of the printer. (On the later model 8510S, the underscore character is moved down so that it looks all right, and I used the ^Y to switch in and out of italics, which are provided on that model of the printer.)

The following patches implement these changes, which work on both the Z-100 and the Z-150:

075E	PSHALF	09 1B 54 31 32	0D 0A 1B 41	Half-line space	
076B	PALT	01 0E		Elongated characters	(^A)
0770	PSTD	01 0F		Standard-width characters	(^N)
077F	USR1	02 1B 26		Greek font	(^Q)
0784	USR2	04 1B 24 1B 4E		Pica font	(^W)
0789	USR3	04 1B 24 1B 45		Elite font	(^E)
078E	USR4	01 1B		ESC	(^R)
0793	RIBBON	02 1B 58		Underscore on	(^Y)
0798	RIBOFF	02 1B 59		Underscore off	(^Y)
079D	PSINIT	0B 0D 1B 45 1B 24 0F 1B 4C 30 30 30		Reset	

Alternate for 8510S

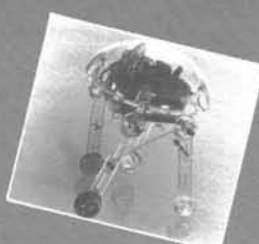
0793	RIBBON	03 1B 69 31	Italics on	(^Y)
0798	RIBOFF	03 1B 69 30	Italics off	(^Y)



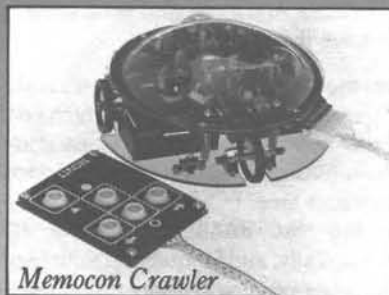
Peppy



Circular



Medusa



Memocon Crawler

# MOVIT

Educational Electronic Controlled  
Robot Kits From OWI.

## NINE WAYS TO MOVIT.

Use your mind and a few tools and enjoy the educational and pure fun benefits of MOVIT.

Only the choice is difficult... Sound Sensor Controlled models which include Peppy, Piper Mouse, Medusa and Turnbacker... Infra-Red Sensor Controlled kits like the AVOIDER or Line Tracer... Circular, a Radio Frequency Controlled kit... Mr. Bootsman, our Hand Controlled kit or... the incredible 4K Ram Programmable Memocon Crawler.

See your local Heathkit dealer. MOVIT kits range from \$24.95 to \$74.95 suggested list price.



OWI Inc. 1160 Mahalo Place  
Compton, CA 90220 (213) 638-4732

It may be useful for the new owner of an 8510 with a parallel interface to know that he should set SW1-7 CLOSED to avoid problems with line spacing using certain of the ZDS system programs which send LF CR rather than CR LF.

Alan T. Moffet  
4848 Glenalbyn Drive  
Los Angeles, CA 90065

### Still Bugged With HT

Dear HUG:

L. T. Scotney (April 84) and Glenn F. Roberts (July 84) both "Bugged HUG" about the horizontal tab (HT) problem when using the H/Z-100 to send graphics to a printer. Here we give a solution for parallel printers.

Parallel printing is controlled by the MC68A21 Parallel Interface Adapter on the main circuit board. This chip is described in the Hardware Manual in the appendix, D.56. The situation is somewhat complicated, since the same chip is also used to interface the light pen and the vertical sync signal from the video board as described in the Hardware Manual, 2.17. The trick is to operate the parallel port without wrecking the vertical sync signal.

A listing of the Bios file, BCHRIO.ASM contains the routine CHRFOOD—PAR which controls the MC68A21 during parallel printing. This routine contains clear remarks describing each step. The same routine could be implemented in languages other than Assembler.

We are attaching a listing of a BASIC subroutine that executes the routine. Although we do not disable interrupts as does the Assembler routine, this BASIC version does faithfully transmit all ASCII codes to the printer. The routine was tested by printing 0 to 255 on an Epson FX80 operated in the Hex Dump mode.

