$2.50

REMark®

Issue 41 • June 1983

WATZMAN ROMs NOW AVAILABLE

see page 27 for details

Official magazine for users of HEATH ZENITH computer equipment.

Heath Users' Group

HUG Manager	Bob Ellerton
Software Engineer	Pat Swayne
HUG Bulletin Board and Software Developer	Terry Jensen
Software Coordinator	Nancy Strunk
HUG Secretary	Margaret Bacon
REMark Editor	Walt Gillespie
Assistant Editor	Donna Melland
Printers	Imperial Printing St. Joseph, MI

REMark is a HUG membership magazine published 12 times yearly. A subscription cannot be purchased sparately without membership. The following rates apply.

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

*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 availble 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 Users' Group
Hilltop Road
St. Joseph, MI 49085

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®
Issue 41 • June 1983

on the stack

ON THE COVER: Pictured is the new Watzman ROM set. See page 27 for information. (Photo by Jon Falkner)

THE H-100 SERIES: THE WORLD'S FIRST 16-BIT COMPUTER KITS.

with authorized sales and service *only* through the Heath Company and Heathkit® Electronic Centers.*

The world-famous H-100 Series Heath/Zenith computers are the first to give you advanced 16-bit computing at a kit price. Many who have never considered kitbuilding are now interested because most circuit boards are prewired, making the H-100 Series our simplest computer kits. Dual microprocessors deliver 16-bit speed and 8-bit compatibility. The industry standard S-100 card slots allow a host of peripherals and memory expansion to 768K RAM. And our stores and catalogs have the software to help you take full advantage of the faster 16-bit operating speed.

When you're ready to move up to H-100 Series, remember: these computers are available *only* through the companies that give you solid service and a trustworthy warranty...Heath and Heathkit® Electronic Centers.* The companies that stand by their promise of support to kitbuilders. The companies with the pledge: *"We won't let you fail."*

Many companies would like to sell our product but no other dealer or vendor can resell the H-100 Series without voiding the original factory warranty. When buying the world's first 16-bit computer kit, buy only from the world leader in electronic kits.

Authorized sales and service available only through the Heath Company and your...

Heathkit® ELECTRONIC CENTER*

*Heathkit Electronic Centers are units of Veritechnology Electronics Corporation.

VEC-770

Business as Usual??

During the past two years, the microcomputer has been advertised, promoted, and sold as the answer to the business communities' problems, i.e., accounting, inventory, forecasting, and special needs. The microcomputer is very capable, and actually ideal, for doing these repetitious tasks. However, to the business manager trying to utilize a computer system I say, "Yes, but ....".

YES, the computer is capable of being used as a productive tool in the business environment. BUT wait ... each business has unique problems different from the next. One business may benefit from a pre-written software package. Another business may need to have a special program written to do a particular job. Careful consideration of all software options must be taken by a manager when deciding which, if any, software package should be purchased. The greatest benefit of the computer may not come from "canned" software or "custom" software, but from software "tools" such as spreadsheets and data bases.

Pre-written, pre-tested software packages (e.g. accounting, inventory, etc.), which are becoming more numerous, are programs that have been created, based upon certain design specifications. The packages are written in a general format so as to be useable by as many types of businesses as possible. The packages may work well and, in themselves, be excellent programs which contain no major flaws or bugs. However, if it doesn't produce an accurate account of a particular need, then the package may not be of practical use to that business. Also, the manager must determine if the software package is affordable. Good software packages do not come "cheap", and rightfully so!

If cost is not important, the ideal software package is one that is created and written specifically for a business. The programmer can work directly with the manager and the personnel who will actually be using the package. The programs can then be custom designed to do each of the tasks requested by the user (provided it is within the capability of the programming language).

For a manager who cannot find an appropriate software package, or for a small business that cannot afford the luxury of custom designed or pre-written software, the computer can still be a useful device. Many packages are available that provide "tools" to aid a manager in maintaining an accurate account of the needs of a business. The manager or user will be required to study and learn the basics of the program. After continued practice, the manager will become familiar with the options of the "tool" which will best benefit the business.

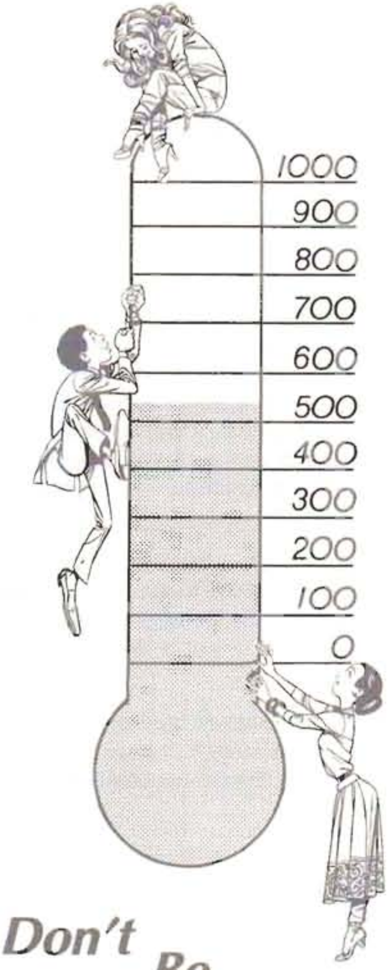
Based on this criteria, a wise manager will research the software packages that are available to find if one is compatible for his/her business. After all, nobody knows a business better than the manager. This may require extensive study by the manager, or it may require the footwork of another to determine the best possible system for the business.

If there is enough interest in this area in the coming months, I may, in a future issue, write an article on the mysteries of business software. Why is this type of software hard to find ... or is it? Why is it avoided by programmers? What types of problems are there in writing business software? What makes it so expensive? What should a manager look for when reviewing software?

A successful business is created on a careful, solid study of the marketability, accessibility, and profitability of a particular product or service. The decision to buy a computer system and workable software for a business should not be taken lightly either. It requires the same care and research. If this is done, the computer system can offer the business community a useful and viable tool to solve some, not necessarily all, of its problems.

Terry L. Jensen
Software Developer

The 1983 HUG National Conference Is Filling Up



Don't Be The One Left Out, Send Your Registration Form In Now!!

Heath Users' Group

NATIONAL HUG CONFERENCE II

Official Conference Registration Form

O'Hare Hyatt Regency Hotel, Chicago, Illinois

August 19, 20 and 21

Name(s) ______________________________

Address ______________________________

Company ______________________________

City ____________________ State__________ Zip__________

Enclosed is $20.00 per individual to attend The Second National HUG Conference to be held the weekend of August 19, 20 and 21 1983. Please send ticket(s) and information regarding hotel reservations.

AMOUNT ENCLOSED__________ NUMBER ATTENDING______

For our information:

Which Heath/Zenith computer do you now operate? ______________

Are you a Non-User-Attendee? □ YES □ NO

Are you a Heath/Zenith related vendor? □ YES □ NO

For your information:

Space limitations for the dinner to be held Saturday August 20, 1983, will restrict the number of attendees for that dinner to 1000. Therefore, it is important that you register as soon as possible. Visitor tickets for those of you simply attending and not planning to stay for the dinner and prize drawings will be available at the registration booth for $10.00. Send your registration form or a suitable copy to:

Heath Users' Group
Attention: National HUG Conference Registration
Hilltop Road
Saint Joseph, Michigan 49085

Special Note to Vendors:

Vendor Information Packages will be made available to Heath/Zenith Related Vendors who are planning to exhibit their products while at the conference. Three times more space is available this year for the purpose of showing those products of interest to owners of Heath/Zenith computer products.

BUGGIN' HUG

Screen Control

Dear Walt,

David Warnick was right; his article on screen control was sure to elicit letters. Here's mine.

You have published several interesting programs in recent months that use various forms of screen control similar to those advocated by Mr. Warnick. As a recent purchaser of an H-89, I have learned a lot from those programs, particularly Jennie McGraw's XMASPROG and Bob McFarland's CheapCalc. Both use graphics control sequences that are better than Mr. Warnick's. In this letter, I would like to suggest further refinements and some systematic development of control sequences.

First, CheapCalc shows the neat and easy way to position the cursor on the screen: FN PC$(R%, C%) is a whole lot easier to program with than scouring through the graphic symbols chart to find the character with the ASCII number 31 higher than the needed row or column.

Second, Warnick's string designations for his various escape sequences are unrelated to the designated action, therefore difficult to remember and difficult to program with. Borrowing from McGraw and McFarland, and expanding on my own, I have developed the following sequence of escape codes. I have them saved as a high-numbered program block. When I start writing a new program, I just load that block, then start the program with a GOSUB to the graphic control sequences.

As much as possible, these codes follow a pattern. The first letter usually represents the thing being acted upon (C for Cursor, S for Screen, etc.) and subsequent letters relate to the action taken (U for Up, E for Erase, etc.).

Here are the codes I use:

E$	Escape	CHR$(27)
CH$	Cursor Home	E$ + "H"
CU$	Cursor Up	E$ + "A"
CD$	Cursor Down	E$ + "B"
CL$	Cursor Left	E$ + "D"
CR$	Cursor Right	E$ + "C"
CR2$	Cursor Right Two	CR$ + CR$
CR3$	Cursor Right Three	CR2$ + CR$
	(continue through CR0$ for Cursor Right Ten)	
SCP$	Save Cursor Position	E$ + "j"
GSP$	Goto Saved Position	E$ + "k"
GSU$	Goto Saved Position, Go Up and Save	GSP$ + CU$ + SCP$
GSD$	Goto Saved & Down & Save	GSP$ + CD$ + SCP$
GSL$	Goto Saved & Left & Save	GSP$ + CL$ + SCP$
SE$	Screen Erase	E$ + "E"
LE$	Line Erase	E$ + "I"
GR$	Graphics Mode	E$ + "F"
GX$	Graphics Mode Off or Exit	E$ + "G"
RV$	Reverse Video	E$ + "p"
NV$	Normal Video	E$ + "q"
EN$	Enable Line 25	E$ + "x5"
DI$	Disable Line 25	E$ + "y5"
DCA$	Direct Cursor Addressing	E$ + "Y"
FN PC$(R,C)	Position Cursor	DCA$ + CHR$(R+31) + CHR$(C+31)
L25$	Enter Line 25	EN$ + FN PC$(25,1) + SE$ + RV$
X25$	Exit Line 25	FN PC$(25,1) + SE$ + DI$ + NV$
EXIT$	Exit Everything	GX$ + NV$ + X25$ + FN PC$(23,1)
BEEP$	Beep	CHR$(7)

Logical extensions of these codes would include KS$ for Keypad Shifted Mode, KSX$ for Exit Key Shifted Mode, etc. I would also use X to signify exit or shut off a function.

The GSU$, GSD$, and GSL$ functions are extensions of a nice touch in XMASPROG, but with more meaningful monikers than K1$. EXIT$ is also borrowed from the same source; it's very useful to restore everything to normal from the command mode while writing a program, and sometimes useful within a program.

I hope all this subtracts from the level of confusion rather than adding to it.

Is there any hope of getting HUG and REMark to agree on a standard set of control codes?

Lewis M. Phelps

Dear HUG,

I enjoyed Dave Warnick's article on "Screen Control" in Issue 39. Such articles should be periodically repeated as membership grows and changes. I was going to enter the program on page 20, column 1, but it dawned on me that there was a way to accomplish the same result with an 8 line program and I have included it for those who might be interested.

Thanks again to Dave Warnick for a great article. I look forward to his future articles because he is very good at keeping it simple for guys like me. I'm just a beginner using MBASIC.

I could use some tips on how to make my H-14 print out the same format as that on the console screen. Everything comes out "left-justified".

```
10 E$=CHR$(27)
20 E1$=E$+"p"
30 E2$=E$+"q"
40 FOR C = 1 TO 26
50      PRINT E1$;CHR$(C+64)
55      C=C+1
60      PRINT E2$;CHR$(C+64);
70 NEXT
80 END
```

Wayne K. Leikam, Sr.

Dear Hug,

I enjoyed the contents of your most recent issue (#39), including David Warnick's traversal of the H/Z-19 terminal escape codes. However, I have a very minor quibble: the names he gave to MBASIC string variables containing the escape codes. Early last summer, Frank Adams uploaded a suggested standard listing of terminal

Vectored to 50 ☞

Grocery Shopping Using SuperCalc

Jim Johnson
1413 Hillcrest Drive
Blacksburg, VA 24060

For several years, my wife and I have shopped for groceries on a monthly basis which saves both time and money. However, the preparation of an accurate and complete shopping list before we left for the store was a dreaded task. Before we purchased our Z-89, we tried printed check lists and inventory lists. After a few weeks of a new efficient way to make a grocery list, we retreated to the "as you think of it list" on a scrap of paper. As I cranked up SuperCalc, I decided that the first task I would tackle would be the grocery list.

As further background, I tend to be the type who reads the instructions only as a last resort. After less than 30 minutes of reading and general practice with SuperCalc, I was working on the first version of Grocer (my file name for the spreadsheet).

Preparing the Format

The first step was to develop and enter the list of groceries by categories such as Staples, Fruits, Cereals, etc. For example, to enter the first item, I typed "All for cell D8. My wife and I found category listing to be frustrating, so we switched to the alphabetical list. A listing by category or aisle display placement are other possibilities, but the alphabetical list was the best for us since we shop at 3 different stores. I used the "/M" feature to rearrange the list into alphabetical order. Multiplan or the soon to be released SuperCalc2 could rearrange the list using the sort feature.

We then decided on the number of each item (NEED) we would need for the month. We priced each item from what we had in the pantry or on our next trip to the grocery store. This was all the base data I needed to get started on the program.

Formulas

First let me explain the math involved using the column titles. The items ON-HAND are subtracted from the the items needed (NEED) to determine what to (BUY). (BUY) is then multiplied by (PRICE) to determine (COST). The only columns that involve math formulas are the (BUY) and the (COST) columns. From this point on, I will use the SuperCalc column headings of A, B, C, D, E, F, G, H, I, and J.

For example, the formula in cell B8 is MAX(0,F8-E8). When I first developed the list, I used F8-E8, only to end up with minus numbers. I then resorted to "faking" the program by never entering a number in column E that was larger than the number in column F. The MAX formula selects the larger of the two values, thus the number never falls below 0 because I have set it as my minimum value.

Another formula that would have worked in column B would have been IF((F8-E8)<0,0,F8-E8). I opted for the simpler method and chose the MAX version. The IF formula says that if F8-E8 is less than 0, use 0, otherwise use F8-E8.

Note: In describing the commands, I have used only the letters that have to be entered. For the command /S, the program completes the command as /SAVE,. SuperCalc enters commas between options in a command. In most cases the comma appears immediately after entering the one letter required. If a comma does not automatically appear, the command requires a RETURN before proceeding to the next option. Additional options can be called by inserting an additional comma in some commands. For example, with the /Replicate command the "from" is entered, then a RETURN is required before entering the "to" cell(s). Immediately after entering the "to" cell and before the RETURN is depressed, the entry of a comma will call options that will hold constants in a formula. The default on the /R is to adjust every replicated formula to it's new location. My method of describing such formulas throughout this article will be / R,B1,B2:B25,A.

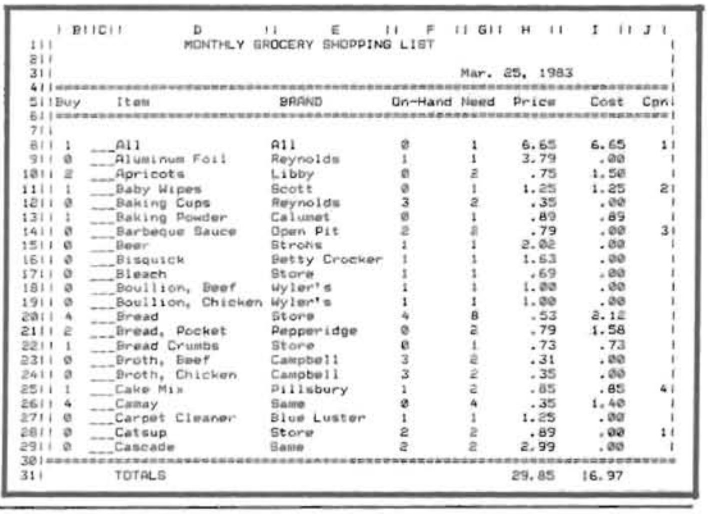
The formula in column H is B8*G8. To save retyping the formula over each time, I used the Replicate command /R,H8,H9:H128 and /R,B8,B9:B128. I protected the formulas in columns B and H using the /P,H8:H128 and /P,B8:B128 commands. The Protect command prevents accidental erasure of a formula that is fixed. I did not protect columns E, F, and G because they are subject to change as the family needs or prices change. Of course column E is entered each month.

In column A, C, and J, I used the Replicate command to draw the lines in each cell once I had entered the first cell. The purpose of column C is to provide a place to check when an item has been purchased. Columns A and J are for aesthetics.

At the bottom of columns G and H, I used SUM commands to add the columns. The commmands are SUM(G8:G185) and SUM(H8:H185). The sum of column G gives me a running monitor on grocery price increases and decreases. I only update the prices about twice a year. The sum of column H provides my wife and I a very close estimate of what we will have to spend on our grocery shopping trip. The sum is usually within 5 percent of what the actual bill is.

Setting Up the Spreadsheet

I used the Format commands to set the columns widths and decimal places. For example, the command to make column A only one



```
    | B||C||       D        ||      E       ||   F   || G||  H  ||   I   || J |
 1||                MONTHLY GROCERY SHOPPING LIST                            |
 2||                                                                         |
 3||                                                Mar. 25, 1983            |
 4||=========================================================================|
 5||Buy    Item                BRAND            On-Hand Need  Price     Cost  Cpn|
 6||=========================================================================|
 7||                                                                         |
 8|| 1   ___All               All                 0       1     6.65    6.65   1|
 9|| 0   ___Aluminum Foil     Reynolds            1       1     3.79     .00    |
10|| 2   ___Apricots          Libby               0       2      .75    1.50    |
11|| 1   ___Baby Wipes        Scott               0       1     1.25    1.25   2|
12|| 0   ___Baking Cups       Reynolds            3       2      .35     .00    |
13|| 1   ___Baking Powder     Calumet             0       1      .89     .89    |
14|| 0   ___Barbeque Sauce    Open Pit            2       2      .79     .00   3|
15|| 0   ___Beer              Strohs              1       1     2.02     .00    |
16|| 0   ___Bisquick          Betty Crocker       1       1     1.63     .00    |
17|| 0   ___Bleach            Store               1       1      .69     .00    |
18|| 0   ___Boullion, Beef    Wyler's             1       1     1.00     .00    |
19|| 0   ___Boullion, Chicken Wyler's             1       1     1.00     .00    |
20|| 4   ___Bread             Store               4       8      .53    2.12    |
21|| 2   ___Bread, Pocket     Pepperidge          0       2      .79    1.58    |
22|| 1   ___Bread Crumbs      Store               0       1      .73     .73    |
23|| 0   ___Broth, Beef       Campbell            3       2      .31     .00    |
24|| 0   ___Broth, Chicken    Campbell            3       2      .35     .00    |
25|| 1   ___Cake Mix          Pillsbury           1       2      .85     .85   4|
26|| 4   ___Camay             Same                0       4      .35    1.40    |
27|| 0   ___Carpet Cleaner    Blue Luster         1       1     1.25     .00    |
28|| 0   ___Catsup            Store               2       2      .89     .00   1|
29|| 0   ___Cascade           Same                2       2     2.99     .00    |
30|=========================================================================
31|         TOTALS                                             29.85   16.97
```

space wide is /F,C,A,1. The command to set column D at a width of 22 spaces is /F,C,D,22. To set column G for two decimal places, I used /F,C,G,$. The second letter (C) in the Format command stands for column. The third letter is the actual Column designation.

To add new items to the list, I place the cursor on the line below where I want the new entry and enter /I,R,return. Since the cursor is at the right place, the carriage return following the "R" (row) command will insert a new line above the current line. If I wanted to insert a new row just above row 25, another alternative would be to enter /I,R,25.

I normally set up the basic spreadsheet with calculations before I enter headers. Usually, I drop to line 5-8 to begin calculations and to allow room for the headings. I use the "=" sign to draw most lines across the page. I move the cursor to the left hand cell where I want to start the line and enter a left hand quote mark ("), then depress the = key and the repeat key at the same time. I then use the Replicate or Copy command to copy the line to other places where I need a line. For example, I copied the line I started in cell B6 to cell B4 by entering /C,B6,B4. The Replicate command would be /R,B6,B4.

Using Grocer

Before my wife and I go to the grocery store, we take a pantry inventory using last month's printout. I then enter the figures from the inventory in the (ON-HAND) column. I also inventory our coupons for each item and enter that number in the (CPN) column. I then hit the manual calculate key "!", print the list, and go to the store. I use manual calculate on this spreadsheet to speed the data entry process. This feature is set using /G,M. On automatic calculate, a complete recalculation is done after each entry, preventing the next entry until the calculation is complete. With manual feature the computer allows me to type ahead and not wait on recalculation.

I change the date each month by using the Edit command. I move the cursor to the date in cell F3, then enter /E,. I then press the carriage return twice and make the changes that need to be made.

Possible Modifications

The columns could be easily squeezed and more information added if it were useful to the family. If I were going to add another column, I would reduce columns C, D, and F in order to create a column 10-14 spaces wide. Column A and J (2 spaces) could easily be eliminated if needed.

Brand name listings may be important for some families wishing to use such a spreadsheet. The following are the steps I would take to add a Brand Name column immediately after the Column D. First, enter /F,C,C,3 to reduce Column C to 3 spaces. Then enter / F,C,D,18 to reduce that column to 18 spaces. Follow this by entering /F,C,F,4. With these format changes, there are now 14 spaces for the new column. Next, enter /I,C,E. This would insert a column immediately after column D. To expand the new column to 14 spaces, enter /F,C,E,14. Then move the cursor to E5 and enter "Brand. You could then add the appropriate brand names as text using the left hand quote mark.

The coupon column is used as a flag to check coupons when in the grocery store. I had considered entering the amount of the coupons, but decided the value wasn't worth the extra time involved in entering the data.

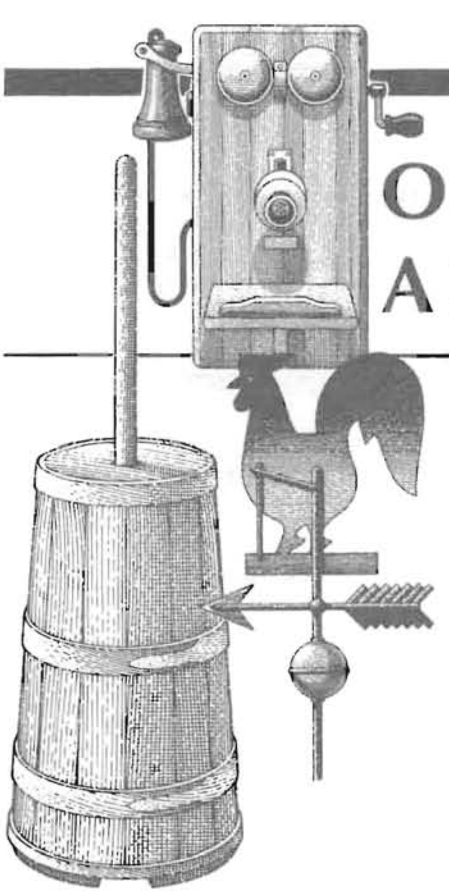
Before going to the grocery store, I line out the items that I will not have to purchase. With Multiplan or SuperCalc2, I would be able to sort on column A. I could then use the /B command to delete all the items with 0's. I could then resort on Column D and return the remaining list to alphabetical order. Of course, I would need to make any needed saves before Blanking items.

My wife and I find our SuperCalc Grocer to be a very functional and efficient approach to grocery shopping. It takes us less than 30 minutes to complete the inventory, update the file, and print the list. This approach has significantly reduced return visits to the store, thus aiding the family budget process.

This is one example of how SuperCalc has allowed me, a non- programmer, to use the computer to aid my family in managing family finances, rather than the family having to change to manage it the computer's way.

"HERO I CHATS" CORRECTION

Refer to "HERO I Chats With the H89" which appeared in REMark Issue 39, page 41. In the diagram of the 8255 chip, pin 40 should read pin 13.



Out In The Boonies With A Single Drive H/Z-89

Bill Pinkston
PO Box 488
Saltillo, MS 38866

Ever want to run up to 3 disks under HDOS in your single H-17 type drive, copy files between HDOS and CP/M with one drive, break large files down to sort or edit-then recombine in order, make a cheap two chip video output for your terminal, or have your computer protect your home without leaving it on all the time? If so, this article is for you.

Stand Alone

First of all I always run my H/Z-89 in stand-alone mode to open up more disk space. This mode is entered by typing in 'SET HDOS STAND-ALONE'. After that, all you must do is LOAD any device drivers you need and reset the drive by typing 'RESET SY0:'. Disks other than the bootable ones need SYSCMD.SYS in order to keep from causing a reboot message when exiting to HDOS. PIP.ABS is useful for cataloging, renaming, and copying but not mandatory for normal operation like SYSCMD.SYS!

A useful program to have made up would be a file named PROLOGUE.SYS to LOAD your normal device drivers and dismount SY0:, waiting for a return, then mounting SY0:. This program is on my bootable disks and is called R.ABS on the nonbootable disks so I may just type 'R' to change disks. A program very similar to mine is in REMark, Issue 16, page 10; in fact I have updated my program to use it's improvements plus my loading of drivers and setting the motor run time to 8 seconds. Most of this knowledge

E5.BAS PROGRAM LISTING

```
10 CLEAR 300
20 PRINT "THIS PROGRAM FORMATS HDOS DISKS FOR CO-RESIDENT CP/M USE.":PRINT
30 LINE INPUT "ENTER DRIVE TO BE FORMATED: ";DR$:PRINT
40 IF RIGHT$(DR$,1)<>":" THEN DR$=DR$+":"
50 OPEN "O",1,DR$+"CPM.SPC":'CP/M AREA
60 INPUT"HOW BIG IN SECTORS ";S:FOR I=1 TO S
70 PRINT #1, STRING$(255,&HE5):'FILL SECTORS WITH E5s
80 NEXT I:'  ---- GET THE NEXT SECTOR
90 CLOSE
100 PRINT "FUNCTION COMPLETED.":PRINT
110 PRINT "ANYMORE DISKS TO BE FORMATTED? <N> ";:A$=INPUT$(1) :PRINT A$:PRINT
120 IF A$="Y" THEN PRINT: GOTO 30
130 SYSTEM
150 'SAVE"E5",A
```

FILEBRK.BAS PROGRAM LISTING

```
10 CLEAR 5000'          FILEBRK.BAS  Breaks files into N sectored parts
20 ON ERROR GOTO 160:C=ASC("A"):ES$=CHR$(27):E$=ES$+"E":G$=ES$+"j":P$=ES$+"k"
30 PRINT:LINE INPUT"RESET SY0: (Y/N) <Y> ";I0$:IF I0$<>"N" THEN RESET"SY0:
40 PRINT:LINE INPUT"INPUT FILE NAME  >";I$:OPEN"I",1,I$:D$=ES$+"x5"
50 PRINT:LINE INPUT"OUTPUT FILE NAME (No Point!)  >";O$:U$=ES$+"y5":@
   C9$=CHR$(C):O$=O$+".X":O2$=O$+C9$:OPEN"O",2,O2$
60 PRINT:INPUT"INTO HOW MANY SECTORS PER FILE (EVEN NUMBER PLEASE)  >";S:S=S-1
70 PRINT E$:PRINT"FILES MADE";TAB(14);"SECTORS";TAB(25);"RECORDS"
80 PRINT STRING$(32,126);D$:PRINT G$;
90 LINE INPUT #1,R$
100 PRINT #2,R$:N=N+1:PRINT O2$;TAB(17);LOC(2)+1,N:PRINT P$;:R$=""
110 IF LOC(2)=S THEN CLOSE #2:PRINT:PRINT G$;:NT=NT+N:@
    N=0:C=C+1:C9$=CHR$(C):O2$=O$+C9$:OPEN"O",2,O2$
120 GOTO 90
130 NT=NT+N:PRINT U$:PRINT:@
    PRINT TAB(27);NT;"RECORDS TRANSFERED FROM FILE ";I$:PRINT
140 CLOSE:IF I0$<>"N"THEN PRINT:LINE INPUT"RETURN TO RESET SY0:";E$:RESET"SY0:"
150 CLOSE:SYSTEM
160 IF ERL=40 THEN PRINT TAB(40);"CAN'T FIND!";CHR$(7):RESUME 40
170 IF ERL=50 THEN PRINT TAB(40);"CAN'T OPEN!";CHR$(7):RESUME 50
180 IF ERR=63 THEN RESUME 130'  INPUT PAST END
190 IF ERR=51 OR ERR=30 THEN PRINT:@
    PRINT TAB(30);CHR$(7);"**ERROR** LAST FILE NOT CLOSED, DO AGAIN":@
    RESUME 130
200 PRINT U$:ON ERROR GOTO 0'     STOP ON OTHER ERRORS
210 'SAVE"FILEBRK",A
```

has been presented to me by REMark Magazine, by the way!

3 In 1 Disk Drive

On the disk drive unit you have, no doubt, been sent to cut the jumpers to set up your drive to identify as SY0: (DS3). One day it hit me that if I would connect DS2 then the drive would be SY1: and connecting DS1 would make it SY2:. To experiment with this new idea, I went out and bought a DIP switch and 'extension cord' to put it out where I could get to it. To make it quieter in here I immediately switched to the HM instead of the HS jumper; no more clunk, clunk, clunk. Then I could select which 'drive' I was using by just switching the drive select to mount, catalog, and run programs from SY1:. This worked fine if I reset SY0: first as to permanently load the overlays. Then, I made a 'mistake', I had the drive selects on for SY0: and SY1:! To my amazement HDOS could 'tell' me, buy returning to the track zero microswitch up to ten times, that I had the 'wrong' disk in the drive as long as the volume numbers were different. All I needed to do then was open the door and try another 'mounted' disk. This was a great improvement over going to cassette device drivers for secondary storage and opened up new flexibility in programs that don't support SY0: resets like old BASIC, EDIT, etc., but will allow a file to be on SY1:.

Some programs read a sector, then write when ready and are a lot of trouble, like ASM, as disks must be changed a lot. Others read all the source, then write the output like FORTRAN or L80 and are, of course, easier to use with 'one drive'.

CP/M To HDOS On One Drive

If you read how to get HDOS and CP/M to act neighborly, REMark Issue 21, page 5, then you should also be informed of how to have a bigger CP/M area. Just INIT with 1.6 and almost all of the disk may be E5'ed. So up to about 184 sectors for a 46K CP/M program can be moved to the HDOS side simply by copying the CP/M program to this area, running CTOH, and calling this disk SY0: AND SY1:. Making a smaller directory, like 4 sectors, will push you close to 48K! This does not resolve the syntax differences but will move FORTRAN, MBASIC, etc. source code or other ASCII type data. Going to CP/M may also be done but files expand slightly due to the CR-LF on the end of line. A friend of mine that is CP/M only and I, HDOS only, were swappng source code this way for over a year until we added second drives.

'Multi-Drive' Problems

Note that when you do either of these 'Hardware cheats' you should have complete

FILEMRG.BAS PROGRAM LISTING

```
10 CLEAR 5000'         FILEMRG.BAS Merges N files made by FILEBRK.BAS
20 ON ERROR GOTO 150:C=ASC("A"):ES$=CHR$(27):E$=ES$+"E":G$=ES$+"j":P$=ES$+"k"
30 PRINT:LINE INPUT"RESET SY0: (Y/N) <Y> ";I0$:IF I0$<>"N" THEN RESET"SY0:
40 PRINT:LINE INPUT"OUTPUT FILE NAME  >";I$:OPEN"O",2,I$:D$=ES$+"x5"
50 PRINT:LINE INPUT"INPUT FILE NAME (No Point!)  >";O$:U$=ES$+"y5":@
   C9$=CHR$(C):O$=O$+".X":O2$=O$+C9$:OPEN"I",1,O2$
60 PRINT E$:PRINT"FILES READ";TAB(14);"SECTORS";TAB(25);"RECORDS"
70 PRINT STRING$(32,126);D$:PRINT G$;
80 LINE INPUT #1,R$
90 PRINT #2,R$:N=N+1:PRINT O2$;TAB(17);LOC(1),N:PRINT P$;:R$=""
100 GOTO 80
110 CLOSE #1:PRINT:PRINT G$;:NT=NT+N:@
    N=0:C=C+1:C9$=CHR$(C):O2$=O$+C9$:OPEN"I",1,O2$:GOTO 80
120 NT=NT+N:PRINT U$:@
    PRINT TAB(27);NT;"RECORDS TRANSFERED TO FILE ";I$:PRINT
130 CLOSE:IF I0$<>"N"THEN PRINT:LINE INPUT"RETURN TO RESET SY0:";E$:RESET"SY0:"
140 CLOSE:SYSTEM
150 IF ERL=40 THEN PRINT TAB(40);"CAN'T OPEN!";CHR$(7):RESUME 40
160 IF ERL=50 THEN PRINT TAB(40);"CAN'T FIND!";CHR$(7):RESUME 50
170 IF ERR=53 THEN PRINT:RESUME 120'    FILE NOT FOUND
180 IF ERL=80 THEN RESUME 110'  INPUT PAST END
190 IF ERL=90 THEN PRINT:PRINT TAB(40);"OUT OF ROOM ON ";I$:RESUME 120
200 PRINT U$:ON ERROR GOTO 0'    STOP ON OTHER ERRORS
210 'SAVE"FILEMRG",A
```

FILESMRG.BAS PROGRAM LISTING

```
10 CLEAR 4000'                  FILESMRG.BAS
20 DIM RF$(4,3):ON ERROR GOTO 320:DBUG=1:IF DBUG THEN PRINT:PRINT"DBUG ON"
30 PRINT:PRINT"FILESMRG.BAS MERGES SORTED FILES TO MAKE A SORTED OUTPUT"
40 OPEN"I",5,"DIRECT.SYS":CLOSE:DU$=STRING$(81,126)
50 PRINT:PRINT:INPUT"NUMBER OF INPUT FILES ";FI
60 IF FI>4 THEN PRINT TAB(40);"1-4 ONLY":GOTO 50
70 IF FI=0 THEN PRINT TAB(40)"4 ASSUMED":FI=4
80 FOR OL=1 TO FI'        START OPEN INPUT LOOP
90 PRINT:PRINT"INPUT FILE NAME FOR CHANNEL #";OL;
100 LINE INPUT" >";IC$
110 OPEN "I",OL,IC$'     OPEN CHANNEL OL WITH FILE IC$
120 NEXT OL'             END OPEN INPUT LOOP
130 PRINT:PRINT:PRINT:LINE INPUT"OUTPUT FILE NAME >";O5$
140 OPEN"O",5,O5$
150 INPUT"WIDTH OF SORT FIELD ";W
160 INPUT"START OF SORT FIELD ";S
170 FOR RR=1 TO 4        'START READ RECORD LOOP
180 IF  RF$(1,1)="/*" @
    AND RF$(2,1)="/*" @
    AND RF$(3,1)="/*" @
    AND RF$(4,1)="/*" @
    THEN CLOSE:SYSTEM
190 IF RF$(RR,1)="" THEN @
    RF$(RR,1)="R":@
    LINE INPUT#RR,RF$(RR,2):@
    RF$(RR,3)=MID$(RF$(RR,2),S,W):IF DBUG THEN PRINT"READING CHANNEL #";RR
```

☞

```
200 IF DBUG THEN PRINT"RR=";RR,"<";RF$(RR,3);">"
210 NEXT RR                   'END   READ RECORD LOOP
220 OT=0                                                  'START "SORT"
230 IF RF$(1,3) <= RF$(2,3) AND @
       RF$(1,3) <= RF$(3,3) AND @
       RF$(1,3) <= RF$(4,3) THEN OT=1:GOTO 270
240 IF RF$(2,3) <= RF$(1,3) AND @
       RF$(2,3) <= RF$(3,3) AND @
       RF$(2,3) <= RF$(4,3) THEN OT=2:GOTO 270
250 IF RF$(3,3) <= RF$(1,3) AND @
       RF$(3,3) <= RF$(2,3) AND @
       RF$(3,3) <= RF$(4,3) THEN OT=3:GOTO 270
260 IF RF$(4,3) <= RF$(1,3) AND @
       RF$(4,3) <= RF$(2,3) AND @
       RF$(4,3) <= RF$(3,3) THEN OT=4:GOTO 270
270 IF OT=0 THEN OT=1:PRINT CHR$(7);"** OT=0 **"'END   "SORT"
280 PRINT #5,RF$(OT,2)           'OUTPUT SELECTED RECORD
290 RF$(OT,1)=""                 'CLEAR FOR READ RECORD
300 IF DBUG THEN PRINT"OUTPUT FROM CHANNEL #";OT:PRINT RF$(OT,2)
310 GOTO 170
320 IF ERL=40 AND ERR=52 THEN GOTO 360            'START ERROR TRAPS
330 IF ERL=110 AND (ERR=53 OR ERR=65) THEN PRINT"CAN'T FIND":RESUME 90
340 IF ERL=190 THEN RF$(RR,1)="/*":RF$(RR,2)=DU$:RF$(RR,3)=DU$:RESUME 210
350 ON ERROR GOTO 0                    'END   ERROR TRAPS
360 PRINT:PRINT"SINCE YOU DIDN'T SET UP 5 FILES, DO AGAIN !"
370 PRINT:PRINT">SYx:MBASIC SYx:FILESMRG/F:5";:SYSTEM
480 'SAVE"FILESMRG",A
```

CALLHELP.BAS PROGRAM LISTING

```
10 '    CALLHELP.BAS               TAKE REM'S OFF LINES 20 AND 30 WHEN IN USE!!!!!
20 REM  PRINT CHR$(27);"}"'DISABLE KEYBOARD
30 REM  ON ERROR GOTO 330'NEED FILES VO: & CALLHELP.NUM
40 CLEAR 8000
50 OUT 249,33:PRINT"START UP ON"'IF NEW PORT DECODER THEN TRY PORTS 56 & 57
60 FOR D9=0 TO 3000:NEXT'MAKE SURE WE GET DIAL TONE FIRST TIME ANYWAY
70 PRINT T$,
80 POKE 8264,2' SET DISK MOTOR TO 1 SEC OVERRUN
90 OPEN"I",#1,"CALLHELP.NUM"'FILE OF NUMBERS, ONE PER RECORD ie: 8445170
100 IF EOF(1) THEN GOTO 330 ELSE LINE INPUT#1,D$'GET NUMBER
110 FOR D9=0 TO 900:NEXT D9
120 FOR D=1 TO LEN(D$):PRINTTAB(50) LEN(D$)
130 FOR D9=0 TO 50:NEXT D9
140 C9=(VAL(MID$(D$,D,1))):IF C9=0 THEN C9=10
150 FOR C=1 TO C9:PRINT (VAL(MID$(D$,D,1)));
160 OUT 249,0:FOR D9=0 TO 20:NEXT D9
170 OUT 249,33:FOR D9=0 TO 7:NEXT D9
180 NEXT C
190 NEXT D
200 FOR D9=0 TO 500:NEXT D9:PRINT'      DELAY AFTER DIAL
210 OPEN"O",#2,"VO:"
220 READ TK$
230 IF TK$="/*"THEN N=N+1:RESTORE
240 GOTO 260
250 PRINT"NEXT CALL IF NOT EOF":GOTO 100
260 PRINT#2,TK$
270 IF N=4 THEN N=0:GOSUB 360:GOTO 100
```

☞

back-ups as things sometimes can go wrong (and will go wrong). The soft error rate will jump to record heights because the head will be on the wrong track when the identity of the drive changes, and the wrong volume when you must change disks. These are not usually big problems but can cause an unreadable disk once in a while, so be safe! Do not use the direct sector type programs like DUP in this way as they will not work.

Breaking Large Files Down To Size

The FILEBRK.BAS file breaks your input file down into smaller files of whatever size in sectors you specify with the extensions changed to put back together with FILEMRG.BAS for regular files or FILESMRG.BAS for combining in sorted order. Of course you could put your files back together with a PIP or COPY command like BIG=FILE1,FILE2 etc. but a section of nulls will be between files which might cause some programs to malfunction.

Video Output On An H/Z-19 Or H/Z-89

This hardware addition will support a 75 ohm video output into a monitor or a VCR. More circuitry could be added to make the vertical sync shorter as it covers some of the first line. If a TV set is used, this will not help much as TV sets are overscanned and the edge information will be lost anyway. The main use for this output is just to check on other operators to see if the assembly is done or if errors have occurred, while being sixty feet away. Secondary uses include recording the giant spider from the Y-WING game to see just how to get by or watching someone else play and laughing. I could have used just the exclusive or gate and used one chip, but that was an after thought and is unproven.

Theft Protection With An H/Z-89

The biggest real problem threatening my computer system, in this area, is power surges and failures. Because of this I have had a latching relay to turn the computer on and off since the beginning. More recently I made a circuit to turn this relay on when the place was armed if the door and window switches' continuity was interrupted. Add to this the auto-boot option, a program to dial the phone using the cassette relay (who said the cassette board was useless!), and a Votrax unit with audio coupled to the phone line; and instant protection! The program outputs to the VO: device driver as I bought this package from the Studio Computers, Inc. *[Ed. Note: Previously known as Keyboard Studio.]*Phone numbers to dial are in a file so changing is easy. I have the Sheriff's Dept. number first, then the number where I expect to be, each repeated three times. I patched track 0 sector 2 byte 2C from 3C to 01 to make a fast boot on the question 'Action <Boot>' plus a PROLOGUE.SYS to 'LOAD VO:' and run 'MBASIC CALLHELP' by filling the type ahead buffer to complete this system.

```
280 GOTO 220
290 DATA "BURG LER E, BURG LER E IN PROG RESS AT BILL PINKSTON RES AH DENCE"
300 DATA "AT ~YSzR? TRAILER PARK OFF OLD SAL TILLO ROAD,
310 DATA "FIRST TRAILER ON RIGHT,,"
320 DATA "/*"
330 OUT 249,0:PRINT"EXIT":OUT 249,33
340 PRINT CHR$(27)"(":CLOSE'ENABLE KEYBOARD
350 POKE 8264,15:SYSTEM' SET DISK MOTOR TO 7.5 SEC OVERRUN & EXIT
360 CLOSE#2:RESTORE:FOR D9=0 TO 9000:NEXT:'DELAY FOR VOTRAX TO SPOOL
370 OUT 249,0:OUT 248,0:PRINT"SIREN RELAY ON"'EXTERNAL LATCH REQUIRED
380 FOR D9=0 TO 1000:NEXT:PRINT"SIREN RELAY OFF":OUT 249,0
390 OUT 249,33:PRINT"ON"
400 RETURN
410 'SAVE"CALLHELP",A
```

```
PHONE TO VOTRAX INTERFACE BOX
~~~~~ ~~ ~~~~~~ ~~~~~~~~~ ~~~

PLAY RELAY ON CASSETTE BOARD
      CONNECTS HERE                     (RECORD RELAY FOR SIREN LATCHING RELAY)
          *   *
           * *    AUDIO
            *  TRANSFORMER
       *---^ ^---(0)!!(0)----*-------------------------\
       !         (0)!!(0)    !                          \
       !         (0)!!(0)    <                           \
       !         (0)!!(0)    > 10 OHM                     \ VOTRAX
       !         (0)!!(0)    > RESISTOR                   / AUDIO OUTPUT
       !         (0)!!(0)    <                           /
PHONE--'         (0)!!(0)    !                          /
LINE-------------(0)!!(0)----*-------------------------/
```

```
VIDEO OUTPUT BOARD PROTOTYPE        7486 XOR            BILL PINKSTON
~~~~~ ~~~~~~ ~~~~~ ~~~~~~~~~        --------            ~~~~ ~~~~~~~~
 *   = CROSSING CONNECTED           !1   14!+5V
-!-  = CROSSING NO CONNECTION       !2   13!
                                    !3   12!
                      *-------------!4   11!
                      !      *------!5   10!
                      !  *---!------!6    9!
                      !  !   !  GND !7    8!
                      !  !   !      --------
                      !  !   !
                      !  !   '------------------------*
                      !  !                            !
                      !  !   *-------------------*    !
                      !  !   !                   !    !
                      !  !   !      7404 INV     !    !
                      !  !   !      --------     !    !
                      !  '---!------!1   14!+5V  !    !
*---------------------!------!------!2   13!-----!----!-----------HORIZ P14-8
!                     !      '------!3   12!-----!----'
!                  *--!-------------!4   11!-----!----------------VIDEO P14-10
!                  !  !      *------!5   10!-----'
!                  !  '------!------!6    9!-----------------------VERT P14-6
!                  !         !  GND !7    8!----*
!                  !         !      --------    !
!                  !         !                  !                  GND  P14-9
!                  !         '------------------'
!                  !
!                  '----------------*
!                                   !
!                                   !
'------*                            !
       !                            !
       V                            V
+5V--/\/\/\-----------*-----------/\/\/\
    500K              !          500K
                      !
                      !
                      '-------------------------------------------VIDEO OUT \
                                                                             >
                                                                        GND /
```

Getting Started With Assembly Language

(or, What's That Other Thing For?)

Pat Swayne
Software Engineer

This article is the third installment in a series on assembly language programming. Readers who missed the first two may want to obtain copies of REMark Issues #39 and #40 to catch up. In those issues, I discussed console input and output in HDOS and CP/M. This month, I will help you get some work out of that other gadget on your computer table, your printer.

Part IV - Using a Printer in HDOS

Even after new programmers get as far as being able to write a program like the console I/O examples presented in this series on their own, they may still have trouble handling a printer. That is because operating systems usually do not have the kind of built-in support for printers that they have for the console. It is also because many programmers make it look harder than it is, and students who look at the work of those programmers find it difficult to figure out what is going on. This brings up the second of my rules for assembly language programming.