Sincerely,  
Don Vickers  
Box 1, Parnassus Station  
New Kensington, PA 15068

```
10000 'SUBROUTINE CONTROLS MC68A21 PIA
10010 '
10020 'December 7, 1983
10030 '
10040 'BY Don Vickers
10050 'Box 51, Parnassus Station
10060 'New Kensington, PA 15068
10070 '(412) 339-7553
10080 '
10090 '
10100 'The HZ-100 computer evaluates
      HORIZONTAL TAB [CHR$(9)]
10110 'internally and outputs appropriate
      number of spaces
10120 '[CHR$(32)]. Therefore CHR$(9) must be
      intercepted and
10130 'output directly to the Centronix parallel
      interface. This
10140 'is a BASIC subroutine that will output
      directly to the
10150 'Centronix port.
10160 '
10170 'A = ASCII VALUE OF CHARACTER TO BE PRINTED.
10180 '
10190 STAT = INP(&HE2) AND 3
10200 'status = value of two low-order bits of port B.
10210 IF STAT < 2 THEN PRINT "PRINTER ERROR!":END
10220 'because error bit is low.
```

```
10230 IF (INP(&HE2) AND 3) > 2 THEN PRINT "PRINTER BUSY"
      :GOTO 10230 'because busy bit is high.
10240 '
10250 '
10260 OUT &HE2,A 'output full byte to B.
      The two low-order bits
10270 'won't matter since PIA control word sets
      them for input.
10280 '
10290 TEMP = &HAC 'This is the bit pattern to adjust the
10300 'high-order six bits for values to maintain
      operation of
10310 'VERTICAL RETRACE and LIGHT PEN. The two low
      order bits
10320 'are set to zero.
10330 'ATEMP = A AND 3 'Mask out all but two low
      order bits of
10340 'byte.
10350 AY = TEMP OR ATEMP 'Now tack two low order
      bits onto high-
10360 'order 6 bits of A port.
10370 '
10380 OUT &HE0,AY 'Output composite byte to port A.
10390 '
10400 STROBE = &HFB AND AY 'Drive bit 3 of
      composite byte low.
10410 '
10420 OUT &HE0,STROBE 'Now output so as to
      strobe printer.
10430 '
10440 NORM = STROBE OR 4 'Set bit three high.
10450 '
10460 OUT &HE0,NORM 'Output to reset printer strobe.
10470 '
10480 RETURN
```

### Unsolicited Final Word!

Dear HUG,

Recently, you have run some letters on the virtues and vices of various word processors for the Z-100. I haven't ever written an "unsolicited testimonial," but I have seen so much junk that passes for wordprocessing software, that I'd like to put in a good word for *Final Word*. I have been using it for about three months now, and it's the closest thing to satisfying my word processing needs that I have ever seen. First, it seems to have been designed to fill a multitude of needs, not just to function as a business-oriented word processor. I bought it because it automatically formats numerous footnotes, but I find that it has numerous other capabilities. By sacrificing the "what-you-see-is-what-you-get" philosophy of word processing, it permits much greater flexibility of formatting. Readers familiar with "runoff" and similar programs, which permit embedded formatting commands, will know how this capability can make word processing tasks easier. And those who want to stick with on-screen formatting can have that, too.

It permits the construction of templates and forms of various kinds, and of "personalized" form letters which enable you to slightly alter a form letter by using variables and case statements. It can automatically chain files together, and accept console input to the printer during printout time. In addition, it supports numerous printers including the NEC 8023, the Diablo, IDS, Okidata, Qume, Mannesman-Tally, and Epson (most with two or three different fonts, including superscripting and true proportional spacing). Or you can fully define your own printer using a comprehensive configuration program. There is also a terminal configuration program that permits you to specify color and other attributes (cursor, for instance) on your Z-100, as well as a program to permit you to map your function keys on the Z-100. In addition, there are windows, numerous buffers, highlighting, undelete, and so forth. And any feature which might be



irritating (automatic saves, for instance) can be disabled. Aside from the infinite line length permitted by Watchword, I have never seen a feature advertised for another program that was not easily achievable with *Final Word*.