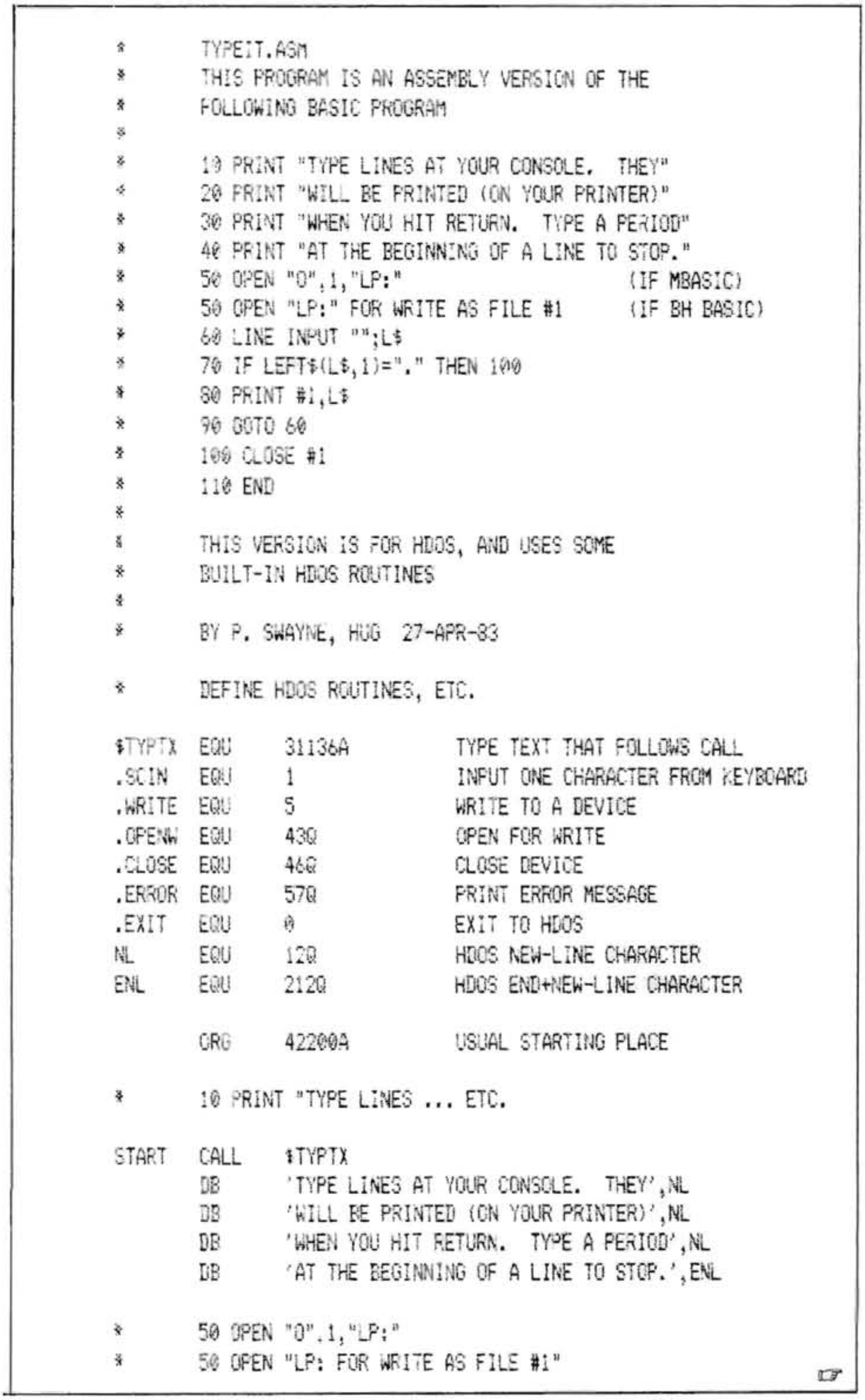
Rule 2. *Include a generous amount of comments in your program, with the idea in mind that someone may be trying to learn from your work, even though your purpose in writing it is not to teach.*

If you follow this rule, you'll wind up teaching yourself a thing or two, by figuring out for yourself how things work in order to write the comments.

A Computer Typewriter

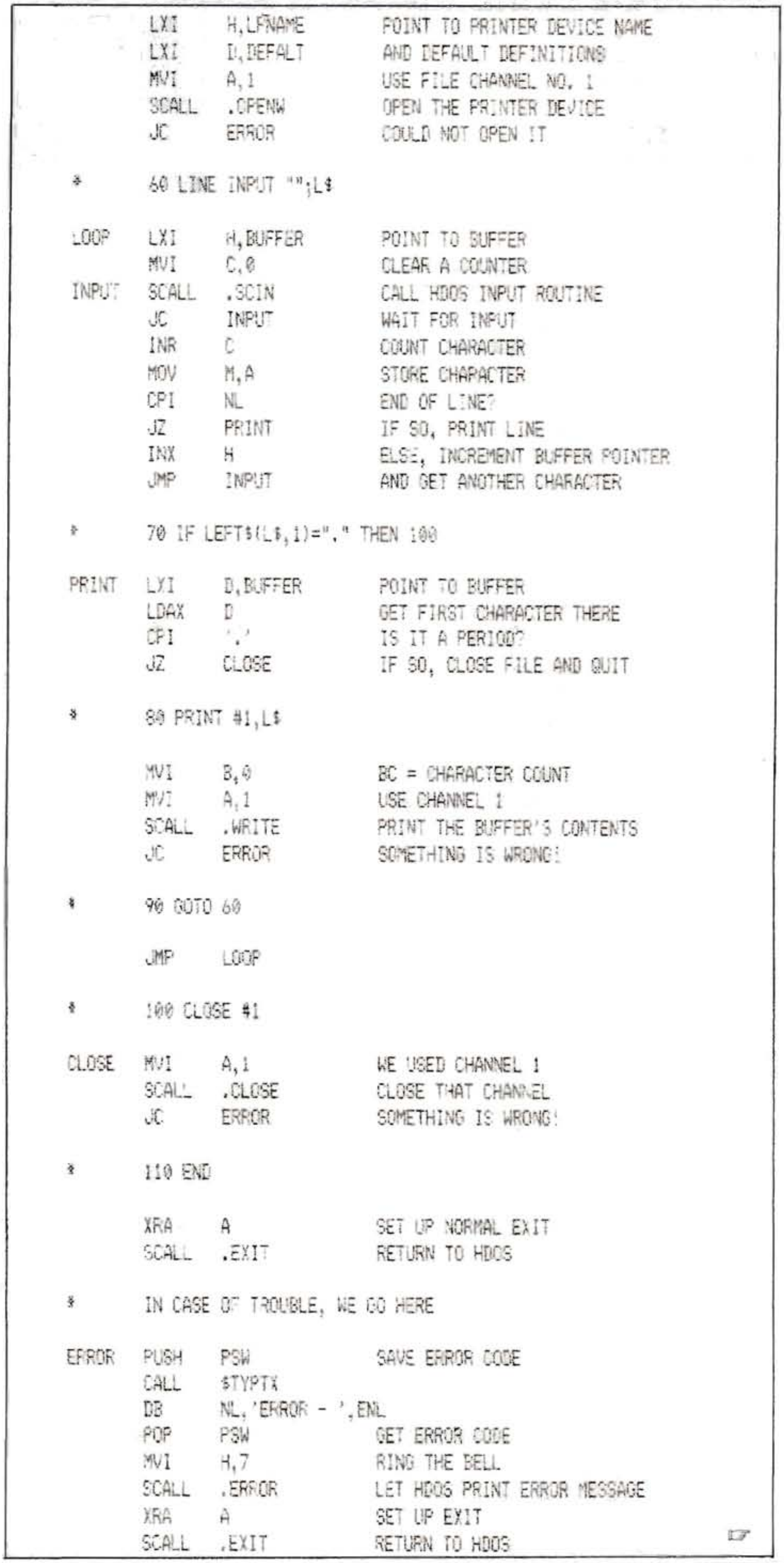
The example program I will use in this part turns your computer into a simple typewriter, with the ability to correct a line before it is committed to paper. Listing 1 shows the program, and, as before, you can see from the comments that it is an assembly language translation of a BASIC program. If you try to run the BASIC program, use only one of the two lines numbered 50. Choose the one indicated for the version of BASIC you are using.

Following the comments, the program has some EQUate statements that define the operating system calls that it uses, and other



```
*       TYPEIT.ASM
*       THIS PROGRAM IS AN ASSEMBLY VERSION OF THE
*       FOLLOWING BASIC PROGRAM
*
*       10 PRINT "TYPE LINES AT YOUR CONSOLE.  THEY"
*       20 PRINT "WILL BE PRINTED (ON YOUR PRINTER)"
*       30 PRINT "WHEN YOU HIT RETURN.  TYPE A PERIOD"
*       40 PRINT "AT THE BEGINNING OF A LINE TO STOP."
*       50 OPEN "O",1,"LP:"                   (IF MBASIC)
*       50 OPEN "LP:" FOR WRITE AS FILE #1    (IF BH BASIC)
*       60 LINE INPUT "";L$
*       70 IF LEFT$(L$,1)="." THEN 100
*       80 PRINT #1,L$
*       90 GOTO 60
*       100 CLOSE #1
*       110 END
*
*       THIS VERSION IS FOR HDOS, AND USES SOME
*       BUILT-IN HDOS ROUTINES
*
*       BY P. SWAYNE, HUG  27-APR-83

*       DEFINE HDOS ROUTINES, ETC.

$TYPTX  EQU     31136A          TYPE TEXT THAT FOLLOWS CALL
.SCIN   EQU     1               INPUT ONE CHARACTER FROM KEYBOARD
.WRITE  EQU     5               WRITE TO A DEVICE
.OPENW  EQU     43Q             OPEN FOR WRITE
.CLOSE  EQU     46Q             CLOSE DEVICE
.ERROR  EQU     57Q             PRINT ERROR MESSAGE
.EXIT   EQU     0               EXIT TO HDOS
NL      EQU     12Q             HDOS NEW-LINE CHARACTER
ENL     EQU     212Q            HDOS END+NEW-LINE CHARACTER

        ORG     42200A          USUAL STARTING PLACE

*       10 PRINT "TYPE LINES ... ETC.

START   CALL    $TYPTX
        DB      'TYPE LINES AT YOUR CONSOLE.  THEY',NL
        DB      'WILL BE PRINTED (ON YOUR PRINTER)',NL
        DB      'WHEN YOU HIT RETURN.  TYPE A PERIOD',NL
        DB      'AT THE BEGINNING OF A LINE TO STOP.',ENL

*       50 OPEN "O".1,"LP:"
*       50 OPEN "LP: FOR WRITE AS FILE #1"
```

☞



```
        LXI     H,LPNAME        POINT TO PRINTER DEVICE NAME
        LXI     D,DEFALT        AND DEFAULT DEFINITIONS
        MVI     A,1             USE FILE CHANNEL NO. 1
        SCALL   .OPENW          OPEN THE PRINTER DEVICE
        JC      ERROR           COULD NOT OPEN IT

*       60 LINE INPUT "";L$

LOOP    LXI     H,BUFFER        POINT TO BUFFER
        MVI     C,0             CLEAR A COUNTER
INPUT   SCALL   .SCIN           CALL HDOS INPUT ROUTINE
        JC      INPUT           WAIT FOR INPUT
        INR     C               COUNT CHARACTER
        MOV     M,A             STORE CHARACTER
        CPI     NL              END OF LINE?
        JZ      PRINT           IF SO, PRINT LINE
        INX     H               ELSE, INCREMENT BUFFER POINTER
        JMP     INPUT           AND GET ANOTHER CHARACTER

*       70 IF LEFT$(L$,1)="." THEN 100

PRINT   LXI     D,BUFFER        POINT TO BUFFER
        LDAX    D               GET FIRST CHARACTER THERE
        CPI     '.'             IS IT A PERIOD?
        JZ      CLOSE           IF SO, CLOSE FILE AND QUIT

*       80 PRINT #1,L$

        MVI     B,0             BC = CHARACTER COUNT
        MVI     A,1             USE CHANNEL 1
        SCALL   .WRITE          PRINT THE BUFFER'S CONTENTS
        JC      ERROR           SOMETHING IS WRONG!

*       90 GOTO 60

        JMP     LOOP

*       100 CLOSE #1

CLOSE   MVI     A,1             WE USED CHANNEL 1
        SCALL   .CLOSE          CLOSE THAT CHANNEL
        JC      ERROR           SOMETHING IS WRONG!

*       110 END

        XRA     A               SET UP NORMAL EXIT
        SCALL   .EXIT           RETURN TO HDOS

*       IN CASE OF TROUBLE, WE GO HERE

ERROR   PUSH    PSW             SAVE ERROR CODE
        CALL    $TYPTX
        DB      NL,'ERROR - ',ENL
        POP     PSW             GET ERROR CODE
        MVI     H,7             RING THE BELL
        SCALL   .ERROR          LET HDOS PRINT ERROR MESSAGE
        XRA     A               SET UP EXIT
        SCALL   .EXIT           RETURN TO HDOS
```

☞

definitions. You probably have figured out by now that I nearly always start out a program this way. Although the assembler will allow EQUates to be anywhere in a program, I think it is a good idea to put them at the beginning just in case a programmer who uses Pascal or another language which requires definitions at the beginning wants to study the program.

I covered $TYPTX, .SCIN, and .EXIT previously, and will not discuss them here. NL is a common acronym used to represent the HDOS New-Line character, and ENL is the New-Line character with the 8th bit set.

The program starts by printing instructions on the console, using the $TYPTX routine. Then the line printer device is opened. HDOS treats all communications with the outside world (that is, to printers, disks, etc.) in the same way, so the basic techniques used in this program to output to a printer will also apply to disk operations (which I will cover in a future installment). You can even output to the console using the method used for the printer in this program, although it is not efficient since there is extra support for console I/O in HDOS. Many of you may already know this, if you experimented with printing to the "TT:" device in BASIC.

When you open a device in assembly language, you must point the HL register pair to a "file name descriptor". Since we are not really opening a file, but a printer, our descriptor is just "LP:", followed by a termination character. In this program, a zero terminates the name, but you can also use a comma, slash, or space. The DE register pair points to a 6 byte area called the "default block descriptor". For file operations, the first three characters of this block can contain a default device and unit number (such as SY0), and the last three characters can be a default extension (such as .BAS). If the file name descriptor contains only a name, HDOS will use the device and extension in the default block. Since our device is a printer, and we have already defined it in the file name descriptor, our default block contains all zeros.

The number of the file channel that will be used for the printer is placed in the A register, and then the program uses the HDOS .OPENW (Open for Write) system call to open the printer device driver. There is also .OPENR for reading from a file, and .OPENU for random file access. If the open operation fails, HDOS returns to the program with the Carry flag set, which causes a jump to an error exit in the program.

After the printer device is opened, the program enters a loop designated by the label LOOP. At the beginning of the loop, it inputs a line from the console using a technique we have already discussed. Note that in this program we are not limiting the number of characters input, as we did last time. However, we do count the characters, and that count will be used later. After a line is input, the program checks to see if the first character entered is a period, and exits from the loop if it is.

```
*       DATA AND STORAGE AREA

LPNAME  DB      'LP:',0         PRINTER DEVICE NAME
DEFALT  DB      0,0,0,0,0,0
BUFFER  DS      110             MAIN INPUT-OUTPUT BUFFER

        END     START
```

Next, the input line, which is stored in the location called BUFFER, is sent to the printer. To send characters to a printer or any device in HDOS, you point the DE register pair to the buffer containing the characters, put the count of characters in the BC registers, and the channel number in register A. Then the HDOS .WRITE routine takes care of the rest. The HDOS System Programmer's Guide states that the number of characters (bytes) in a file I/O operation must be a multiple of 256, but actually that only applies to disk operations. BASIC uses the same routines for both disk and printer operations, and so it sticks to the 256 multiple rule even while printing, which explains why the BASIC version of the example program will not produce anything on the printer until you have typed a few lines.

This illustrates one of the advantages of assembly language programming in that we can tailor our program specifically for printer operations and do not have to worry about rules governing disk operations. It takes a bit of "software overhead" to put characters into a buffer and then output the buffer's contents when it is full. It is easier to just put the exact number of characters to print into the BC register, or to print one character at a time, than it is to maintain a separate 256-byte multiple buffer.

Since there already is a count of characters in the C register when it is time to print, we just put a zero in the B register to make BC equal to the character count. DE already points to the buffer because we put it there when we checked for a period as the first character. All that remains is to put the channel number in the A register and let HDOS send the characters to the printer. If something goes wrong, HDOS returns with the Carry flag set to indicate an error.

After printing the line of text, the program jumps back to LOOP to get another line. When the user types a period as the first character in a line, the program goes to the label CLOSE. All you have to do to close a file in HDOS is to put the channel number in the A register and let HDOS do the rest with .CLOSE. There is little chance of anything going wrong here, but we have provided the usual jump to ERROR if the Carry flag is set. After the channel is closed, the program exits to HDOS.

The error processing section of the program uses an HDOS system call that uses error code in the A register (placed there either by an HDOS routine such as .OPENW or by a programmer) to look up an error message in a file called ERRORMSG.SYS and print it on your screen. If ERRORMSG.SYS is missing from your system disk, the number of the error is printed, and you can look it up in your HDOS manual. The .ERROR system call, which prints the message, also prints whatever character is in the H register, so we put 7, the ASCII code to beep your terminal, in H. The program exits to HDOS after the message is printed.

After the error section is the data and storage area. This contains the file name descriptor for the printer device, the default block, and a buffer for inputting lines of text. The buffer is 110 characters big because that is the most that HDOS will let you type in any one line when you use .SCIN in the line input mode.

Next month, I will present a CP/M version of this program. HDOS users might want to read that article because printing in CP/M is a "whole 'nother ball game".

BE A HAPPY HUGGER.

Z-100 Users! Call us for floppy & hard disk add-on prices. We have 8" & 5¼" drives for you!

Get to know Floppy Disk Services.

We are contracted dealers for Siemens, Shugart and Tandon disk drives. You may do a double take when you see our prices, but there's no catch—we simply buy in large quantities and sell at reasonable prices.

We've sold thousands of drives to our Heath/Zenith friends. And with our good service and fair guarantee, we keep our friends friendly.

Check our prices on these Heath/Zenith compatible drives . . .

FDD-100-5 'flippy' (Just like your internal H89 drive)	$215.00
FDD-221-5 DS/DD 96tpi (80 track) same as TM-100-4	330.00
FDD-200-5 same as 100-5 but double sided (2 heads)	250.00
FDD-100-8d5 SS/DD 8 inch with 3ms step! Sale!	240.00 2 for $450.00 ea.
FDD-200-8 DS/DD 8 inch 3ms step Sale!	345.00

New Shugart half height disk drives!!

SA-455 double sided, double density, 48tpi half height 5¼" step time to 6ms!	295.00 2 for $245.00 ea.
SA-465 double sided, double density, 96tpi half height 5¼" step time to 6ms!	350.00 2 for $295.00 ea.
SA-860 double sided, double density, 8 inch half height. Step time to 3ms!	495.00 2 for $445.00 ea.

PAYMENT POLICY We accept Mastercard, Visa, personal checks & M.O. We reserve the right to wait 10 working days for personal checks to clear your bank. All shipping standard UPS rates unless otherwise requested. New Jersey residents *must* add 6% sales tax. Due to production deadlines, prices in this ad could be as old as 2 months. If in doubt, call!

Prices and specs subject to change without notice.

Need an enclosure for 5¼ or 8"?

Single vertical case w/ps A&T styled to match Heath color	$ 50.00
Dual vertical case with power supply A&T	75.00
Dual horizontal case with dual floppy supply A&T	125.00
Dual horizontal 8" A&T	275.00

Half Height Enclosures

Floppy Disk Services has designed these new half height enclosures from the ground up! We had you, the end user in mind and designed these 5 models:

Vertical dual 5¼ half height	$ 75.00
Single full size 5¼ or up to 2 half height	125.00*
Dual side by side horizontal half height 5¼	125.00*
Dual side by side horizontal half height 8 inch	295.00*
Single full size 8 inch or up to 2 half height	295.00*

* These enclosures are of professional design and have data cableing options available at extra charge.

** all of the above enclosures include A&T with power supply.

Magnolia 8" Controller

Magnolia Microsystems controller for the Heath H88, 89, 90 enables you to use any combination of 8 or 5¼ single or double sided, single or double density—up to 4 of each size drive! You even get CP/M in 8 and 5¼ format serialized for you. This controller is available at a special price of $525.00.

Our pledge to you . . .

All Floppy Disk products are brand new and 100% warranted. If you are unhappy with our systems or components for any reason, we will refund your money promptly. There are two requirements: 1. the equipment must be in as good shape as you received it; and 2. you must call or write for a return authorization. This offer is good for 30 days, beginning with your invoice date. Feel free to call us with any questions.

We make Huggers happy. Call today!

Toll Free Order Line	***(800) 223-0306***
Tech Help or Info.	***(609) 799-4440***

Dual Heath Add-on FDD-100-5

Dual Half Height or Single Full Size 8"

FLOPPY DISK SERVICES™ INC.

741 Alexander Rd.
Princeton, NJ 08540

CP/M and H88-89-90 are registered trademarks of Digital Research and Heath Co., respectively.

Introduction To Z-BASIC Part VII

Gerry Kabelman, C.E.T.
Zenith Data Systems

This is the seventh article in a series of articles dealing with the new commands of the H/Z-100's Z-BASIC over BASIC-80. Previous articles have dealt with mostly graphic commands and this article will introduce some additional commands found in Z- BASIC.

The first new command is actually a replacement for the PRINT CHR$(7) or ASCII bell. The BEEP command may be used at any time in a program as it is a stand-alone command. The BEEP command may NOT be used with the PRINT command. It must be separate, either on a separate line or separated by a colon (:).

```
10 BEEP
20 IF X>Y THEN PRINT"Now ring the bell":BEEP
```

The above two lines will beep the bell in line 10. Then in line 20, if the value of X is greater than the value of Y, it will beep it again. Note when using the IF-THEN statement, if the expression is false, the line is terminated unless the ELSE command is located within the same line.

```
 30 IF X>Y THEN PRINT"Now ring the bell":
    BEEP:ELSE PRINT"No Bell!"
```

The above line will print one of the messages and ring the bell only if the expression is true.

Let's take a look at the COLOR command. The syntax for the COLOR command is COLOR [Foreground] [,[Background]]. The background color is optional. If the background is left off, the background will not change.

Valid colors for the COLOR command on the H/Z-100 computers are:

0 Black
1 Blue
2 Green
3 Cyan
4 Red
5 Magenta
6 Yellow
7 White

By using the COLOR command, we can change the foreground and background colors to any of the above colors. By intermixing the color combinations, we can make up to 36 colors. Intermixing is placing two dots next to each other to give a color or shade that appears as a color. Below is a sample program of how the 36 colors may be obtained.

```
10 '      COLORS.BAS            Version 04.21.83          GK:
20 CLS:LINE(0,0)-(639,224),4,B:COLOR 4,6
30 LOCATE 3,28:PRINT" Available H/Z-100 Colors "
40 COLOR 7,0:LOCATE 17,35:PRINT"Colors":SCREEN 1
50 FOR I=0 TO 7:FOR D=0 TO 7:COLOR D,I:IF D<I THEN 90
60 K=K+1:LH=9:IF I=D THEN LH=0
70 FOR J=6 TO LH+3:LOCATE J,K*2+2:PRINT"ii";:NEXT J
80 COLOR 7,0:LOCATE J,K*2+2:
                           PRINT STR$(D);:LOCATE 17,41:PRINT K
90 NEXT D,I:SCREEN 0:COLOR 7,0:K=0:LOCATE 20,20
100 PRINT"Press any key to do it again or RETURN to END. ";
110 A$=INPUT$(1):IF A$<>CHR$(13) THEN 20
120 END
```

The above program will display the 36 color combinations from black to white. A SCREEN command was also introduced in this program. The SCREEN command has two functions. In the above program, we are using it to enter and exit the H/Z-19 graphics modes in lines 40 and 90. The SCREEN command may also be used to enter and exit reverse video. The syntax for the SCREEN command is:

SCREEN [Graphics] [,Reverse Video]

Graphics	0 - Clears or exits H/Z-19 graphics mode.
	1 - Sets or enters H/Z-19 graphics mode.
Reverse Video	0 - Clears or exits H/Z-19 reverse video.
	1 - Sets or enters H/Z-19 reverse video.

If the value for either graphics or the reverse video is left off, Z-BASIC assumes that a zero (0) was entered.

Examples:

SCREEN ,1	No graphics. reverse video on.
SCREEN 1	Graphics, reverse video off.
SCREEN 1,1	Graphics, reverse video on.
SCREEN ,	No graphics, reverse video off.

The SCREEN command may also be used to determine what character is in a location on the screen. Below is a sample program that uses the SCREEN command to determine what character is a location. The syntax is a little different in that parentheses enclose coordinates and attribute expression.

SCREEN (Row,Column [,expression])

```
10 '      SCREEN.BAS     Version 04.21.83              GK:
20 CLS:LOCATE 12,10:
                PRINT"What is the X coordinate? (1 to 25) ";
30 LINE INPUT A$:X=VAL(A$):IF X<1 OR X>25 THEN BEEP:GOTO 20
40 LOCATE 14,10:PRINT"What is the Y coordinate? (1 to 80) ";
50 LINE INPUT A$:Y=VAL(A$):IF Y<1 OR Y>80 THEN BEEP:GOTO 20
60 LOCATE 16,10:PRINT"What character is to be printed? ";
70 A$=INPUT$(1):CLS:LOCATE X,Y:PRINT A$:LOCATE 20,10
80 SC=SCREEN(X,Y)
90 PRINT"The character at location"X","Y"is "CHR$(SC)
100 LOCATE 22,10
110 PRINT"Press any key to do it again or RETURN to END";
120 A$=INPUT$(1):IF A$=CHR$(13) THEN CLS:LIST ELSE 20
```

The above program will ask for the two coordinates and the character to be printed. It will then display the character that the SCREEN command in line 80 has found at the location of X and Y and the ASCII value of the character is assigned to the variable SC. Using a variable allows the character to be located in the line that we are using to print the message in line 90. If a variable is not used, the character may be changed as we print line 90. If the command CHR$() was left off, only the ASCII value of the character would be printed in line 90.

When using the SCREEN command to determine the color (attribute) of a character, the expression must be included along with the row and column of the desired location. The expression must not be equal to zero or only the character will be returned.

Add or change the following lines in the above program, to show how the color of a character on the screen may be obtained.

```
51 LOCATE 18,10:
            PRINT"What color is the foreground? (0 to 7) ";
52 LINE INPUT A$:FC=VAL(A$):IF FC>7 THEN BEEP:GOTO 51
53 LOCATE 20,10:
            PRINT"What color is the background? (0 to 7) ";
54 LINE INPUT A$:BC=VAL(A$):IF BC>7 THEN BEEP:GOTO 53

70 A$=INPUT$(1):CLS:COLOR FC,BC:
                          LOCATE X,Y:PRINT A$:LOCATE 20,10
80 SC=SCREEN(X,Y):S1=SCREEN(X,Y,1) mod 8:COLOR 7,0

91 LOCATE 21,10:PRINT"The color of the character is"S1
```

The color will be returned as a number from 0 to 7. Again, note how a variable is used in line 80, (SC = ASCII value of the character and S1 = the color of the variable).

The SCREEN command allows identifying of the character at a given location on the screen while another command called the POINT command allows an individual dot's color to be identified.

The syntax for the POINT command is:

POINT (Horizontal,Vertical)

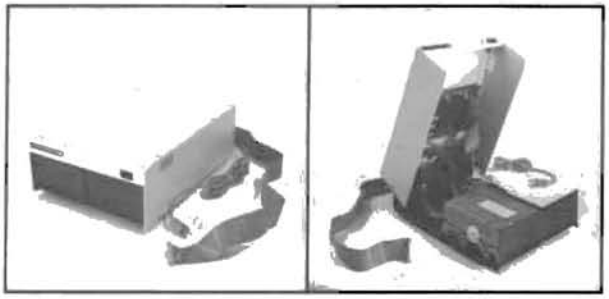
To demonstrate the use of the POINT, we must first turn on a point and then check to see what color was turned on at that point. In the example below, we select the location of the dot and the color of that dot. Using the PSET command the dot is changed to the color we want, and using the POINT command the color of the dot is returned to the variable PT in line 80.

```
10 '      POINT.BAS       Version 04.21.83        GK:
20 CLS:LOCATE 12,10:
            PRINT"What is the X coordinate? (0 to 639) ";
30 LINE INPUT A$:X=VAL(A$):IF X>639 THEN BEEP:GOTO 20
40 LOCATE 14,10:PRINT"What is the Y coordinate? (0 to 224) ";
50 LINE INPUT A$:Y=VAL(A$):IF Y>224 THEN BEEP:GOTO 20
60 LOCATE 16,10:PRINT"What color is the point? (0 to 7) ";
70 LINE INPUT A$:C=VAL(A$):IF C>7 THEN BEEP:GOTO 60
80 CLS:PSET(X,Y),C:PT=POINT(X,Y):LOCATE 18,10
90 PRINT"The color of the point"X","Y"is"PT:LOCATE 20,10
100 PRINT"Press any key to continue or RETURN to END ";
110 A$=INPUT$(1):IF A$=CHR$(13) THEN CLS:LIST ELSE 20
```

The SCREEN and POINT commands may be used to find the character or dot of a location on the screen in a game or a drawing program. An example of a program using the POINT would be a chase or shooting gallery game where something moves across the screen and if it hits something, points may be scored or lost. Try writing one yourself.

Next month some additional commands will be explained. Any suggestions for subjects on Z-BASIC programming, please send them to me in care of the Heath Users' Group.

*



Eliminate the Limitations!

Recapture the Excitement of High Technology in your H/Z89-90

DG SUPER 89 ....$829.00

The Super 89 **(Now including 128K Memory and the Super 89 Electronic Disk)** replaces the central processor board in the Heath/Zenith 89-90 series computers to bring your 89-90 to current state-of-art technology. The Super 89 transforms the Heath/Zenith 89-90 into a powerful, professional quality system meeting the needs of today and tomorrow. The Super 89 is fully compatible with all Heath/Zenith products and also supports many peripheral devices from other manufacturers. New software and hardware are enhanced with the Super 89 by using all the features of the Z80 technology.

The Highlights of the Super 89:

- Shipped with 128K memory
- Includes Super 89 Electronic Disk
- Twice the operating speed (4MHz+)
- Memory Capacity to 256 K in Software Bank Selectable 64K blocks
- CP/M and HDOS Compatible without modification
- Twice the number expansion slots (Six)
- Real time clock on-board
- Two serial I/O Ports
- Designed for multi-user capability
- Parity checking for RAM assures integrity of memory transfer operations
- Arithmetic processor provision facilitates mathematic operations

Expanded Memory Capacity

This feature allows you to use the advantages of the more sophisticated programming languages; enables you to use enhanced memory software such as print spoolers and electronic disks to increase spped; allows the use of "scratch pad" memory to increase efficiency; and provide for multi user capabilities.

Super 89 Electronic Disk

This software package for the Heath/Zenith CP/M 2.2.03 allows the Super 89 user to access auxiliary RAM as a very fast mass storage device. Provides up to 180K bytes (fully populated) of storage area that is accessible without the slowness of disk drives. The Electronic Disk also includes display capability for the Real Time Clock.

Peripheral Expansion

This important feature lets you use your Super 89 in more ways with peripherals from DG, Heath and many other manufacturers.

Real Time Clock

The Real Time Clock allows you to program activities and control functions according to time; allows the use of interactive time functions with an electronic disk; and is very useful in accounting functions.

Parity Checking

This feature ensures the integrity of memory transfer operations. The Super 89 alerts you if a parity error occurs.

Full CP/M and HDOS Compatibility

The Super 89 supports full compatibility with the HDOS or CP/M disk operating systems. This feature gives you the best of both worlds in the amount of existing software you may use.

Ease of Installation

The Super 89 is simple to install and takes only minutes. No soldering required. Simply remove the old CPU board, configure and install the Super 89 to eliminate the limitations.

SUPPORT PRODUCTS for the SUPER 89

Arithmetic Processor $200.00

The Super 89 has on-board provisions for the optional AM9511A. This is a separate processor that features basic arithmetic as well as exponential, logarithmic, trigonometric and binary functions. Calculations are high speed and can be accomplished as a "hardware subroutine". This device is a must for anyone using any amount of mathematical computation whether complex functions or arithmetic calculations.

Enhanced Super 89 Monitor $49.00

Gives you all the features of Heath's MTR-89 monitor plus the ability to display all the Z80 register contents; Single-step through a program and set up break-points; Supports H/Z and other manufacturers of disk systems; Improved system diagnostic routines; and Supports the Super 89 Real Time Clock.

MP/M II® $550.00

MP/M II is a multi-user, multi-tasking operating system for use with the Super 89. Supports up to four users and the Z37, Z47 and Z67 disk systems. Compatible with CP/M software.

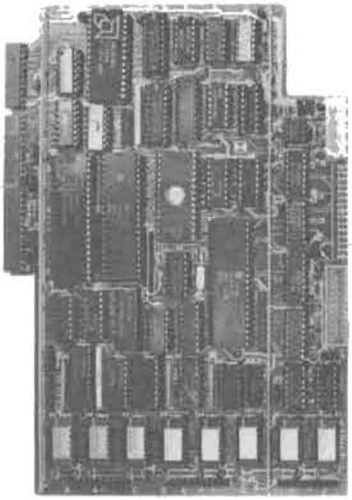
DG Super 89/128 K	**829.00**
DG Super 89/192 K	**909.00**
DG Super 89/256 K	**989.00**
Documentation Only	**25.00**

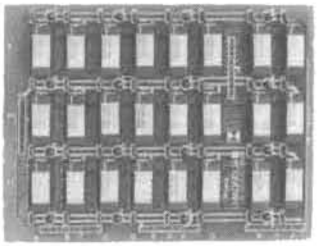
D·G ELECTRONIC DEVELOPMENTS CO.

Ordering Information: Products listed available from DG Electronic Developments Co., 700 South Armstrong, Denison, Tx. 75020. Check, Money Order, VISA or MasterCard accepted. Phone orders (charge only) call (214) 465-7805. Freight prepaid. Allow 3 weeks for personal checks to clear. Texas residents add 5%. Foreign orders add 30%. Prices subject to change without notice

Heath®, H8®, H17®, and H89® are registered trademarks of Heath Corporation, Benton Harbor, Michigan
Z80® is the registered trademark of Zilog Corporation. AMD9511 is a trademark of Advanced Micro Devices.
CP/M® and MP/M® are the registered trademarks of Digital Research of Pacific Grove, California

INTRODUCING -- MAGNOLIA's MOST POWERFUL BOARD YET!





The Z89/90 is a good solid computer. But it does have certain limits.

We've had many requests to increase its CPU speed to 4MHz, but nothing met our standards for both field installation and reliability. And who needs anything less.

We increased its RAM to 176K with our 128K board, allowing use of MP/M™. However, without DMA, multi-user operations under MP/M are slow.

Networking '89s together has been a long-term goal of ours. Through the Corvus Constellation multiplexer we provided the first stage -- allowing computers to share Winchester disk storage. But that technique leaves much to be desired -- since each computer independently manages its disk allocation thera are significant operational limitations on sharing disk files.

We first announced our impending 'RS422 Network Interface' at the 1982 West Coast Computer Faire. Our design goals included:

- RS422 Network Communication at 800K baud
- On-board Network Control and Data Buffering

As it progressed, we realized that we had designed an 'I/O Board' which was FAR MORE POWERFUL THAN THE '89 ITSELF!

After agonizing delays, the board is now in production as the '77422 Network Controller'. Compare these specifications with the Z89 itself [or the new 'upgrade' CPU boards]:

- 4MHz Z80™ CPU
- 64K RAM, 256K [bank selectable] optional
- 8 K EPROM
- 4 CHANNEL DMA controller
- 2 Serial Ports: RS422 at 500KBaud
  RS232 at up to 19.2 KBaud

We also changed its name from '77321 Network Interface' to '77422 Network Controller'. Our 773xx products must be used in a Z89/90 computer. Although this product CAN be an I/O card in an '89 or 90, it can also be used independently as a stand-alone station on the network:

- permitting a terminal [Z19 or other] as a network station [with local disk storage], or
- permitting a printer to spool data directly from the network.

Network applications software is currently being developed and tested.

Software for an exciting non-network application is included. The workload is split between the Z89/90 CPU and the CPU on the '422 board:

- The Z89's 2MHz CPU continues to run Magnolia CP/M™, handling all physical I/O [console, printer, floppy and Winchester disks]
- The '422 board's 4MHz CPU runs an interface program which 'looks like' CP/M to an application program -- but passes CP/M system calls to the Z89's CPU for execution, allowing a 63K TPA on the '422 board!

With this division, the following Z89 limitations are overcome:

- The application runs on a 4MHz CPU
- The application has 63K of RAM available, independent of BIOS size
- The DMA controller [making a slave I/O processor of the '89 board] frees the application CPU from detailed I/O functions.

This last advantage is especially significant when executing MP/M on the '422 board, using the optional bank-selectable memory.

Hardware-dependent programs [including utilities like 'format'] which perform actual physical I/O, must run on the 'slow' Z89 CPU. A special utility is provided to 'mark' those application programs which can execute more efficiently on the '422 board CPU.

The board is presently available in 64K and 256K versions which mount in an I/O slot [P504/P510] on the '89 CPU board. To limit power dissipation within the '89, no disk drive may be used internally.

64K Network Controller Board
order 77422-064-xx **$695**

256K Network Controller Set
order 77422-256-xx **$1295**

CP/M Operating System by MMS if needed
order CPM-xx add **$100**

MP/M II Operating System
requires 77422 order MPM-xx **$595**

CP/NET™ Network Software
requires 2 or more 77422 boards, one with MPM
order CP/NET-xx **$450**

Use 'xx' to specify preferred media format

MAGNOLIA MICROSYSTEMS
2264 - 15th W • Seattle, WA 98119
(206) 285-7266 (800) 426-2841

CP/M, MP/M II, and CP/NET are trademarks of Digital Research, Pacific Grove, CA.
Z80 is a trademark of Zilog, Cupertino, CA.

Using Binary Files With Benton Harbor BASIC

David A. Sandage
808 Oakland Ave. #107
Urbana, IL 61801

David Sandage is a graduate student in Computer Science at the University of Illinois. He received his B.A. degree in Computer Science at the University of California at Berkeley in 1982. He is a great fan of UNIX and HDOS. Besides computers, his interests include astronomy and photography.

One of the first things that I did when I switched over from cassette BASIC to HDOS and its version of BASIC, was to learn how to access files from a BASIC program. The HDOS manual does a fairly good job of explaining the use of ASCII files from BASIC, but there were things that I wanted to do that needed to use binary files. There were also things that I wanted to do with ASCII files, but needed some sort of end-of-file indicator. Both of these needs are taken care of by a Benton Harbor BASIC function called CIN.

CIN(x) reads one byte from a file that has been explicitly opened for read on channel #x, and returns that byte as its value. Let us first look at how CIN can work as an end-of-file function. If you call CIN to read from a file that actually has reached the end, CIN will return a negative value. This is fine for finding the end of a binary file, but remember that the last sector of an ASCII file is padded with NULL (ASCII 0) bytes from the end of text to the actual end of the sector. So, since most ASCII files do not have NULL bytes in the text, we can test for the end-of-text by looking for CIN to return a 0 or a negative number.

Let's take an example. Suppose you want to read lines of text from a file and do something with them, but you don't know exactly how many lines there will be. The following subroutine will read in a line of text from the file open on channel 1 and return it in the string variable L$. If, however, an EOF is encountered, the routine halts execution of the program.

```
100 B = CIN(1) : REM READ FIRST BYTE, MAYBE THE FIRST CHAR IN THE LINE
110 IF B <= 0 THEN STOP : REM WE HAVE REACHED THE END OF THE TEXT
120 LINE INPUT #1,"";L$ : REM GET REST OF LINE
130 L$ = CHR$(B) + L$ : REM JOIN FIRST CHAR WITH REST OF LINE
140 RETURN : REM RETURN TO CALLING ROUTINE WITH LINE IN L$
```

The routine could, of course, do something other than halt when an end-of-file is encountered. Using CIN as an end-of-file indicator with binary files is very similar except that you don't want to consider a value of zero to indicate EOF, since a binary program could indeed have bytes with value 0 in it.

Let's now look at some other ways to use CIN with binary files. One of the more useful things to do is to use an .ABS file as data for a BASIC program. Such a use might include disassembling and up or downloading from another system over the phone. Before we launch right into a discussion of using CIN, it will be useful to look at the format of an .ABS file under HDOS. As most of you may know, an .ABS file contains binary machine instructions which are directly executable by the CPU. The individual bytes mean nothing to us, and if you tried to list an .ABS file on the terminal, you would just get garbage. This is because the bytes are not ASCII characters. When you type "FNAME" at the HDOS prompt, the operating system looks for FNAME.ABS on the disk, loads it into memory, and transfers control to the start of the program. But how does HDOS know where in memory to load it, or how long it is, or where it starts? In fact, HDOS must be sure that it is actually an absolute binary program and not just a text file that may have an .ABS extension. It knows these things by reading what is known as the header at the beginning of the file. For an absolute binary file the header looks like this:

BYTE	MEANING
0	contains 377Q if a binary file (not text, etc.)
1	File type. (ABS = 0, PIC = 1, etc.)
2,3	Load address
4,5	Length of entire record (in bytes)
6,7	Start Address.

As you can see, the header for an absolute binary file is 8 bytes long. A PIC file has a 6 byte header. For more information on this, you can look at FILDEF.ACM, ABSDEF.ACM, and PICDEF.ACM on your HDOS distribution disks. These headers are just appended to the beginning of any binary file. You don't have to worry about these bytes yourself when you are writing a program because the assembler or compiler adds the header for you.

With the above information, we are now ready to use an .ABS file as input to a BASIC program. In listing #1, I have presented a program to convert an .ABS file to an ASCII file containing a series of DB statements which can be used by ASM as input to reproduce the original .ABS file. Don't worry if you don't know what a DB statement is. It is just an assembler command to write out what follows it to the assembler's output file. For example, the statement:

DB 23,56,114,127

just writes those numbers to consecutive bytes in the assembler's output file (the binary program). What good is this, you may ask. Why bother to convert to ASCII, and then back to binary when we already have the binary file? Well, suppose you have written a program in FORTRAN and want to share it with a friend who doesn't have a FORTRAN compiler, or you don't want to distribute your source code. You could just make a copy of your program on a disk, but if you want to do it by MODEM, you may be stuck. Most MODEM programs allow the transfer of ASCII files, but not binary files. This is where my program comes in. You can convert your .ABS file to an .ASM file, send it over the phone, and your friend can just assemble it back into an .ABS program for his own use.

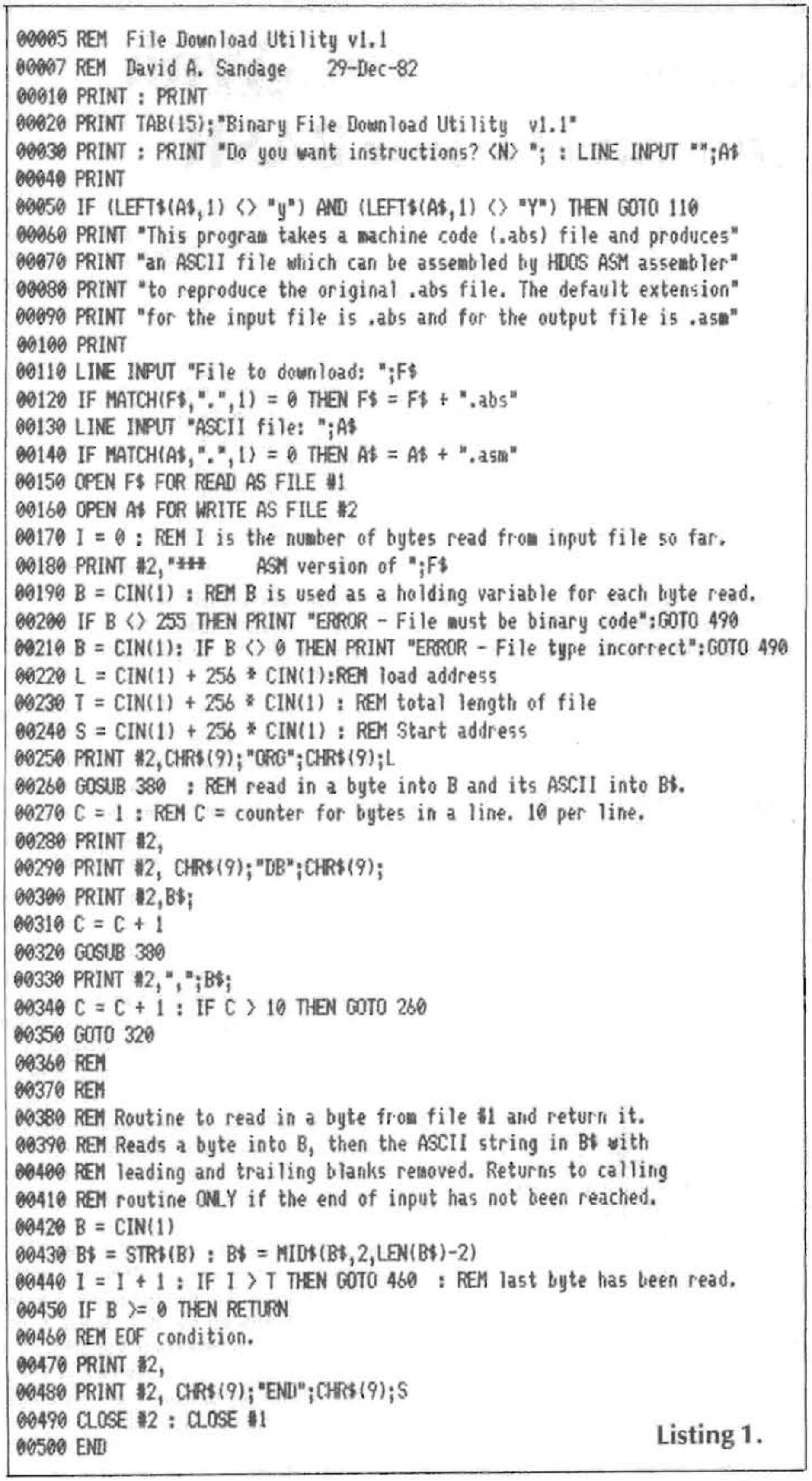
The program itself is fairly straightforward. After asking the user if he or she wants a description of the program, it prompts for the name of the binary file to convert and the name of the ASCII file in which to put the assembly language output. If no extensions are specified, .ABS is assumed for the input file, and .ASM for the output file. This is done very easily by checking whether or not the input string (file name) contains the character ".". If so, then an extension was specified, otherwise it just adds the appropriate string to the file name. The program then opens the two files. Starting at line 190, it reads in the header. It checks the first byte to make sure the user has specified a binary file. If not, it prints an error message, closes the files and exits. If the first byte does indicate a binary file, then it determines the type of file by examining the second byte of the header. If the file is not of type .ABS, then the program quits. This could be modified to handle PIC files also, but that would be a slightly more complex case. If the file type is OK, then it reads and stores the rest of the information in the header. The load address is stored in L, the length of the file in T, and the start address in S.

The first assembly language statement to be output is an ORG statement. This tells the assembler where the program is to be loaded in memory when it is run. We already know that this is the address in L, so we output:

ORG L Whatever L happens to be.

From here, we just enter a loop which reads bytes from the input file and prints DB statements to the output file. It will print 10 bytes per line, all separated by commas.