Its documentation is excellent, and when I called the help number listed in it, the response was prompt and effective. Of course, complexity has its costs, and this program's drawbacks are that its control sequences are three, instead of the normal two, keystrokes, and the simple problem of trying to remember how to run such a complicated program. But these problems are mostly overcome by the key-definition feature. Another problem is puzzling -- the Z-100 function keys 9 through 12, when shifted, produce a two-character control code, and *Final Word* only accepts one-character codes, so shifted function keys 9-12 do not work. (But *Final Word* has ways of overcoming this too, by making use of a second "escape" character which can remap any key on the board.) One personal problem I have encountered is that it's so tempting to tinker with its configuration in order to fully "customize" the program, that I have spent too much time playing with the program and not enough time writing. I continue to find nice features that I might find a use for someday.

On another matter, I recently ordered HUG's Keymap program and it's a real bargain, but I have been unable to get the "alternate character" feature to work or to get shifted f12 to work. Is it a bug in the program or my machine or me?

Yours truly,

Joe Riehl

The Round Table

South Central College English Assoc.

USL Box 40022

Lafayette, LA 70504

## Update on ZDOS "CRTSAVER"

In the July/84 REMark (Volume 5, Issue 7), Frank Clark wrote an article called "Advanced Assembly Language Programming." This article included an assembly source program called "CRTSAVER" for ZDOS. This program, as written, has a couple of flaws. The major flaw exists with interrupts at program initialization which can cause system crashes. Interrupts are not locked out between the setting up of the programs interrupt vectors and the setting of the addresses for the old (chained) interrupt. If one of these interrupts happen just before the chained address is stored, then it will be vectored out to nowhere! System crash or hang!

To remedy this, I suggest the following changes. Insert an "STI" instruction just after each of these three instructions:


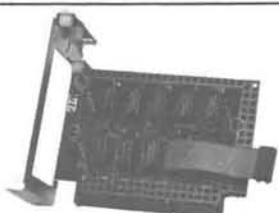
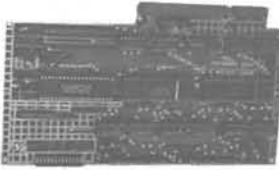

```
MOV  NXT_CRT+2, ES
MOV  NXT_KB+2, ES
MOV  NXT_TM+2, ES
```

Then remove the "STI" instruction toward the end of the program (just before the "POP DS" instruction). This will keep interrupts locked out until after the chained interrupt addresses have been stored.

Another, less fatal problem exists in the timer countdown routine. the "AX" register is used for decrementing the timer. At this point, the contents of the "AX" register is undeterminate. The instruction after the tag "INT-TM:" should have read:

```
SUB  CS:TIMER.1 ; Countdown
```

The constant of one (1) will cause a timeout of no keyboard or crt activity for a little over ten minutes and blank the screen. This