The routine that reads bytes from the input file starts at line 380. It uses the CIN function to read a byte into B. It then converts the value in B to an ASCII string which is put into B$. The leading and trailing blanks are then stripped from B$. The variable I contains the number of bytes read so far. It gets incremented by one each time CIN is called. If I > T then all of the input file has been processed and we can exit. Also, if CIN returned a negative number, we have reached the actual end-of- file. If neither of the above cases hold, the routine simply returns to the main program with the string to be output in B$. If, however, we have reached the end of the input, then the routine prints an END statement along with the starting address (found



```
00005 REM  File Download Utility v1.1
00007 REM  David A. Sandage   29-Dec-82
00010 PRINT : PRINT
00020 PRINT TAB(15);"Binary File Download Utility  v1.1"
00030 PRINT : PRINT "Do you want instructions? <N> "; : LINE INPUT "";A$
00040 PRINT
00050 IF (LEFT$(A$,1) <> "y") AND (LEFT$(A$,1) <> "Y") THEN GOTO 110
00060 PRINT "This program takes a machine code (.abs) file and produces"
00070 PRINT "an ASCII file which can be assembled by HDOS ASM assembler"
00080 PRINT "to reproduce the original .abs file. The default extension"
00090 PRINT "for the input file is .abs and for the output file is .asm"
00100 PRINT
00110 LINE INPUT "File to download: ";F$
00120 IF MATCH(F$,".",1) = 0 THEN F$ = F$ + ".abs"
00130 LINE INPUT "ASCII file: ";A$
00140 IF MATCH(A$,".",1) = 0 THEN A$ = A$ + ".asm"
00150 OPEN F$ FOR READ AS FILE #1
00160 OPEN A$ FOR WRITE AS FILE #2
00170 I = 0 : REM I is the number of bytes read from input file so far.
00180 PRINT #2,"***     ASM version of ";F$
00190 B = CIN(1) : REM B is used as a holding variable for each byte read.
00200 IF B <> 255 THEN PRINT "ERROR - File must be binary code":GOTO 490
00210 B = CIN(1): IF B <> 0 THEN PRINT "ERROR - File type incorrect":GOTO 490
00220 L = CIN(1) + 256 * CIN(1):REM load address
00230 T = CIN(1) + 256 * CIN(1) : REM total length of file
00240 S = CIN(1) + 256 * CIN(1) : REM Start address
00250 PRINT #2,CHR$(9);"ORG";CHR$(9);L
00260 GOSUB 380  : REM read in a byte into B and its ASCII into B$.
00270 C = 1 : REM C = counter for bytes in a line. 10 per line.
00280 PRINT #2,
00290 PRINT #2, CHR$(9);"DB";CHR$(9);
00300 PRINT #2,B$;
00310 C = C + 1
00320 GOSUB 380
00330 PRINT #2,",";B$;
00340 C = C + 1 : IF C > 10 THEN GOTO 260
00350 GOTO 320
00360 REM
00370 REM
00380 REM Routine to read in a byte from file #1 and return it.
00390 REM Reads a byte into B, then the ASCII string in B$ with
00400 REM leading and trailing blanks removed. Returns to calling
00410 REM routine ONLY if the end of input has not been reached.
00420 B = CIN(1)
00430 B$ = STR$(B) : B$ = MID$(B$,2,LEN(B$)-2)
00440 I = I + 1 : IF I > T THEN GOTO 460  : REM last byte has been read.
00450 IF B >= 0 THEN RETURN
00460 REM EOF condition.
00470 PRINT #2,
00480 PRINT #2, CHR$(9);"END";CHR$(9);S
00490 CLOSE #2 : CLOSE #1
00500 END
```

Listing 1.

in S) to the output file, closes both files and exits. The output file then contains a program which can be assembled using .ASM to get back the original binary file. The biggest problem with this program is that it is VERY slow. This is due to the fact that we strip off the leading and trailing blanks from the ASCII representation of each byte before we print it. This is necessary because the ASM assembler will not allow spaces between bytes in a multiple byte DB statement. There may be a more efficient method to strip these spaces, but I could not think of one.

As you can see, there are many ways to use binary files in BASIC programs. In addition to reading .ABS programs, you can save disk space by writing all of your data files in binary and reading them in using CIN. Also, if you specify channel 0 with CIN, it reads from the terminal. It will return a value of -1 each time it is called until a carriage return is pressed, then it will return the ASCII value of each character in the line. Using this method, your program need not stop dead in its tracks while waiting for a line of input if it has something else to do in the mean time. In my experience, CIN is a useful function which makes Benton Harbor BASIC a very usable language.

HUG NEW PRODUCTS

885-1124 HDOS HUGMAN and Animated Movies $20.00

Introduction: HUGMAN is a video game similar to the popular arcade game PACMAN™. The player accumulates points while eating peanuts, power pills, fruit, and monsters, without being eaten by one of the nasty monsters. The player has the ability to design new HUGMAN boards.

Three additional programs allow the user to create animated "movies" on the screen of the H/Z-19 or H/Z-89.

Requirements: This disk requires the HDOS operating system version 2.0 on an H8/H19/H17 or H/Z-89 with a minimum of 32K of memory. Only one disk drive is required.

The programs have been written in Tiny PASCAL (HUG P/N 885-1086) and the source code is available for some of the programs. The executable (.ABS) files are included. Extensive documentation is included to aid the player in running HUGMAN and the animated package.

The following files are included on the HUG P/N 885-1124 HDOS HUGMAN and Animated Movie disk:

README	.DOC	HUGLIB	.TPI
HUGMAN	.DOC	MOVIE	.DOC
HUGMAN	.ABS	CAMERA	.ABS
HUGMAN	.TPS	CAMERA	.TPS
MAKBRD	.ABS	SPLICE	.ABS
MAKBRD	.TPS	SPLICE	.TPS
INTRO	.PIC	MOVIE	.ABS
HUGMAN1	.PIC	SING	.MOV
HUGMAN2	.PIC	CONST	.TPI
HUGMAN3	.PIC	VIDEO	.TPI
HUGVAR	.TPI	CONSOL	.TPI
HUGCNT	.TPI	INKEY	.TPI

The source is not included for MOVIE.ABS due to the lack of room on the disk.

Author: Gary Cramblitt

HUGMAN — The documentation of HUGMAN will explain how to get started playing HUGMAN. There are some brief instructions which must be followed to run HUGMAN.

Once HUGMAN has been started, the screen will display the first board for playing the game. The monsters are shown on the screen as the letters B, P, W, and M which stand for Bob, Pat, Walt, and Margaret, respectively, of the HUG Staff. Peanuts, fruit, and power pills are scattered throughout the maze. The player must eat these while evading the nasty monsters.

HUGMAN becomes "energized" whenever a power pill is eaten. This allows him to turn on the monsters and eat them. The power pill, of course, has a limited effect before HUGMAN returns to normal.

The arrows on the keyboard are used to manipulate HUGMAN around the board. The program keeps track of the score and awards an additional HUGMAN at the score of 10000.

MAKBRD.ABS — This HUGMAN program gives the player the feature of creating custom playing boards for HUGMAN. The documentation contains rules for designing custom boards.

MOVIE and ANIMATION — The movie and animation package can be used to create custom slide shows or animated "movies" on the screen.

The program CREATE is used to "paint" pictures on the screen. Reverse video and graphic characters are supported. The program has facilities for saving the pictures on disk. In addition, animated or "moving" displays can be created by making small changes to the picture, then saving the resulting "frames" one at a time.

SPLICE.ABS will join two or more animated sequences into one sequence.

The program MOVIE will show a picture of animated sequence on the screen. Projection speed can be adjusted and the sequence can be displayed over and over again in continuous mode.

SING.MOV is an example of animated sequence. The documentation contains detailed instructions on creating your own "movies".

Comments: This entire disk will provide many hours of enjoyment for those who wish to play the existing games. It will provide many more hours of enjoyment for those who wish to create their own HUGMAN and MOVIES.

885-1125 HDOS MAZE MADNESS $20.00

Introduction: The programs on this disk produce a number of mazes of fast action fun. One or two players must try to make their way through the mazes without being caught.

The author has provided instructions for modifYing the maze games. The documentation is easy to follow.

Requirements: This disk requires the HDOS operating system version 2.0 on an H8/H19/H17 or H/Z-89 with 32K of memory. Only one disk is required.

The programs are written in assembly language and the source code is included. The author provides instructions for modifying the source for anyone not familiar with assembly language.

The following files are included on the HUG P/N 885-1125 HDOS MAZE MADNESS disk:

```
README   .DOC        MAZEMAD  .ABS
MAZEMAD  .DOC        H8MAZE   .ABS
MODIFY   .DOC        MAZEMAD1.ABS
H8MODIFY.DOC         MAZEMAD2.ABS
MAZEMAD  .ASM        MAZEMAD3.ABS
MAZEMAD  .ACM
```

Author: John Sirera

MAZEMAD — MAZE MADNESS will generate any size maze from 1 by 1 to 39 by 11 cells. Each maze is inhabited by one to nine creatures, depending on its size. The player must work his way through the maze to only one exit point.

There are three types of creatures within the maze; KILLERs, WARPERs, and BLOCKERs. If caught by a KILLER, the player is instantly killed. To survive, the player must avoid each KILLER at all costs.

The WARPER will transport the player to another part of the maze. The new location may or may not be in a more advantageous spot in the maze.

The last creature is a BLOCKER, which if encountered will freeze the player for a brief time. While the player is unable to move, he is at the mercy of the KILLERs which are still free to move about.

Two players can play at one time. The 2, 4, 5, 6, and 8 keys on the keypad are used to move around the maze for one player. A second player will use the A, S, D, W, and Z keys.

The program MODIFY provides detailed instructions for changing the source code to modify existing parameters which creates the mazes. H8MODIFY explains how to prepare MAZEMAD for use with the H8 computer (8080 processor).

MAZEMAD1, MAZEMAD2, and MAZEMAD3 provide additional unique features to the mazes which are created.

Comments: The random mazes which are generated from MAZEMAD will provide endless hours of "frustration" and fun!

885-4600
Watzman/HUG ROM $45.00

Introduction: This product contains the integrated circuits (IC'S) to the Watzman/HUG ROM. The two (2) IC ROM set is the replacement parts for the code and keyboard encoder ROMs in the H19, H19A, or Z19 terminal, or the Terminal Logic board in the H89, H89A, Z89, or Z90. It can greatly add to the capabilities and usefulness of the terminal.

Requirements: The ROMs can be replaced in an H19, H19A, or Z19 terminal, or in the H89, H89A, Z89, or Z90. The documentation contains complete instructions for the installation of the ROMs.

Program Content: For a description of the features of the Watzman/HUG ROM, refer to the HUG Software Catalog P/N 885-1221 Watzman/HUG ROM source code.

HUG Price List

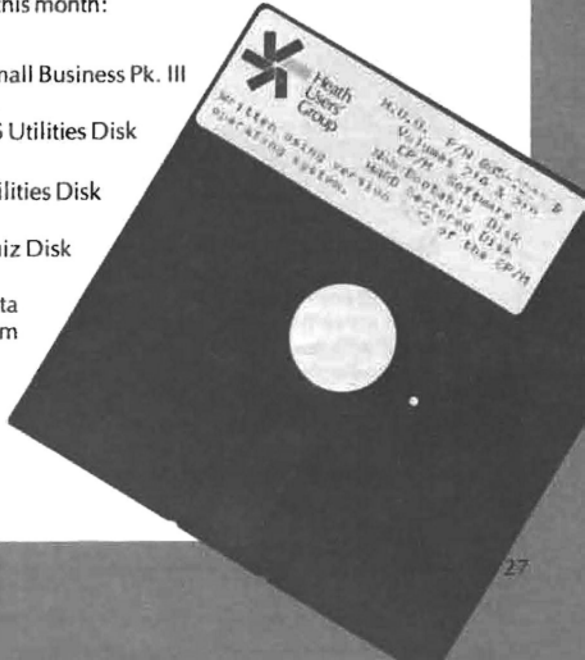
The following HUG Price List contains a list of all products not included in the HUG Software Catalog. For a detailed abstract of these products refer to the issue of REMark specified.

Part Number	Description of Product	Selling Price	REMark Issue
HDOS			
885-1029 [-37]	Disk II Games 1 H8/H89	$ 18.00	40
885-1060 [-37]	Disk VII H8/H89	$ 18.00	40
885-1062 [-37]	Disk VIII H8/H89 (2 Disks)	$ 25.00	40
885-1067 [-37]	Disk XI H8/H89 Games	$ 18.00	40
885-1086 [-37]	Tiny HDOS Pascal H8/H89	$ 20.00	40
885-1121	Hard Sectored Support Package	$ 30.00	37
885-1122	MicroNET Connection	$ 16.00	37
885-1123	XMET Robot & Cross Assembler ...	$ 20.00	40
CP/M			
885-1211 [-37]	Sea Battle	$ 20.00	36
885-1222 [-37]	Adventure	$ 10.00	36
885-1223 [-37]	HRUN HDOS Emulator	$ 40.00	37
885-1224 [-37]	MicroNET Connection	$ 16.00	37
885-1225 [-37]	Disk Dump and Edit Utility (DDEU) ..	$ 30.00	38
885-1226 [-37]	CP/M Utilities by PS:	$ 20.00	38
885-1227 [-37]	CP/M Cassino Graphic Games	$ 20.00	38
885-1228 [-37]	CP/M Fast Action Games	$ 20.00	39
885-1229 [-37]	XMET Robot & Cross Assembler ...	$ 20.00	40
885-3003 [-37]	ZTERM Modem Package	$ 20.00	36
885-8012 [-37]	Modem Appl. Effector (MAPLE)	$ 35.00	36
ZDOS			
885-3004-37	ZBASIC Graphic Games Disk	$ 20.00	37
885-3005-37	ZDOS ETCHDUMP	$ 20.00	39
MISCELLANEOUS			
885-0004	HUG 3-Ring Binder	$ 5.75	
885-4001	REMark VOLUME 1, issues 1-13 ...	$ 20.00	
885-4002	REMark VOLUME 2, issues 14-23 ..	$ 20.00	
885-4003	REMark VOLUME 3, issues 24-35 ..	$ 20.00	

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

The following five HDOS products are available in soft-sectored format beginning this month:

885-1071[-37]
MBASIC Small Business Pk. III
885-1089[-37]
HDOS Utilities Disk
885-1090[-37]
HDOS Utilities Disk
885-1097[-37]
MBASIC Quiz Disk
885-1108[-37]
MBASIC Data
Base System



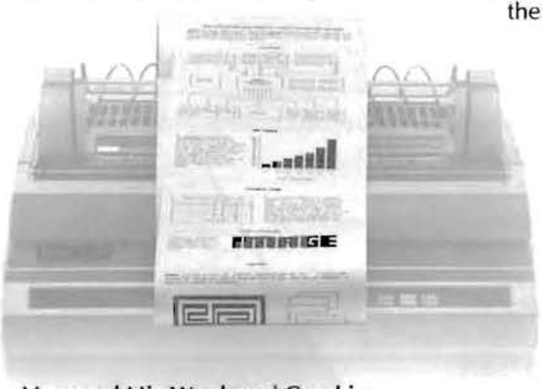
THE INTEGRATION OF WORD PROCESSING AND GRAPHICS.

THE **IMAGE WORD/GRAPHICS PROCESSOR** FOR

TeleVideo® AND ZENITH data® systems

Advanced Word Processing

Now you can enjoy a great screen-oriented word processor that provides powerful, new editing features. Besides headers, footers, page numbering, superscripts, subscripts, boldface, underline, and other standard W/P benefits, IMAGE gives you flush and indented word wrap, on-screen left, right, center, and full justification, variable cursor direction, and the simplest, most flexible column move in the industry.



Move and Mix Words and Graphics

Draw lines and blocks as easily as you did with your "Etch-A-Sketch." Then move what you've created or mix it with text anywhere on your terminal screen. Create bar charts, forms, organizational charts, flow charts, block letters, calendars, signs, invoices, purchase orders, diagrams, drawings, and more.

Point-and-Press Commands

Finally, say "goodbye!" to memorizing control codes, dot commands, and embedded printer gobbledygook. IMAGE displays commands for you directly above the top row of function keys. Point to the command you want, then press the key below.

InfoWorld
Software Report Card

Image

	Poor	*Fair*	*Good*	*Excellent*
Performance	□	□	□	■
Documentation	□	□	□	■
Ease of Use	□	□	□	■
Error Handling	□	□	□	■

Copyright 1982 by Popular Computing, Inc. a subsidiary of CW Communications, Inc., Framingham, MA—Reprinted from InfoWorld.

Immaculate printing quality

Previously obtainable from only the most expensive dedicated W/P systems, IMAGE provides true letter proportional printing for an impeccable, typeset look. No other system at any price will mix true proportional printing with graphics on a standard, letter quality printer.

Top marks from InfoWorld

"IMAGE certainly deserves accolades in the performance category... the documentation is simply superb. It is professionally done from cover to cover... without a doubt, this program is easy to use... the program is bombproofed so well that I had trouble finding any errors."

Extraordinary Value

Again from InfoWorld: "the modest $295 price tag is a bargain."

Specifications

Computers:
TeleVideo 802, 802H
TeleVideo 950 connected to Z-80 based CP/M system
Heath/Zenith Z89, Z90
Heath/Zenith Z19 connected to Z-80 based CP/M system

Printers:
Diablo, NEC, CItoh, EPSON, Zenith Z-25 and others.
(No terminal or printer retrofitting required.)

To Order Contact your local TeleVideo or Zenith dealer or call toll-free 1-800-MICROART (1-800-642-7627); in Oregon 1-692-3950.

MICRO ARTⓞ

MicroArt Corporation
200 Market Bldg., Suite 961
Portland, OR 97201

MasterCard, VISA, COD, and qualifying purchase orders accepted.

All quotes are from InfoWorld's *IMAGE* software review, by Marty Petersen, June 14, 1982.

BASIC Computing

A New Column is Coming

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

HUG is growing. And with that growth, REMark is expanding. With Issue 30, we had our first full-time editor in the person of Walt Gillespie. That issue had 39 pages. Issue 38 had 50 pages. Walt has made a lot of progress with a new format and a regular readers letter column. In the Issue 38 Editorial, he indicated a need for contributing editors to provide monthly columns in REMark. That's where we came in. A call was made to Walt and the BASIC Computing column was born.

The information which will be presented in this column is copyrighted by the author and is intended for the sole use of HUG members. It may not be reproduced without the written permission of the author and HUG. The programs to be presented are also proprietary and are copyrighted by "Applied Computing", an independent software vendor. By all means, use and enjoy this material. It is written so that we can share ideas and learn from them. But, please remember the restrictions. Much of the software to be written in this column is available commercially in its compiled form. It can be offered here because your author owns the company holding the copyright.

The system we're using to develop and test these programs is an H-89 with 64K, H-17 and H-37 disk controllers, 100K internal drive and 640K external drive, and the H-14 printer. We started out by running HDOS and Benton Harbor BASIC for six months, then we got MBASIC for the next six months. Finally, we moved over (notice I didn't say up or down) to CP/M and MBASIC. Why tell you all this? First, to let you know that all the programs presented here are written for CP/M and MBASIC. They can easily be converted to run under HDOS. Now you know that we've had experience with that too, so if you have a problem with the conversion, drop us a line and we'll be only too glad to help.

Now a bit about what we hope to accomplish in the coming months. Too many programs are written in a manner which makes them usable to the writer and other programmers, but not to Joe Average. We're not all programmers and many of us don't want to be. There's a real need for an understanding of what makes a good program great. Things like Computer Invisibility and Program Maintainability. Don't get me wrong. The programs referred to above are often real children of genius. They show great ingenuity and solve complex problems or perform difficult data processing, and they do it well. In a way, these programs are like a great cake without frosting, or donuts without coffee. They do the job, but they could do it better. Hopefully we'll be able to give you the ideas which can make this difference in your programs.

As this column progresses, we'll address programming techniques which make the computer lead you through the steps you must perform to complete a job. We'll also look at things like modular and structured programming. These are fancy words for writing a program in such a way that you can come back to it later and make updates or change its whole function easily. There will be an article on flow charting your programs and sub- programs. This is an all too often overlooked step in program development which makes debugging and future modifications much easier. Once you learn and practice it, you'll wonder how you ever got by without it. We'll include flow charts with many of the programs we present in future articles.

We also plan to build your sub-program and sub-routine library. While these will not be complete stand-alone programs, they will be sections of complete programs which can be merged to do the job you want, and will save a lot of typing of program lines you've written before.

In short, we hope to provide monthly tutorial articles which can be understood by the newest of computer owners, yet will go into sufficient depth to provide new information and techniques of programming and data processing for those who are accomplished in these areas.

What about reader feedback and the questions these articles will generate? If you have a question, or use a different method to do something we presented, please write me at the address given in the heading of this article. If it's something which will require an answer, include a stamped self-addressed envelope for my reply. The amount of correspondence a column like this can generate is monumental, and postage can get out of hand. We welcome, even hope for, correspondence. It's your way of letting us know whether we're doing our job and, most important, fulfilling your needs. That's what HUG is all about. Hopefully, we'll be able to periodically devote an entire column to your thoughts, ideas, and questions. A large show of interest in a single area or subject may be addressed as a feature article. So drop us a line and keep us up to date on what you want.

The specific subjects we have planned right now are:

Jumbo Characters
4-character-high letters and numbers we'll use later.

A Game of Words
Fun for the whole family with a touch of CAI.

Flow Charting
The answer to understanding how a program works.

Plotting Graphs
Curve and bar graph plotting on the CRT and printer.

Files and File Handling
An overview of data processing.

Random Files
How to get information into and out of your computer.

File Sorting 1
How to get your files organized.

File Sorting 2
Advanced sorting and index files.

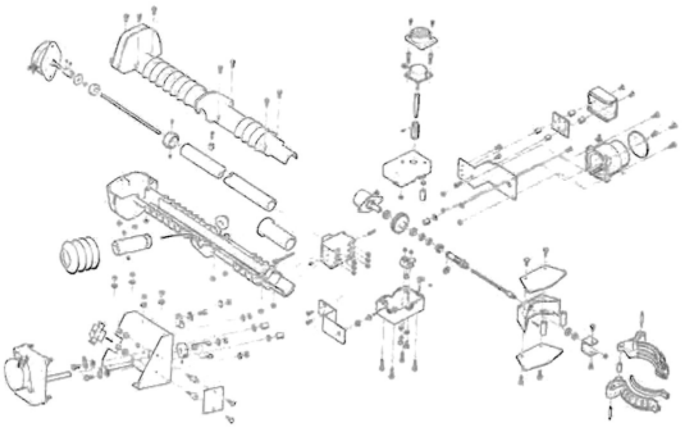
Data Base Management
Making your computer work for you.

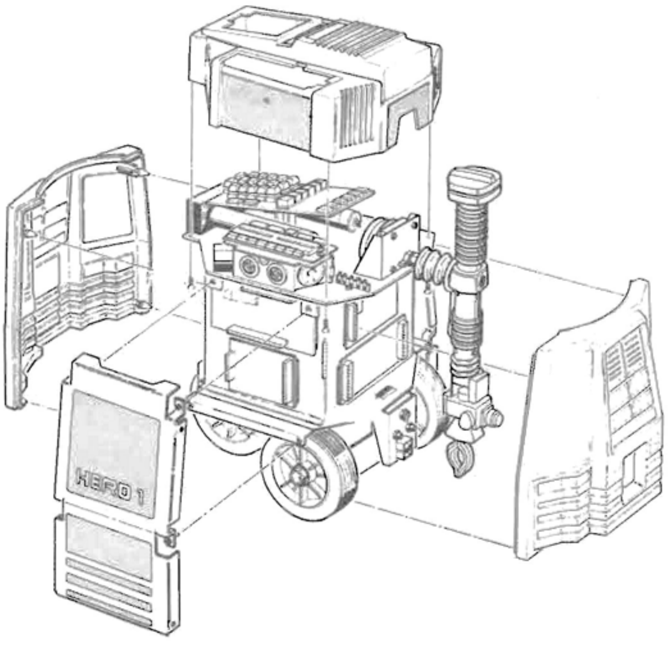
That should keep us busy for a while. A look at some of the subjects shows they may need more than one month to cover if we're to do it well and include the beginners. We're determined to do that, so the list may expand a bit in the middle. There's also the hope for a readers' ideas column in there somewhere, too. Before we close, here is one of those sub-programs we promised. It consists of a lot of common control characters used in future programs, so call up your MBASIC, type it in, and save it in the ASCII format using the command SAVE "CONTROL",A. This way it will be available for use with the large letters program we'll present with the next article. See you then.

```
10 *********** CONTROL.BAS ***********
20 ********* COPYRIGHT 1981 **********
30 ******* APPLIED COMPUTING *********
100 E$=CHR$(27)            'ESCAPE
110 E1$=E$+"Y"             'DIRECT CURSOR ADDRESS
120 E2$=E$+"E"             'CLEAR DISPLAY
130 E3$=E$+"x5"            'CURSOR OFF
140 E4$=E$+"y5"            'CURSOR ON
150 E5$=E$+"["             'ENTER HOLD SCREEN MODE
160 E6$=E$+"\"             'EXIT HOLD SCREEN MODE
170 E7$=E$+"F"             'ENTER GRAPHICS MODE
180 E8$=E$+"G"             'EXIT GRAPHICS MODE
190 E9$=E$+"p"             'ENTER REVERSE VIDEO MODE
200 E10$=E$+"q"            'EXIT REVERSE VIDEO MODE
```

Helpful Hints for Builders of Hero I

(Note: The following material was taken from CHUG, The Capital Heath Users' Group, Inc. Newsletter. CHUG, P. O. Box 2653, Fairfax, VA. 22031)





Mike Frieders, the first CHUG Member to complete the assembly of a HERO I, discovered a few things that could be very helpful to other builders.

Before even starting work on the project, go out and buy a crimping tool for the hundreds of spring connectors to be attached to the ends of the wires. It will save you an untold number of hours and you will do a much better job than can be done with long-nosed pliers. One such tool is the WALDON/Molex HT-1921 Crimper.

Instructions are silent on the installation of an insulator around the large hole in the Base Plate, but it is definitely needed and a piece of F11 insulator in Pack #14 will do nicely. Use it, you will not run short.

The 8-32 x 1/4" set screw, called for on page 128 of the Assembly Manual, is in Pack #14 - not in Pack #13 where it should be.

There is at least one error on page 49, right column, item D, of the Arm Assembly Manual. It should be 0-0-0-1 instead of 0-0-0-4. You can verify this by checking the port addresses in the Technical Manual. Also, the Arm Kit has a replacement page for page 20 which might well be ignored. It switches the wires which control the direction of rotation of the shoulder motor. The Tech Consultants at Heath told Mike that the motor supplier has been inconsistent in the arrangement of the clockwise and counterclockwise rotation control parts of the motor. Thus, there is no way to tell which wiring instruction should be followed until it is run. Therefore, Mike's advice is to ignore, initially at least, the addendum to page 20. Wire the motor in accordance with original instructions, but keep both wires 3-3/4" long in the event they have to be switched.

Lastly, when the shoulder-head bushing is installed into the plastic arm halves, do it so that when the arm is horizontal (approximately half way through its travel), the cables are at the bottom and can easily enter the slot in the head. The head slot should have been made a half inch wider at the bottom.

When finished, the little devil performs admirably.

The Best Book I've Ever Read On CP/M

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

".... there are so many programs intended to be used by mere mortals (as opposed to programmers) that are so poorly designed that only a programmer can run them."

CP/M ASSEMBLY LANGUAGE PROGRAMMING by Ken Barbier, published by Prentice-Hall, Inc., as a Spectrum Book (I think that means it's a soft cover) is purported to be "a guide to the integrated learning of the CP/M operating system and Assembly Language programming". It is.

This could be a beginning beginner's book, because it starts with descriptions of hardware and what an operating system does, but it would help to know a little about CP/M: the structure, the basic commands, and knowledge of the Section titles in your CP/M manual, not to mention the location of the On/Off switch on your computer. It would also be nice to want to know Assembly Language, because the second half of the book is on that. Also included are some basics on using ED.COM, the line editor that comes with the CP/M operating system. For those who have already read thousands of books starting out very basically, the chapter headings are clear, and it is quite easy to skip the introductory sections without getting too lost. I originally jumped to Chapter 10, 'Preserving the User's Environment', but later went back and read the beginning chapters.

Mr. Barbier is clear, concise, and leads you by the hand through an introduction to hardware, software, CP/M organization, and writing programs in ASM that can be used as subroutines in later, more involved programs. He throws in explanations about why CP/M was constructed the way it is, in particular explaining where the TTY:, RDR:, and PUN: terms for the input/output devices came from.

"In the list of devices shown by STAT VAL: we don't find any physical device name like MOD: for modem. This is another holdover from the days when CP/M was created. It was originally programmed on an Intel Microcomputer Development System, and all the logical and physical device names shown above are part of the 'MDS Syndrome.'"

After introducing the computer system, the CP/M operating system and 8080 Assembly Language programming, the book starts on a tutorial of writing, assembling and testing a series of programs that end up as one program entitled COPY.COM, a simple file copying program which is more user oriented than PIP.COM, but not nearly so powerful.

The things I like in particular are the little bits of humor, usually corny, and the explanations of some things that have happened to me. For instance, he goes in to some detail on BDOS errors:

"If you ever see the BDOS error message

```
BDOS ERR ON R: SELECT
```

or this message with any other illegal drive specified, it means your program is totally lost, and has garbaged the DRIVE select byte at location 4 in RAM. This means that it has probably garbaged lots of other locations as well. NOW IS THE TIME TO HIT THE RESET BUTTON. You know you always wanted to!"

I do believe this explains the cryptic message I received the other day, to wit: INSERT DISK IN DRIVE P: AND HIT RETURN. (Wow, just think: drive P: would be the 16th drive on your system!)

One thing should be made clear at this point. Almost all the books I have read on software for microprocessors require that you sit down at the computer and go step by step along with them, hands fluttering over the keyboard. You can't simply sprawl by the fire, read the book and then expect to know what it is talking about. Just learning the terms requires constant use of and exposure to them. It is the same with this book. You'll get lost among things like BDOS and FCB (Basic Disk Operating System and File Control Block). But once you use them, play with them, and shed tears over them, then you will at least know what other people are talking about in careless conversation at your local computer club.

Mr. Barbier stresses two things as most important in writing a program. One is clarity in writing a program. Not only should you freely use remarks (this goes for BASIC programmers, also) but blank lines inserted between logical blocks make for a much more readable and easier to debug program. And when you read it next year, you just might be able to figure out what you were doing this year.

Second, take the time to make it usable by anyone, even the non-programmer. It's tedious, but just might make the program saleable.

"'Ergonomics' is the current buzzword referring to making your computer a friendly place to work. .... No matter what you call it, you should always try to write programs that interact with the operator in such a way that the fallible human knows what is going on, and is told what to do next."

I highly recommend this book and consider it well worth the $12.95 cover price.

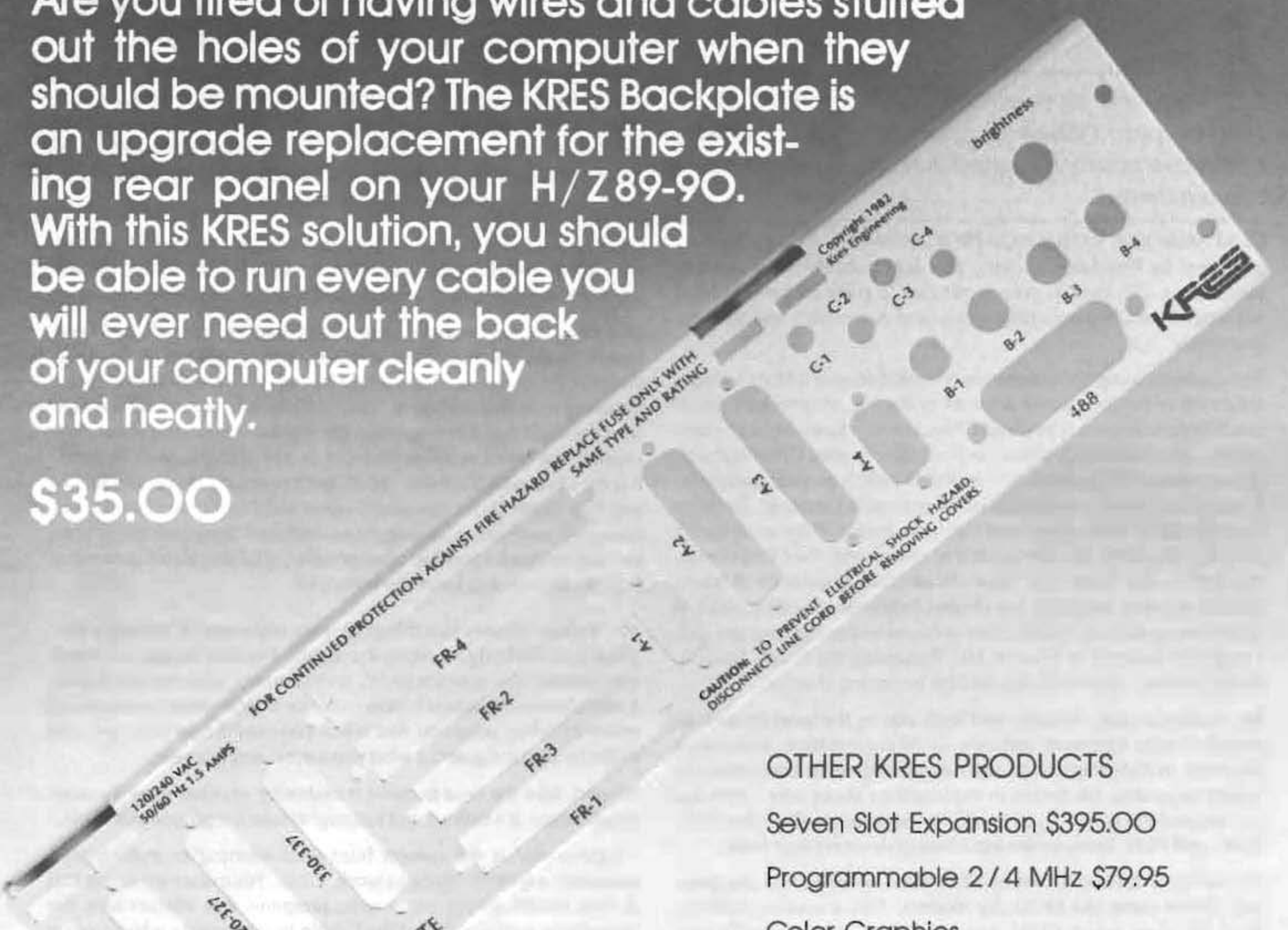
Since I will be doing more articles for REMark, I would appreciate any suggestions for subjects. Bear in mind that my computer knowledge is completely haphazard, gained through reading, taking home courses, talking with others, and a lot of late-night swearing at the people and programmers responsible for the manuals, books, and what-not required to run all but the simplest programs. I have a minimal knowledge of hardware, a pretty good knowledge of MBASIC, a smattering of ASM, and some familiarity with both HDOS and CP/M. And being a part time bookkeeper at a Heathkit Electronic Center gives me a large backup library of references. Also, if you have any questions, please don't hesitate to ask. I can always refer the hard ones to Pat Swayne or Terry Jensen.

The BACKPLATE

a neater solution from KRES

Are you tired of having wires and cables stuffed out the holes of your computer when they should be mounted? The KRES Backplate is an upgrade replacement for the existing rear panel on your H/Z89-90. With this KRES solution, you should be able to run every cable you will ever need out the back of your computer cleanly and neatly.

$35.00


brightness
Copyright 1982 Kres Engineering
C-4
C-2
C-3
C-1
B-4
B-3
B-2
B-1
488
KRES
A-4
A-3
A-2
A-1
FOR CONTINUED PROTECTION AGAINST FIRE HAZARD REPLACE FUSE ONLY WITH SAME TYPE AND RATING
CAUTION: TO PREVENT ELECTRICAL SHOCK HAZARD DISCONNECT LINE CORD BEFORE REMOVING COVERS
120/240 VAC 50/60 Hz 1.5 AMPS
FR-4
FR-2
FR-3
FR-1
330-337
320-327
340-347
DTE
DCE
DCE

OTHER KRES PRODUCTS

Seven Slot Expansion $395.00

Programmable 2/4 MHz $79.95

Color Graphics

Music Board $125.00

Prototyping Cards $39.95

Joysticks $69.95

Supplemental Fan and Bracket $49.95

Power Supply - 5 volts at 10 amps

Software - wide selection

KRES
ENGINEERING
P.O. Box 17328, Irvine, CA 92713
(714) 559-1047
(213) 957-6322
Bulletin Board: (714) 559-8579

All prices FOB Irvine, CA
California Residents add 6%

H/Z 89 is a Registered Trademark of the Heath Company

Morse Code Practice in MBASIC for the H/Z-89A

Bob Horn
Horn Engineering Assoc.
1714 Patricia Lane
Garland, TX 75042

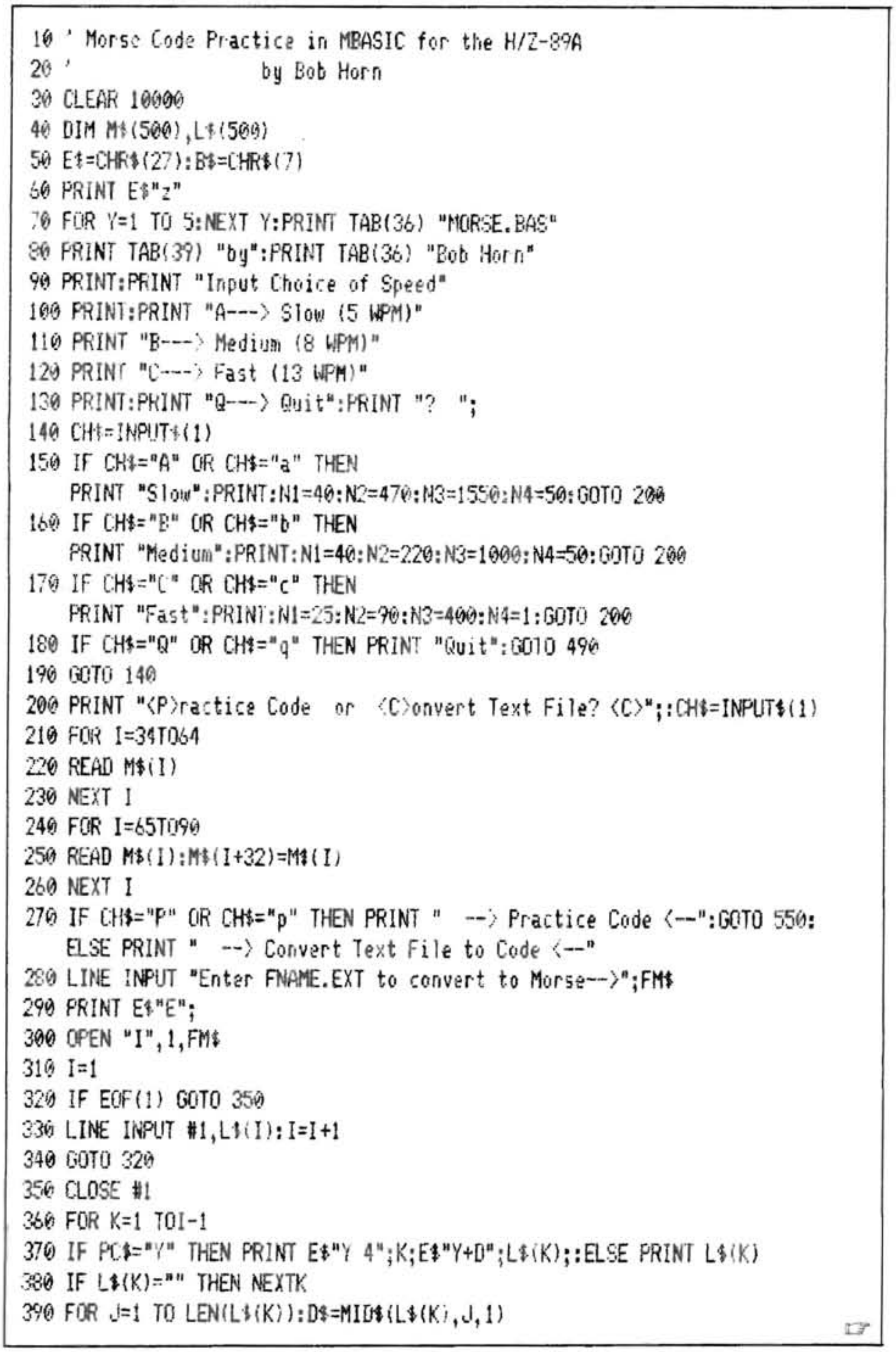
MORSE.BAS is a very effective code practice generator that uses the H/Z-89A beeper (bell) for an output. It does not require hardware mofifications to the H/Z-89A, which other morse programs usually do. MORSE.BAS runs under MicroSoft BASIC. When signing on it offers several menu choices. One can select from three speed choices. "Slow" runs at about 5WPM, "Medium" at about 8WPM, and "Fast" at about 13WPM. It will generate a specified number of random 5-letter code groups or it will run from any arbitrary text file and will handle all punctuation.

MORSE.BAS will run at such high rates on the H/Z-89A because the H/Z-89A provides a much shorter "beep" than the previous H/Z-89 models. The "beep" on the older models is so long that it is impossible to create a satisfactory "dot". However, a very simple modification will correct this problem and will provide the H/Z-89 with the same sharp, crisp "beep" that the H/Z-89A produces.

The H/Z-89 modification is to the Terminal Logic Board (the large board toward the rear of the cabinet - not the CPU board). The "beep" duration is controlled by the capacitor, C426, located near the notched end of the I.C. at U411. This chip is located at the lower left corner of the board. Change the value of this capacitor from 2.2 mF (25-221) to 1.0mF (25-197), as used in the H/Z-89A. This change should not affect the performance of the H/Z-89 in any other way.

We do not know if MORSE.BAS will run properly on the H-8 or not. Since the H-8 can be configured for full audio output, it should be possible to program it for the short duration "beep".

The experienced code operator will find the rhythm and timing of this code generator to be good at all speeds. It should provide the amateur license candidate with an effective aid to code practice, and should be useful to the experienced "Ham" in maintaining his code skills.



```
10 ' Morse Code Practice in MBASIC for the H/Z-89A
20 '                   by Bob Horn
30 CLEAR 10000
40 DIM M$(500),L$(500)
50 E$=CHR$(27):B$=CHR$(7)
60 PRINT E$"z"
70 FOR Y=1 TO 5:NEXT Y:PRINT TAB(36) "MORSE.BAS"
80 PRINT TAB(39) "by":PRINT TAB(36) "Bob Horn"
90 PRINT:PRINT "Input Choice of Speed"
100 PRINT:PRINT "A---> Slow (5 WPM)"
110 PRINT "B---> Medium (8 WPM)"
120 PRINT "C---> Fast (13 WPM)"
130 PRINT:PRINT "Q---> Quit":PRINT "?  ";
140 CH$=INPUT$(1)
150 IF CH$="A" OR CH$="a" THEN
    PRINT "Slow":PRINT:N1=40:N2=470:N3=1550:N4=50:GOTO 200
160 IF CH$="B" OR CH$="b" THEN
    PRINT "Medium":PRINT:N1=40:N2=220:N3=1000:N4=50:GOTO 200
170 IF CH$="C" OR CH$="c" THEN
    PRINT "Fast":PRINT:N1=25:N2=90:N3=400:N4=1:GOTO 200
180 IF CH$="Q" OR CH$="q" THEN PRINT "Quit":GOTO 490
190 GOTO 140
200 PRINT "<P>ractice Code  or  <C>onvert Text File? <C>";:CH$=INPUT$(1)
210 FOR I=34TO64
220 READ M$(I)
230 NEXT I
240 FOR I=65TO90
250 READ M$(I):M$(I+32)=M$(I)
260 NEXT I
270 IF CH$="P" OR CH$="p" THEN PRINT "  --> Practice Code <--":GOTO 550:
    ELSE PRINT "  --> Convert Text File to Code <--"
280 LINE INPUT "Enter FNAME.EXT to convert to Morse-->";FM$
290 PRINT E$"E";
300 OPEN "I",1,FM$
310 I=1
320 IF EOF(1) GOTO 350
330 LINE INPUT #1,L$(I):I=I+1
340 GOTO 320
350 CLOSE #1
360 FOR K=1 TOI-1
370 IF PC$="Y" THEN PRINT E$"Y 4";K;E$"Y+D";L$(K);:ELSE PRINT L$(K)
380 IF L$(K)="" THEN NEXTK
390 FOR J=1 TO LEN(L$(K)):D$=MID$(L$(K),J,1)
```

☞

```
400 D=ASC(D$)
410 FOR U=1 TO LEN(M$(D)):R$=MID$(M$(D),U,1):
   IF R$=CHR$(32) THEN FOR A=1 TO N4: NEXT A,U
420 IF R$="0" THEN PRINT B$;
430 IF R$="1" THEN FOR Y=1 TO 15:PRINT B$;:NEXT Y
440 FOR T=1 TO N1:NEXT T,U
450 FOR T=1 TO N2:NEXT T,J
460 FOR T=1 TO N3:NEXT T,K
470 IF PC$="Y" THEN GOSUB 690
480 GOTO 30
490 END
500 '               ---> Morse Code DATA Files <---
510 DATA "010010","*","0001001","*","*","011110","101101","101101",
        "*","*","110011","100001","010101","10010"

520 DATA "11111","01111","00111","00011","00001","00000","10000","11000",
       "11100","11110"

530 DATA "111000","101010","*","10001","*","1100","*"

540 DATA "01","1000","1010","100","0","0010","110","0000","00","0111","101",
        "0100","11","10","111","0110","1101","010","000","1","001","0001",
        "011","1001","1011","1100"
550 INPUT "How Many Five-Letter Words (Max. 100) ";N5
560 IF N5>100 GOTO 550
570 PRINT E$"x5";E$"E"
580 FOR I=1 TO N5
590 PRINT E$"Y#4";"Please Wait...Building Code Groups";
600 FOR T=1 TO 5
610 L=INT(91*RND(91)):IF L<65 GOTO 610
620 L$(I)=L$(I)+CHR$(L)
630 L=0:NEXT T
640 PRINT E$"Y#4";E$"p";STRING$(34," ");E$"q";
650 NEXT I
660 PRINT E$"E";E$"Y  Practice Group ---->     out of";N5
670 PC$="Y"
680 GOTO 360
690 PRINT E$"z":FOR T=1 TO 25:NEXT T:
   PRINT"Print Code Words to Disk File? <Y/N> ";
700 YN$=INPUT$(1)
710 IF YN$="N" OR YN$="n" THEN PRINT "No":RETURN
720 IF YN$<>"Y" AND YN$<>"N" AND YN$<>"y"
   AND YN$<>"n" GOTO 700
730 IF YN$="Y" OR YN$="y" THEN PRINT "Yes"
   :LINE INPUT "Enter File Name ---> ";FM$
740 OPEN "O",1,FM$
750 FOR U=1 TO I-1
760 PRINT #1,L$(U)
770 NEXT U
780 CLOSE #1
790 RETURN
```

About the Author

Bob Horn is a Jr. High School student with about a year's experience in BASIC programming, mostly self-taught. He has also contributed an animated graphics cover to Micro Media Magazine. He is currently developing an interest in C Language as a candidate for game programming. His other interests extend to music and photography. MORSE.BAS was suggested to him as an educational exercise. He took it from there, with no additional help.