		<h2>HEATH/ZENITH 88, 89, 90 PERIPHERALS</h2>	
<h3>16K RAM EXPANSION CARD</h3> <p>Expand your H/Z 88, 89 RAM Memory to a FULL 64K and begin using larger and more powerful programs with our 16K RAM card.</p> <p><b>Fully compatible with:</b> Magnolia Microsystems and CDR CP/M and disk I/O interface cards.</p> <p><b>Featuring:</b> Complete installation instructions • Mounting bracket • 90 day warranty Field reliability record exceeding 3 years</p> <p style="text-align: center;"><b>Only \$65.00</b> Shipping &amp; Handling \$5.00</p>			
<h3>PORT SERIAL CARD</h3> <p>2 I/O 3 PORT PARALLEL</p> <p>"... not your typical vanilla-flavored serial and parallel interface..."</p> <p>Your H/Z 88, 89, 90 can now directly connect and operate EPSON, IDS, ANADEX, GEMINI, SILVER REED, NEC, SUPER 5, PROWRITER, OKIDATA, and many more line printers using CENTRONICS style parallel interface with our 2/3rds, 2 port serial, 3 port parallel interface card. Or you may use all 24 digital lines in various configurations of input, output, or bidirectional modes for industrial control or data sampling.</p> <p><b>Features:</b></p> <ul style="list-style-type: none"> <li>• 2 Serial Ports Supporting Ring Input, and External Clock</li> <li>• 3 Parallel Ports, 24 Total Digital Input/Output Lines</li> <li>• Fully Compatible with All Models of H/Z 88, 89, 90 using Heath/Zenith CP/M or HDOS</li> <li>• Now Supporting CP/M Version 2.2.04</li> <li>• Choice of Centronics Line Printer Support Software for the CP/M or HDOS Operating Systems</li> <li>• Reduced Computer Bus Loading and Chip Independent Design</li> </ul> <p>Complete with Installation Instructions, Documentation, 90 Day Warranty, Two Serial Cables and a Parallel Cable Internal to the Computer.</p> <p><b>Price \$199.00</b></p>			
<p style="text-align: right;"><b>Second Operating System Driver \$25.00</b></p> <p style="text-align: center;">Shipping &amp; Handling \$10.00</p>		<h3>REAL TIME CLOCK</h3> <p>You will be able to perform time and date stamping for point of sales software, and bulletin board software or perform time studies as well as real time data sampling with our REAL TIME CLOCK. This peripheral card is a perfect companion to our 2/3rds card for industrial control and data sampling. STOP WATCH time study and alarm demonstration software is included for either the CP/M or HDOS operating systems. You will be able to view the current date and time on screen continuously or simply listen to an audible beep every fifteen minutes and the hour chimed or disable the clock entirely at your option.</p> <p><b>Features:</b></p> <ul style="list-style-type: none"> <li>• True I/O Addressing, Not Memory Mapped</li> <li>• User Selectable Address</li> <li>• Rechargeable Battery Backup Using Commonly Available Batteries</li> <li>• Installable on the Left or Right Side of the Computer (Left side operation requires our I/O expansion module.)</li> <li>• Month, Day, Year, Hours, Minutes, Seconds, 1/10's, 1/100's and 1/1000's of Second Accuracy</li> <li>• Interrupt Capability Based on Tenths of Seconds, Seconds, Minutes, Hour, Day, Week or a Specific Date and Time</li> <li>• Choice of CP/M or HDOS Operating System Software Driver and Demonstration Programs</li> </ul> <p><b>Price \$105.00 with Batteries</b>      <b>\$89.00 without Batteries</b>  <b>\$ 25.00 Software for Second Operating System</b></p> <p style="text-align: center;">Shipping &amp; Handling \$5.00</p>	
<p>HDOS is a reg. trademark of the Heath Co.      CP/M is a reg. trademark of Digital Research          PRICES ARE LESS SHIPPING AND TAX IF RESIDENT OF CALIFORNIA          MAIL ORDER: 12011 ACLARE ST., CERRITOS, CA 90701 (213) 924-6741          TECHNICAL INFO/HELP: 8575 KNOTT AVE., SUITE D, BUENA PARK, CA 90620 (714) 952-3930</p> <p style="text-align: center;">TERMS AND SPECIFICATIONS SUBJECT TO CHANGE WITHOUT NOTICE — VISA AND MASTER CARD GLADLY ACCEPTED</p>			
			

constant can be increased if a shorter timeout is desired (i.e.: "5" will take about 5 minutes, and "10" will take about one minute). I prefer the 5 minute timeout myself.

After making these alterations to the program, I have had no further problems. I have included the program in my "AUTOEXEC.BAT" file so the CRTSAVER is loaded at boot time. Thanks go to Frank for this very useful routine. This program will, without doubt, extend the life of my CRT! It could yours, too.

Jesse R Luckett  
SPERRY Corporation  
1704 Longacre Terrace  
Indianapolis, IN 46227

Reply from Frank Clark

The suggestion concerning the possible problem with interrupts appears valid. It is interesting to note that after a year of constant use and several months of testing by over a dozen people that the problem has never been known to occur. I would agree that it is probably a good idea.

The second suggestion concerning the use of a constant instead of the AX register is based on a false assumption. As a matter of fact the AX register is initialized by the BIOS with the number of hundredths of a second since the last time the software interrupt was generated. (I apologize for not mentioning this fact in my article.) Remember that these are software interrupts not hardware interrupts. Using a constant will cause considerable fluctuation in the actual amount of delay, since the timer software interrupt will not always occur regularly. This is particularly true when the hardware interrupts which lead to the software interrupt are disabled for relatively long periods

of time as they are when the floppy is used heavily.

I appreciate the enthusiasm concerning this program. Thank you for your comments.

### Expresses Righteous Indignation..

Dear HUG;

Re: Inside Microsoft Basic, D. D. Dodgen; REM Vol. 5, N 8

I am writing this letter in righteous indignation over one of the concepts emphasized in this article on MBASIC. While he did an excellent job in presenting the method of internal storage of the tokens used in MBASIC, his emphasis on the minimizing the storage requirements is one hundred and eighty degrees out of phase with what one wants to emphasize in the teaching of coding in the BASIC language today. Compression of data was a very important concept in the days of "Steam Powered" computers. When 4K was considered a lot of RAM, you did have to worry about what you were storing. With the reduction in cost of storage, both in RAM and in secondary storage, emphasis should be first place on good programming style and readability by people rather than storage minimization. The emphasis on assigning more than one statement to a line number can create many problems.

First, let us look at the advantage of assigning more than one statement to a single line number:

1. It saves a minor amount (5 bytes) of storage per line number.
2. It presents more source statements to a page (screen or paper).

The disadvantage of multiple statement lines are:

1. Multiple statement lines are much less readable to human beings than single statement lines. For the computer to read a program all that is needed is that the source program obey the rules of syntax (there is a tax on everything) for the particular language being used. It is much more difficult for the human to separate the concepts used in each statement when they are strung out on a common line. Mr. Dodgen actually places each statement on a different line of the screen in his printout published with the article, but the form used does not work on the version of MBASIC that I have (4.82). If I do not include the "@" symbol when I wish to continue a logical line on the next physical line, MBASIC returns to the direct or command mode. I also do not know how much space the "@" and the additional spaces will take in RAM. Emphasis should be placed in readability in order to save time in debugging a program and in interpreting the program (by a human) at a later date. Look at the listing of the program "GRAPH.BAS" on Page 64 of the same issue. See how much easier it is to read. Programming, coding, and debugging time are much greater today than the cost of storage.
2. The use of multiple statements in a line make interpretation (by humans) of the error statements much more difficult. In indirect mode, the error statement makes a reference to the line number. It is often difficult enough to try and figure out what an error statement meant when there is only one statement per line. It is even more difficult to determine which statement it refers to in a multistatement line.
3. The same comment can be made when using the TRACE (TRON - TROFF) routines. The Trace routine only prints out the line number as the program is run -- not the statement number.
4. Even with MBASIC's edit routine it is much more difficult to modify a statement in a multistatement line than in a single statement line. There is also the chance of accidentally modifying the wrong statement in the line.
5. Similarly, it is more difficult to add a statement to a program.

The Software Toolworks presents:  
**MYCALC™**

Full featured MyCalc with sort, bar graphs, multiple files and more is ideal for financial planning and budgets. It sells for \$59.95 from The Software Toolworks, 15233 Ventura Blvd., Suite 1118, Sherman Oaks, CA 91403 (818) 986-4885.

- 6. It is also more difficult to delete a statement from a program.
- 7. Much more care must be taken in providing handling points for GISUB, GOTO, ERR, and ERL procedures.

Mr. Dodgen also emphasizes the minimization of the ASCII statements used in prompts and in REM statements. The emphasis on the people readable items in a program should be clarity and not the saving of disc or RAM space. It makes little sense to save the cost of a \$3.00 diskette when you later cannot remember what you meant by a prompt or comment. Also, the deleting of REMs before the storage on permanent medium is poor practice. The remarks are mainly there to help the user interpret the program at some later date. With BASIC's limited variable naming ability makes a glossary of variable names at the start of a program a very useful item. The inclusion of nul remarks (REM with no following text) to separate the blocks in a BASIC program will also make the program more readable. The storage of the complete program, including the comments, on the permanent medium is most important for later study both by the original programmer and the future user.

Again to close, programming, coding, documenting, debugging, and updating costs (commercially or in terms of your hobby time) are much greater than the cost of storage. Making a program as readable to humans as it is to the computer should be a major goal of all involved with the computing process.

Kenneth Mortimer PE  
352 Green Acres Drive  
Valparaiso, IN 46383

**GEMINI Options Exposed!**

Dear HUG,

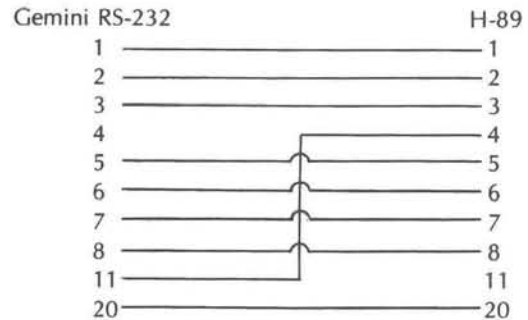
I Recently acquired a GEMINI 10X printer, and wished to run it with my H-89 with HDOS out of the RS-232 port.