Using "C" For Fast Screen Action Games

Clement S. Pepper
3270-96 Caminito East Bluff
La Jolla, CA 92037

Last spring I was working on a couple of video games using MBASIC. Interpretive BASIC has its place, but as I watched my graphics do a leisurely crawl across the screen, I told myself there had to be a better language. I could fall back on Assembly of course, except that I didn't know it too well and was not overly eager to learn. Having read Henry Fale's series on PASCAL, I was thinking seriously of it but kept putting it off. Then in early summer I learned of C/80 through our San Diego HUG, and in "C" I found precisely the high level language I was looking for. One nearly as fast as Assembly and much easier to learn and work with.

Easier to learn?! I didn't think so at the beginning, when I expended an entire weekend in a fruitless effort to write escape sequences to my H-89's screen. In time this obstacle was successfully hurdled, and considerable progress was made until I came to the need for inputting unechoed characters via the keyboard one-at-a-time without the necessity to hit RETURN. This did get me into Assembly, since the standard "getchar" available provides precisely those features I wished to avoid. There are three means by which your program can communicate with the screen: printf, putchar, and #asm. I usually use printf and putchar, but there are times when I have found an Assembly Language insert to be fastest and all around the best. (I am running with CP/M, and my Assembly Language solutions utilize CP/M's function calls, but hopefully someone with HDOS will submit a helpful solution. Other than this, everything I touch on here will be independent of the operating system used.)

I write my programs with The Software Toolworks' PIE editor. This is a full screen editor that is very easy to use, in particular, when inserting the escape sequences for terminal functions into a printf function or assembly DB statement. But to illustrate, let's get our feet wet with a direct cursor address instruction using putchar. With BASIC we would write:

```
100 PRINT CHR$(27);CHR$(89);CHR$(R);CHR$(C);"Text."
```

Where "R" and "C" are row and column assignments, respectively. We can perform the addressing functions in C by writing:

```
putchar(27); /* escape */
putchar(89); /* enable direct cursor addressing */
putchar(34); /* assume we want row 34 */
putchar(48); /* assume begin at column 48 */
```

This will direct the cursor to where we want it. But what good is an address without the text?

OK. Suppose we stay with putchar for a little while yet. This function can only direct a single character to the screen, which, should we adhere to the above format would consume a disproportionate quantity of program for the amount of text transmitted. A "while-do" might be just the ticket however. Let's try something along the lines of:

```
while((c = getchar() != EOF) putchar(c);
```

Putting all the foregoing together along with a slight variation for interest yields the short program for keyboard input and display of Listing 1. It's a while-do routine, a modification of an exercise early on in The C Programming Language, one you can play around with to get a feel for how getchar and putchar operate, though this particular example is not too useful for writing input from the keyboard. Of vital importance is the ability to employ row and column variables with the putchar function. Observe that ra and rb are integers and as such are not enclosed in apostrophes. This approach is hardly a method of choice for screen action inputs. Nor is it productive in writing to the screen from within your program. But, putchar is an important function, the one I employ for most of my cursor addressing needs in combination with printf.

A C program called printf.c is provided as an entity on the C/80 disk. It must be included in your program by writing #include"printf.c" before your first call on its function.

You can employ printf in some very imaginative ways, and it can save a lot of space in your program listing when doing so. To see what I mean, take a good look at the short routine of Listing 2. Four operations are carried out in the initial statement: clearing the screen, turning off the cursor, enabling the 25th line, and enabling direct cursor addressing. Now, it would be great to continue on in this manner with the row and column variables, but to date I have not found a way to do it. That is, a printf statement of the form:

```
printf("[E[x5[x1[Y5648[p C IS THE GREATEST [q[y5");
```

will print the legend, but not on the 25th line. Try it for yourself.

```
printf("[E[x5[x1[Y%d%d[p C IS THE GREATEST [q[y5",ra,rb);
```

performs similarly. Note that Listing 2 will not disable the 25th line. While this could have been included as well, you would only have seen a brief flash across the bottom of your screen for your efforts.

What, you are no doubt wondering, is the meaning of all those ['s. With PIE, escapes are entered by typing Ctrl-K followed by [and the required character. The [appears on the screen in reverse video. Escapes are non-printing characters that in some instances can really louse up your printer's control, so I go through a procedure of what I call "turning off the lights" before using PIP, Ctrl-P TYPE, or TEXT to print my programs. But you can see where a single printf statement replaces a multitude of putchar's.

It is possible to insert variables in printf statements. The %d's in the expression earlier mean to print the variable it refers to as a digital value. %s will print a string. These are well explained in the references so I'll not elaborate further. Nor is it always necessary to declare the variables as static. In fact, in the larger programs I have been working on, a major proportion of the variables end up being declared as globals. When variables are declared static their initial values are as defined, but after the first pass through the values will change as required by your program.

An understanding of what can be done with printf and putchar is essential to the writing of good screen action routines. Screen motion requires directing the cursor to the required location, drawing the object, displaying it for a short time, erasing it, making some sort of change to the row and/or column variables, directing the cursor to the new location, and repeating the sequence. Additional requirements will exist in detection of the screen boundaries, and, for many games, the detection of an interfering object or an intersection, such as a missile hit that must be followed by an alternative course of action. The activity displayed on the screen typically originates in the program, but often must be subject to modification by the user through the keyboard. I have found working with C to be quite enjoyable with its extensive list of operators such as for, while, and if-else controls, and its straightforward writing structure.

One more comment, this is on the editor you are using before launching into an example program. When an object is moved on the screen it must first be erased, then redrawn at the next location. A convenient way to erase is to print spaces over the object. If your editor substitutes tabs for spaces you will observe some unanticipated trailings of graphic fragments streaking your screen. With PIE you can make a copy just for writing C programs and delete the tab feature with a patch. I renamed the copy PIE1 to keep track of it. Using PIE1, I have had no problem with object blanking.

```
/*  PUTTST.C  10-23-82  A C/80 putchar() illustration program.  */
#define EOF    -1

main() {
     char c;
     putchar(27);     /* escape */
     putchar('E');    /* clear screen */
     putchar(27);
     putchar(89);     /* enable direct cursor addressing. */
     putchar(34);     /* row value */
     putchar(42);     /* column value */
     while((c = getchar()) != EOF) replay(c);  /* read the keyboard */
}
replay(c) {          /* display keyboard input in a different format */
     static int ra =34, rb = 42;
     if(c == '\n') { ra = 34; rb = 42; }
     putchar(27);
     putchar('Y');    /* alternate expression for cursor addr enable */
     putchar(ra);     /* row variable */
     putchar(rb);     /* column variable */
     putchar(c);      /* display character read from keyboard */
     ra += 1;  rb +=1;  /* increment variables */
}
```

Listing 1. When you run this program, the cursor will wait at the upper left corner of the screen for your typed input. Type a few characters, then hit return. To exit, type Ctrl-Z (CP/M) or Ctrl-D (HDOS).

As a vehicle for exploration I wrote a short program which I named BOX. It can be found in Listing 3. BOX is just that, a rectangle carrying a label which moves freely on its own around the screen. Remember the early PONG game? The four cursor keys on the keypad allow you to redirect the box. The HOME key will stop its motion completely. In the lower left corner of the screen you will find a pea shooter. Peas can be fired at the box from the keyboard by pressing the D or F keys. Hits are recorded on the 25th line. Keying in Q returns you to the operating system.

Figure 1 is a flowchart for BOX. In some respects it is unconventional, but I think you will find it pretty straightforward when it comes to describing the program. BOX consists of two sections: a setup that declares global variables, and MAIN which performs one-time only needs. PSHUTR is a small blockhouse sort of structure in the lower left corner from which you fire peas at the box. BANNER is a fixed legend that the points are entered in as you make hits on the box with your pea shooting. The program then moves on to RUN, a short looping segment supporting the subroutines that provide the program's dynamics.

The first function call under RUN is MOVBOX. In the absence of any external input, MOVBOX continues to advance the box's vertical and horizontal movement, bouncing it off the screen boundaries much as

```
/*  PUTTSTA.C  10-23-82  A printf screen printing test program    */
#include"printf.c"

main() {
     printf("[E[x5[x1[Y"); /* clr scrn, cursor off, en 25th, en cur add */
    putchar(56);           /* row value */
    putchar(48);           /* column value */
     printf("[p C IS THE GREATEST [q[y5"); }    /* print messag, cur on */

/* Note: [ is escape representation. If using PIE will be reverse video. */
```

Listing 2. When run, PUTTSTA.COM prints "C is the Greatest" in reverse video on the 25th line.

PONG waiting for another quarter.

You will notice a couple of delays in the program, one for the pea as it streaks across your screen, and the other for each drawing of the box. You might try deleting these to see what the effect is. It appears that C can put the pieces together faster than the terminal can draw them on the screen, and without the delays the figures acquire a fragmented appearance. In fact, the shell will not be seen on every line.

KEYINP reads your keyboard/keypad input. The three decision blocks that follow define the program's response to your instruction. If you have typed HOME or one of the cursor keys on the pad, you have signaled for a new box direction and the program returns to BOXMOV. If you have pressed the D or F key, then PEAFYR launches a missile at the box, tests for a box hit, or failing that, the screen boundary, or simply none of these, and takes appropriate action, including the scoring should you have made a hit. In time, you tire of the game and type Q. DONE then resets the terminal and calls EXIT for a return to the operating system.

While BOX is certainly elementary as games go, I think it covers most of the aspects of a more elaborate game. It also contains a bug which to date I have not found a solution for. The bug is simply this: on the very first keyboard/pad input there will be no response. On the second and succeeding inputs the program will respond exactly as it should. It has something to do with the way I am using CP/M function 6 I am sure. If anyone finds a cure for this, I hope they will mail it in to REMark. The same for anyone constructing a suitable scheme for HDOS keyboard/pad input.

Not too many of us are going to write a successor to PACMAN. On the other hand, there is a pleasure to be found in the playing of a game of your own design that can not be purchased at any price. You may discover this for yourself by adding additional angles of fire to the peashooter, or putting in the routine for an airplane, say, that materializes on the screen under certain conditions to launch missiles at the box. Then you can work at shooting down the plane with the peashooter or a rocket launcher of another devising before the missile gets the box.

The possibilities are endless.

Here ar some references to the C Language that I have found to be of value.

1. Bilofsky, Walt. **"Manual for C/80 Version 2.0"**. February, 1982. The Software Toolworks, 14478 Glorietta Dr., Sherman Oaks, CA. (The C language the writer is using with his Heath H89 computer.)

2. Ritchie, D.M. and Kernighan, B.W. **"The C Programming Language"**. Prentice-Hall, Inc., Englewood Cliffs, NJ 07632. (The Bible of C. Don't try C without it.)

3. **"Programming in VAX-11 C"**. AA-L370A-TE, Software Version V1.0, 1982 Digital Equipment Corporation, P.O.Box CS2008, Nashua, New Hampshire. (Included even though it goes far beyond C/80 because the first six chapters are quite general and very, very readable.)

4. Cain, Ron. **"A Small Computer For the 8080's"**. Dr. Dobbs Journal, May 1945 pp 5-19. (A very readable description by Ron of how he came to write the compiler later expanded by Walt Bilofsky and released as C/80.)

5. Gewirtz, David A. **"An Introduction to the C Programming Language"**. Microsystems NOV/DEC 1981, pp 20-38. (I found this article really helpful in getting started.)

6. Lindsay, Jon. **"Introduction to CP/M Assembly Language"**. Executive Computer, P.O.Box 222178, Carmel, CA 93922. (While not a C reference, this readable book guided me in my initial construction of #asm routines. Strictly CP/M.)

Listing 3.

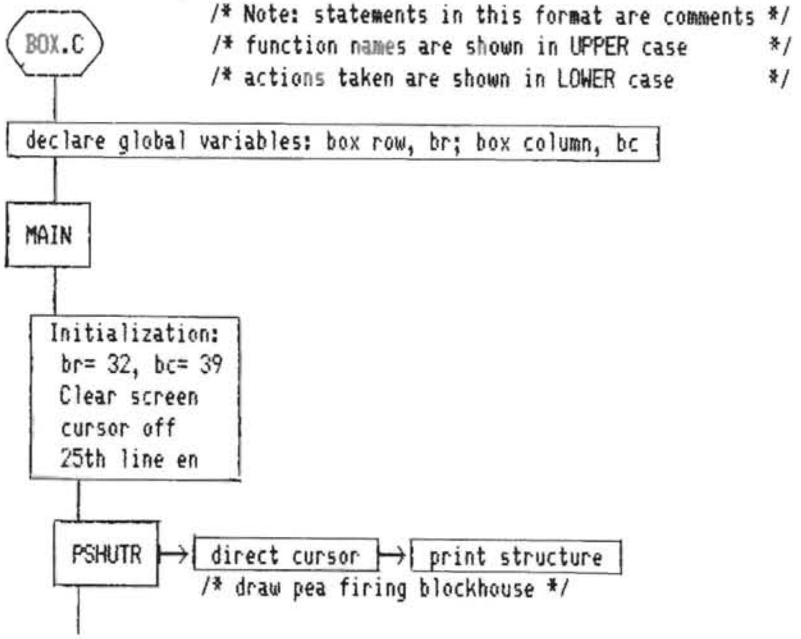
```
/* BOX.C       C.S.Pepper      10-24-82 */
/* created to explore terminal graphics and screen positioning */
/* using The Software Toolworks C/80 Version 2.0 */
#include"printf.c"

  int br, bc;                  /* br = box row, bc = box column */

main()
{
    br = 32, bc = 39;          /* initial box coordinates      */
       printf("[E[x5[x1");     /* clr scrn, cursr off, 25th on */
         pshutr();             /* draw the pea shooter   */
         banner();             /* 25th line legend entry */
         drwbox();             /* initial box draw       */
            run();             /* begin program sequence */
}
run()
{
               boxmov();       /* increment box screen position */
               keyinp();       /* act on keyboard input, if any */
               run();          /* continue looping              */
}
drwbox()  /* draw the box */
{
   curadd(br,bc);
     printf("[k[Ffaaac[k[B[j'BOX'[k[Beaaad[G");
}
blkbox()  /* erase the box */
{
   curadd(br,bc);
     printf("[k    [k[B[j    [k[B    ");
}
curadd(r,c)  /* set cursor to address and save */
{
           printf("[Y");
          putchar(r);
          putchar(c);
           printf("[j");
}
keyinp() /* keyboard routine for box direction control and pea firing */
{
```

☞

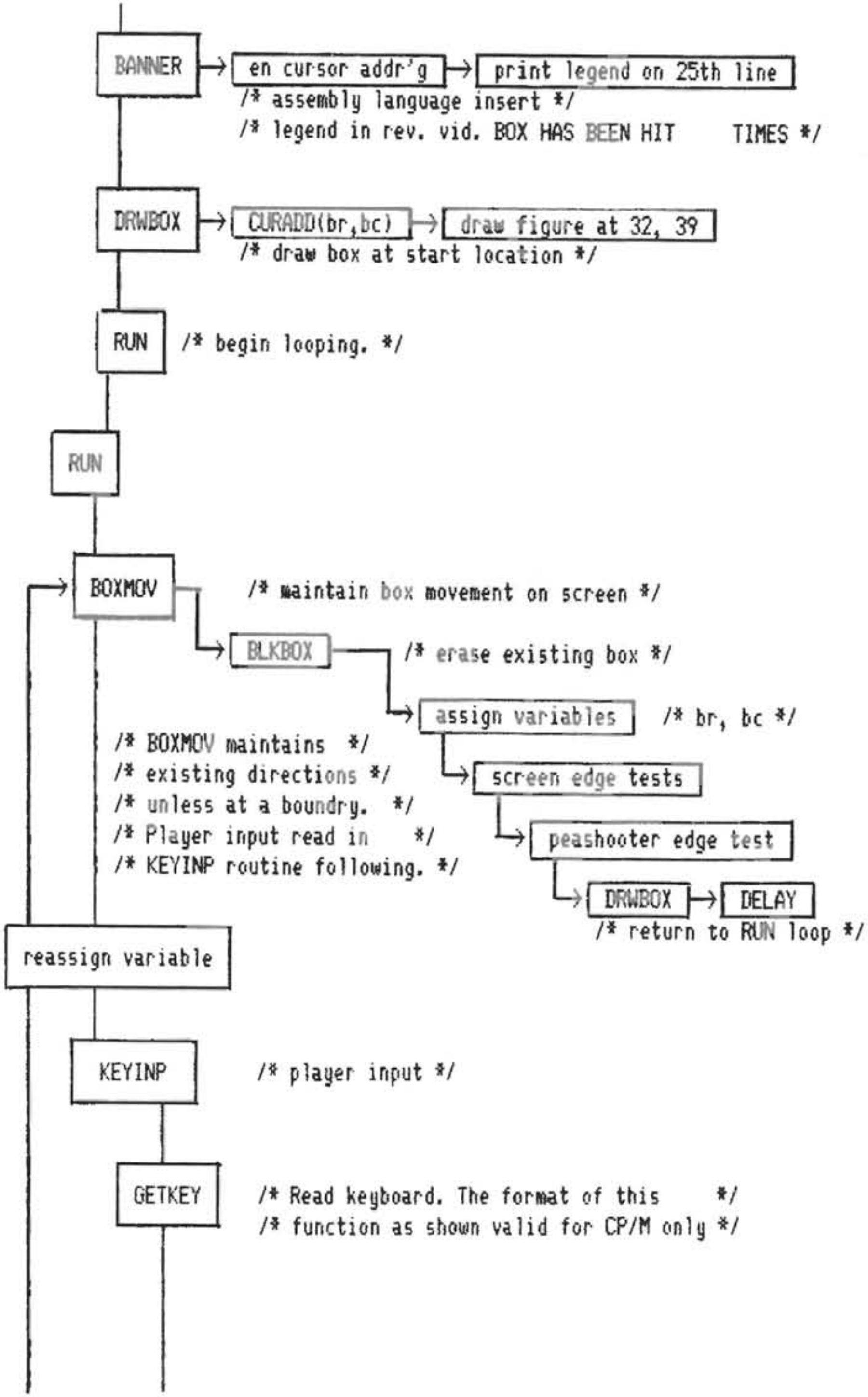
```
      static int bn = 0, sn = 0;
         char d;   /* d = direction variable, 4 or 6, or Q for quit. */
            while(d = getkey())
              switch(d) {
                case '4': { bn = 4; boxmov(bn); break; }
                case '5': { bn = 5; boxmov(bn); break; }
                case '6': { bn = 6; boxmov(bn); break; }
                case '8': { bn = 8; boxmov(bn); break; }
                case '2': { bn = 2; boxmov(bn); break; }
                case 'F': { sn = 1; peafyr(sn); break; }
                case 'D': { sn = 2; peafyr(sn); break; }
                case 'Q': done();                          }
}
boxmov(bn)   /* apply keyed input to box position */
{
   int bci, bri;
   static int bcn = 6, brn = 2;
    blkbox();
      if(bn == 4 || bn == 5 || bn == 6) bcn = bn;              /* var xfer */
      if(bn == 8 || bn == 2) brn = bn;                         /* var xfer */
      if(bcn == 4) bci = -1;                                  /* move left  */
 else if(bcn == 5) { bci = 0; bcn = 0; bri = 0; brn = 0; }/* all stop   */
 else if(bcn == 6) bci = 1;                                   /* move right */
      if(brn == 8) bri = -1;                                  /* move up    */
 else if(brn == 2) bri = 1;                                   /* move down  */
      if(bc == 33) { bci = 1; bcn = 6; }                    /* left edge rev  */
      if(bc == 105) { bci = -1; bcn = 4; }                  /* right edge rev */
      if(br == 33) { bri = 1; brn = 2; }                    /* top edge rev   */
      if(br == 53) { bri = -1; brn = 8; }                   /* low edge rev   */
      if(br >= 50 && bc <= 38) { bri = -1; brn = 8; bci = 1; bcn = 6; }
        bc = bc + bci; br = br + bri;                 /* ^ avoid peashooter */
          drwbox(); delay3();
}
peafyr(sn) /* manages projectile firing from initiation to completion */
{
     static int bh = 0;
     int pr, pc, pcn;
        if(sn == 1) pcn = 2;     /* two column lead angle */
   else if(sn == 2) pcn = 1;     /* one column lead angle */
        if(sn != 0) { pr = 53;  pc = 36; sn = 0; }
        curadd(pr,pc); drwpea(); delay1(); blkpea();
          if(pr >= br && pr <= br + 2 && pc >= bc && pc <= bc + 4) {
            burst(pr,pc);
              bh += 1;
               curadd(56,62); printf("%d",bh); }
```

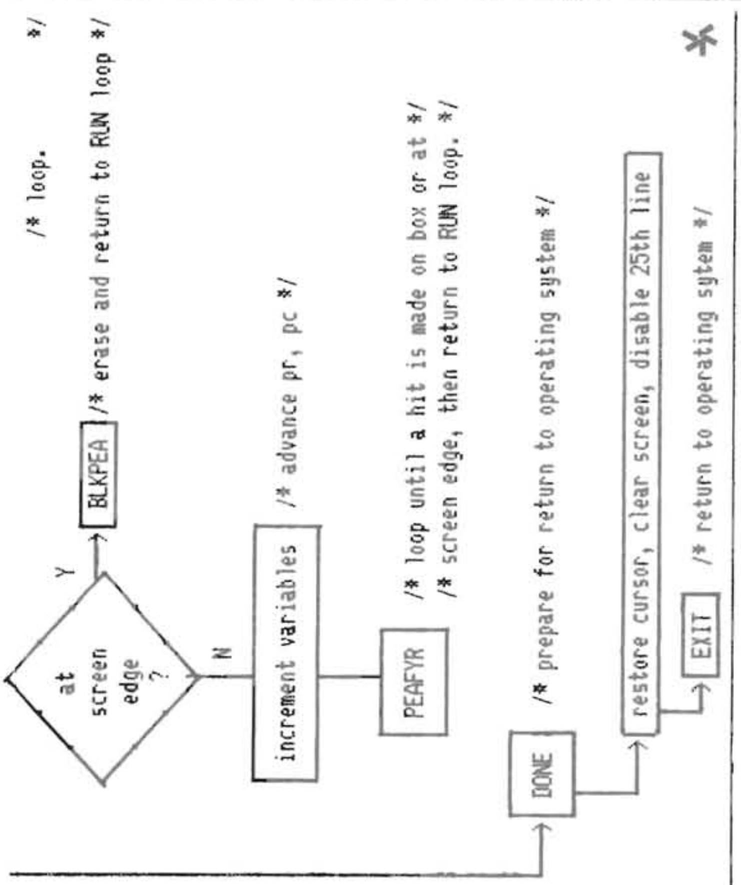
```
        printf("[Y"); putchar(54); putchar(34); printf("[j");
        printf("[k[F[p  _[k[B   [q[G");
}
banner()  /* draw the fixed portion of the 25th line */
{
#asm
BDINDS  CALL    CURSBS
        LXI     D,BXSCR
        MVI     C,9
        CALL    5
        RET
CURSBS  LXI     D,CURSBX
        MVI     C,9
        CALL    5
        RET
        ;
CURSBX  DB      27,'Y',56,32,27,'j$'
        ;
BXSCR   DB      '[k           [p BOX HAS BEEN HIT [q  0  [p TIMES [q$'
#endasm
}
```