GEMINI Options: (W/Serial busy)

DIP Sw	Jumper	Conn
1-off	S-1	A-C
2-off	S-2	A-C
3-off	S-3	A-C
4-off	S-4	A-C

5-off	S-5	A-C
6-on (1200 Baud)	S-6	A-C
7-off (1200 Baud)	S-7	A-C
8-off (1200 Baud)	S-8	A-C

A modified cable is needed to connect the Gemini serial port to the "LP" port (340Q) of the H-89:



H-89 setup:

1. ref CHAP 1, HDOS SYSTEM CONFIGURATION -- Step 8 -- Page 1-34, 35 configure user disk with LPH.DVD
2. ref CHAP 2, HDOS GENERAL OPERATION -- Page 2-45 to establish input/output (I/O) configuration utilize SET command:  
>SET LP:HELP(CR) provides a menu similar to those on table F, G, or H.  
>SET LP:BAUD 1200(CR) programs the H-89 to operate at 1200.  
>SET LP:LENGTH 60(CR) programs the H-89 for a page length of 60 lines per page.  
>SET LP:PORT 340Q(CR) programs the H-89 to output copy to port 340Q.
3. ref CHAP 6, BASIC, A SIMPLE TEST PROGRAM -- Page 6-78 -- "Using a line printer with Basic" has helpful information.

This information will help to get the printer to copy, and a lot will depend on the program (software) prompts to utilize many of the Gemini's features.

Sincerely,

Charles W. Wilson  
W. 3819 Weile Ave  
Spokane, WA 99208-4845

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



CUT ALONG THIS LINE

# HUG MEMBERSHIP RENEWAL FORM

HUG ID Number: \_\_\_\_\_

Check your ID card for your expiration date.

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

Name \_\_\_\_\_

Address \_\_\_\_\_

City-State \_\_\_\_\_

Zip \_\_\_\_\_

**REMEMBER - ENCLOSE CHECK OR MONEY ORDER**  
**CHECK THE APPROPRIATE BOX AND RETURN TO HUG**

	NEW MEMBERSHIP RATES	RENEWAL RATES
US DOMESTIC	\$20 <input type="checkbox"/>	\$17 <input type="checkbox"/>
CANADA	\$22 <input type="checkbox"/>	\$19 <input type="checkbox"/> US FUNDS
INTERNAT'L*	\$30 <input type="checkbox"/>	\$24 <input type="checkbox"/> US FUNDS

\* Membership in France and Belgium is acquired through the local distributor at the prevailing rate.

You're invited to the  
***Eastern Regional HUG Conference***  
***"CHUGCON 84" November 3, 1984***

Free Admission  
 Displays Workshops Swapmeet Prizes  
 Nationally Recognized Speakers

**Grand Prize: HS-161 Portable Computer\***

***Crystal City Hyatt Regency***  
***Arlington, Virginia near Washington National Airport***

Write:  
 CHUGCON 84, P. O. Box 10515, Alexandria, VA 22310  
 \* You must be present at dinner to win.

***Index of Advertisers***

Analytical Products .....	7	New Orleans General .....	40
Apogee Software .....	42	North Coast Intelligence Inc. ....	62
Controlled Data Systems, Inc. ....	42,55	Northwest Computer Algorithms .....	62
DelSoft .....	43	OWI .....	63
Generic Software .....	10	Paul F. Herman .....	19
Headware .....	43	Secured Computer Systems .....	65
Intuitive Logic .....	62	Software Support .....	44
MPI .....	20	Software Toolworks .....	61,66
Mako Data Products .....	62	Software Wizardry .....	2
Microservices .....	55	Technical Micro Systems, Inc. ....	52
Mockingbird Data Systems .....	24	Veritechnology .....	8,56
NewLine Software .....	4		



Hilltop Road  
 Saint Joseph, Michigan 49085

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

Volume 5, Issue 10

POSTMASTER: If undeliverable,  
 please do not return.

P/N 885-2057