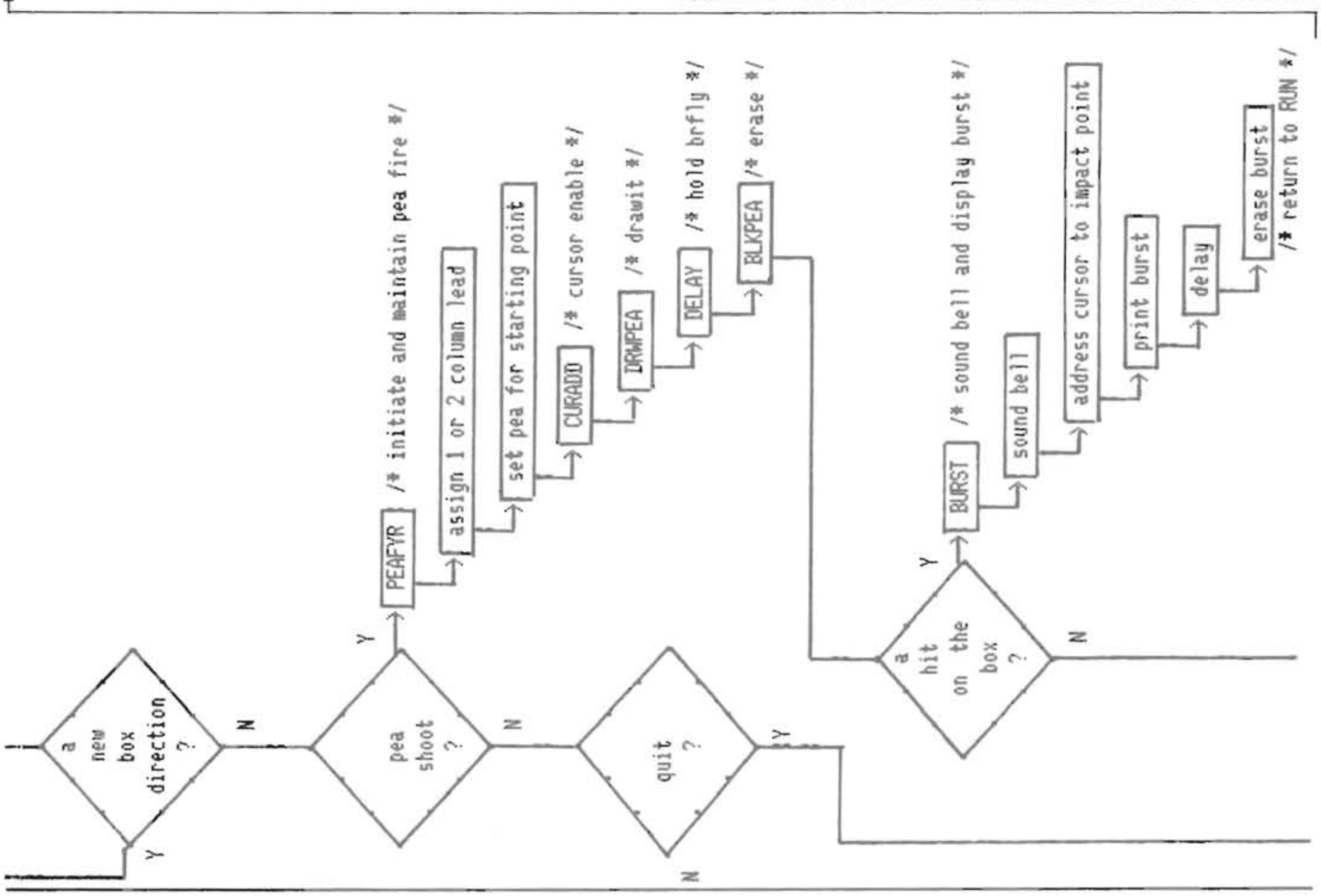
Figure 1.
Flowchart for BOX program

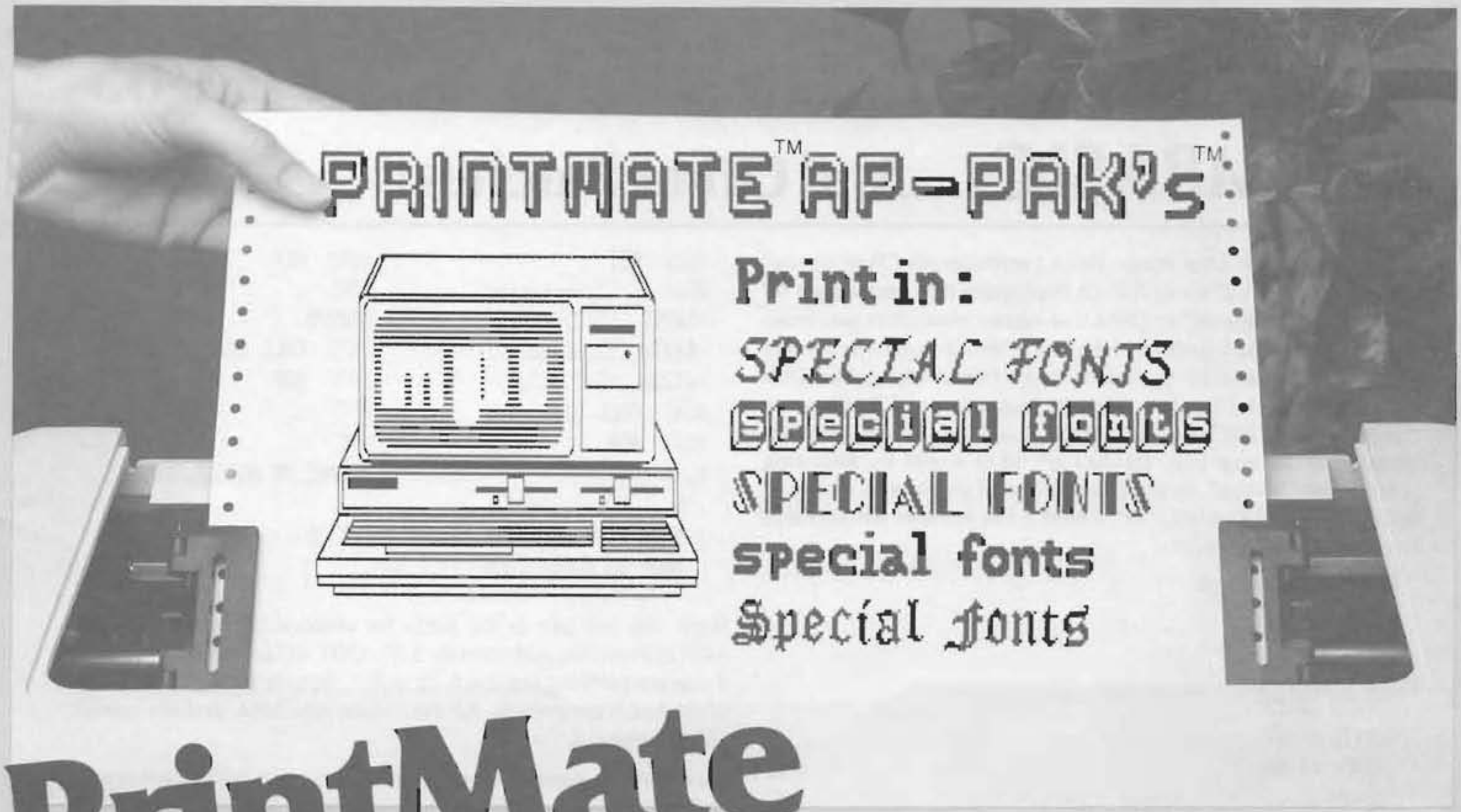

/* Note: statements in this format are comments */
/* function names are shown in UPPER case */
/* actions taken are shown in LOWER case */
BOX.C
declare global variables: box row, br; box column, bc
MAIN
Initialization:
br= 32, bc= 39
Clear screen
cursor off
25th line en
PSHUTR
direct cursor
print structure
/* draw pea firing blockhouse */

```
     else if(pr <= 32 ¦¦ pc >= 110) blkpea();
     else { pr -= 1; pc += pcn; peafyr(sn); }
}
drwpea()  /* draw the projectile */
{
     printf("[k[F^[G");
}
blkpea()  /* blank the projectile */
{
          printf("[k ");
}
burst(sr,sc)  /* display pea burst on box */
{
        printf("\07");
        printf("\07");
        printf("[Y"); putchar(sr); putchar(sc); printf("[j");
        printf("[k\\¦*¦/[k[B/¦*¦\\");
        delay3();
        printf("[k     [k[B     ");
}
delay3()  /* delay of about 30 mS */
{
    int i;
        for(i=0; i < 300; i++) ;
}
delay1()  /* delay of about 10 mS */
{
    int i;
        for(i=0; i < 100; i++) ;
}
getkey()  /* CP/M function 6 input */
{
#asm
        MVI     E,0FFH
        MVI     C,6
        CALL    5
#endasm
}
done()  /* return to operating system */
{
        printf("[y5[E[y1");  /* cursor top lft, clr scrn, 25th off */
               exit();       /* returns control to CP/M */
}
pshutr()  /* draw pea projector */
{
```


BANNER
en cursor addr'g
print legend on 25th line
/* assembly language insert */
/* legend in rev. vid. BOX HAS BEEN HIT   TIMES */
DRWBOX
CURADD(br,bc)
draw figure at 32, 39
/* draw box at start location */
RUN
/* begin looping. */
RUN
BOXMOV
/* maintain box movement on screen */
BLKBOX
/* erase existing box */
assign variables
/* br, bc */
/* BOXMOV maintains */
/* existing directions */
/* unless at a boundry. */
/* Player input read in */
/* KEYINP routine following. */
screen edge tests
peashooter edge test
DRWBOX
DELAY
/* return to RUN loop */
reassign variable
KEYINP
/* player input */
GETKEY
/* Read keyboard. The format of this */
/* function as shown valid for CP/M only */


/* loop. */
at screen edge ?
Y
BLKPEA
/* erase and return to RUN loop */
N
increment variables
/* advance pr, pc */
PEAFYR
/* loop until a hit is made on box or at */
/* screen edge, then return to RUN loop. */
DONE
/* prepare for return to operating system */
restore cursor, clear screen, disable 25th line
EXIT
/* return to operating sytem */


a new box direction ?
Y
N
pea shoot ?
Y
PEAFYR
/* initiate and maintain pea fire */
assign 1 or 2 column lead
set pea for starting point
CURADD
/* cursor enable */
DRWPEA
/* drawit */
DELAY
/* hold brfly */
BLKPEA
/* erase */
N
quit ?
Y
N
a hit on the box ?
Y
BURST
/* sound bell and display burst */
sound bell
address cursor to impact point
print burst
delay
erase burst
/* return to RUN */
N



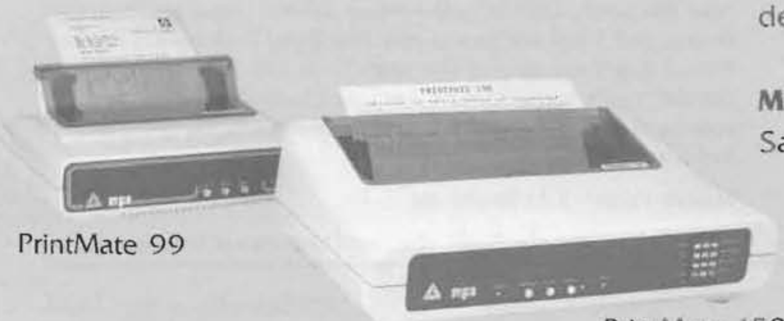
PrintMate GOES GRAPHIC WITH H/Z100!

Picture this—your logos, graphs and fancy print fonts on paper with a PrintMate. Our exclusive AP-PAK lets you print what you picture. In just one keystroke, you can send what's on the screen on to your printer. With an AP-PAK, you can intermix text and graphics on the same line and print in characters up to ⅝-inch high. AP-PAK runs with the Z-DOS™ (MS-DOS) BIOS release 1.00 (Winchester) version 1.10 and above.

With friction and tractor feed standard on the 100 cps PrintMate 99, virtually any form goes through without a hitch...from labels to multi-part. The PrintMate 99's optional single-sheet feeder lets you run single sheets through the front of the machine, thus simplifying all your personal printing. And its 1K (optional 2K) buffer provides additional printer performance.

Our premier performer for home or office is the PrintMate 150, featuring 150 cps print speed, three paper paths and 15-inch carriage. Expandable buffers (up to 68K!) dramatically increase flexibility. And with SoftSwitch,™ you get an impressive range of control at your fingertips right on the front panel of the printer.

No matter how you compute, you get more print versatility for the price with PrintMate: correspondence-quality printing, superb graphics capabilities and versatile paper handling. At MPI, we want to see you in print. See your dealer today for a demonstration.



PrintMate 99

PrintMate 150

Micro Peripherals, Inc., 4426 South Century Drive,
Salt Lake City, Utah 84107 **1-800-821-8848**

The Printer People™
mpi

H/Z 100 is a registered trademark of the Heath Company.
Z-DOS is a trademark of the Zenith Data Systems Corp.

Get Rid Of "Echo On Delete" In CP/M-85 And MBASIC (...And Other Patches)

Pat Swayne
Software Engineer

One of the nice little things about Heath/Zenith CP/M version 2.2.03 is that part of the CONFIGUR program that lets you get rid of the #$%&! "feature" in CP/M that causes characters you erase with the DELETE key (instead of the BACK SPACE key) to be echoed on the screen instead of removed from it. Unfortunately, the CONFIGUR program for the new CP/M-85 that runs on H/Z-100 series computers does not have that option. If you are like me, and that "feature" really bugs you, you can get rid of it with the following patch to MVCPM207. In this example, what you type is shown in bold print. In all patches, "xx" means a hex number whose value is unimportant to the patch.

```
A>DDT MVCPM207.COM
NEXT  PC
2B00 0100
-S147D
  147D CA C3
  147E EF 07
  147F 09 0A
  1430 xx .             (Type a period.)
-^C                     (Control-C)
A>SAVE 42 MVCPM207.COM
```

After you make this patch, run MVCPM207, then run SYSGEN. When SYSGEN asks you for a source drive, just hit RETURN. When it asks for a destination drive, type A. When it asks for a destination drive a second time, hit RETURN. Re-boot the computer, and your DELETE key will now cause the removal of characters from the screen instead of echoing them. This patch affects only the line input function of the BDOS (function 10). Other functions will return the delete character unchanged. This patch duplicates what CONFIGUR does for CP/M 2.2.03 when you suppress echo on delete except that it does it to the BDOS image in MVCPM207 instead of the actual BDOS on the disk.

Microsoft BASIC processes line input within itself, so the patch above does not fix the problem with MBASIC. You can fix it with the following patch. Two versions of the patch are presented, for the 5.2x versions, and version 4.83 (Heath's version of OBASIC). Be sure you have a backup copy of the version you are patching.

```
Version 5.21, 5.22, 5.2          Version 4.83

A>DDT MBASIC.COM                 A>DDT OBASIC.COM
DDT VERS 2.2                     DDT VERS 2.2
NEXT  PC                         NEXT  PC
6100 0100                        4F00 0100
-A616                            -A5E3
0616  MOV D,B                    05E3  MOV D,B
0617  DAD B                      05E4  DAD B
0618  ANI 7F                     05E5  ANI 7F
061A  CPI 7F                     05E7  CPI 7F
061C  JNZ 621                    05E9  JNZ 5EE
061F  MVI A,8                    05EC  MVI A,8
0621  CPI 0F                     05EE  CPI 0F
```

```
0622  RET                        05F0  RET
0624  . (Type period)            05F1  .
-A4246  (5.21 only)              -A397B
-A4212  (5.22 only)              397B  CALL 5E5
-A422E  (5.2 only)               397E  NOP
4246  CALL 618                   397F  .
4249  NOP                        -^C
424A  .                          A>SAVE 78 OBASIC.COM
-^C    (Control-C)
A>SAVE 96 MBASIC.COM  (5.21 and 5.22)
A>SAVE 95 MBASIC.COM  (5.2 only)
```

Note: The last part of the patch for versions 5.22 and 5.2 (after A4212) shows the addresses for 5.21. DDT will print other addresses if you are patching version 5.22 or 5.2. Be sure you enter only one of the last A commands, for the version you have, and the correct SAVE command.

This patch shortens an obscure error message ("Unprintable error") to "UP", and uses the freed space as a patch area. The patch itself checks input characters for delete (7FH), and replaces any delete characters with back space characters. If The "Unprintable error" should ever occur, the letters "UP" will be printed followed by some extraneous characters, which is the patch itself being printed as characters.

An additional patch is required for OBASIC if you want to remove the deleted character from the screen instead of just backspacing under it. This patch shortens the error message "Direct statement in file" to "DS", and moves the next error message (which is the last one) down, and uses the freed space for the patch.

```
A>DDT OBASIC.COM                 06F9  MVI A,8
DDT VERS 2.2                     06FB  CALL 3933
NEXT  PC                         06FE  POP PSW
4F00 0100                        06FF  RET
-A6DF                            0700  .
06DF  MOV D,E                    -A41CD
06E0  NOP                        41CD  JZ 41D6
06E1  .                          41D0  CALL 6F0
-M6F7,705,6E1                    41D3  JMP 418C
-A6F0                            41D6  CALL 39A8
06F0  CALL 3886                  41D9  .
06F3  PUSH PSW                   -^C
06F4  MVI A,20                   A>SAVE 78 OBASIC.COM
06F6  CALL 3933
```

After this patch, OBASIC will remove deleted characters from the screen, and it will not go to a new line if you back up too far, but instead it will just stop. If you want to do this patch and the first OBASIC patch in one session, omit the Control-C and the "SAVE" command from the first patch, and go directly to the line "A6DF" in this patch.

Making Version 5.22 Re-entrant

All of the versions of MBASIC discussed here except 5.22 are re- entrant. What that means is if you should somehow be forced back into CP/M (for example, because of a BDOS error) in the middle of running or developing a program, you can get back into MBASIC with your program intact by jumping back to the TPA (Transient Program Area). To make version 5.22 re-entrant, enter this patch.

```
A>DDT MBASIC.COM
DDT VERS 2.2
NEXT  PC
6100 0100
-A5DC6
5DC6  LXI H,C7F
5DC9  SHLD 101
5DCC  .
-^C
A>SAVE 96 MBASIC.COM
```

To re-enter MBASIC (or any other re-enterable program), you need to first create a dummy .COM file as follows.

```
A>SAVE 0 GO.COM
```

Now, all you have to do to re-enter a program is type GO. To test this procedure, run MBASIC, load a BASIC program, and enter SYSTEM. When the A> prompt appears, type GO, and the MBASIC's OK prompt will be printed. Now, type LIST to verify that your program is still intact.

Allowing Underlines

MBASIC uses the underline character (_) as an alternate for the delete or back space keys, and because of this, it does not allow you to use it in PRINT or INPUT statements (such as PRINT "_"). The following patches will alter MBASIC so that underlines can be used in PRINT and INPUT statements.

```
Versions 5.2, 5.21, 5.22                   Version 4.83
A>DDT MBASIC.COM                           A>DDT OBASIC.COM
DDT VERS 2.2                               DDT VERS 2.2
NEXT  PC                                   NEXT  PC
6100 0100                                  4F00 0100
-S3E99    (5.2 only)                       -S422D
-S3EAF    (5.21 only)                      422D CA 0
-S3E78    (5.22 only)                      422E CB 0
3E99 C2 C3                                 422F 41 0
3E9A xx .                                  4230 xx .
-^C                                        -^C
A>SAVE 96 MBASIC.COM (5.21 and 5.22)       A>SAVE 78 OBASIC.COM
A>SAVE 95 MBASIC.COM (5.2 only)
```

YOU ARE INVITED TO THE

SECOND

NATIONAL HUG

CONFERENCE

see details on page 7 of this issue

A Look At The FAC

(Floating Point Accumulator)

Herbert A. Friedman, MD
1922 Danube Way
Upland, CA 91786

The Heath MBASIC Manual refers to the Floating Point Accumulator (FAC) as the "normal" way to pass arguments between a BASIC program and an Assembly Language subroutine. An attempt to access the FAC with PEEK statements seems to show its contents were immediately reset. This program uses an Assembly Language subroutine and the MBASIC USR function to set up the FAC for the proper argument and then read it before it is reset.

The program suggests comparison to the way MBASIC stores its variables. The order of significance differs, for one thing. The readout is given in HEXADECIMAL, but can easily be modified to display octal or decimal.

```
********************************************
***     Author: Herbert A. Friedman, M.D.     ***
***                                           ***
********************************************
        ORG     8C00H   *** 8C00H=35840D
START   SHLD    8C30H   Get FAC-3 Pointer
        PUSH    D       Save string descriptor pointer
        XCHG
        SHLD    8C32H   Copy DE contents
        STA     8C34H   Save Type byte
        LXI     H,8C35H Dump reception area
        XCHG             and put it in DE
        INX     H       Point to FAC-2
        INX     H             to FAC-1
        INX     H             to FAC
        MVI     B,08H   Load counter
LOOP1   MOV     A,M     Get data
        STAX    D       Store it
        DCX     H
        INX     D
        DCR     B
        JNZ     LOOP1   Done?
        XCHG            Save DE in HL
        POP     D
        XCHG
        MVI     B,03H   New count for strings
LOOP2   MOV     A,M
        STAX    D
        INX     H
        INX     D
        DCR     B
        JNZ     LOOP2   Done. Should have used 16 bytes.
        RET
        END     START
```

☞

```
100 REM 'FACDUMP.BAS' Dump & display values stored in FLOATING POINT ACCUMULATOR
110 CLEAR 100,35839!(35940!)    ' Reserve memory space for machine program
120 DATA &H22,&H30,&H8C,&HD5,&HEB,&H22,&H32,&H8C,&H32,&H34,&H8C,&H21,&H35,&H8C
130 DATA &HEB,&H23,&H23,&H23,&H06,&H08,&H7E,&H12,&H2B,&H13,&H05,&HC2,&H14,&H8C
140 DATA &HEB,&HD1,&HEB,&H06,&H03,&H7E,&H12,&H23,&H13,&H05,&HC2,&H21,&H8C,&HC9
150 FOR I=0 TO 41: READ D: POKE 35840!+I,D: NEXT I      'POKE into memory
160 DEF USR=&H8C00
200                             'Define strings & clear screen
210 ES$=CHR$(27): RV$=ES$+"p": NV$=ES$+"q": PRINT ES$+"E"
220 PRINT"THIS PROGRAM demonstrates the existence of the FLOATING POINT ACCUMULATOR"
230 PRINT"  or FAC. It's hidden from the BASIC user but may be accessed by a DUMP."
250 PRINT: INPUT "      DO YOU WISH to enter a NUMBER or a STRING (N/S)";N$
260 IF N$="S" THEN 400
270 IF N$<>"N" THEN 250
300 PRINT "ENTER numbers in the following way:" ' Number routine
310 PRINT"              INTEGERS%               Example: 456%"
320 PRINT"              SINGLE PRECISION REALS! Example: 23.456!"
330 PRINT"              DOUBLE PRECISION REALS# Example: -453.56745654#"
340 INPUT "                             Your entry, please: ";N$
350 IF RIGHT$(N$,1)="%" THEN N%=VAL(N$): N1%=USR (N%): GOTO 500 ' Dummy argument
360 IF RIGHT$(N$,1)="!" THEN N!=VAL(N$): N1!=USR (N!): GOTO 500
370 IF RIGHT$(N$,1)="#" THEN N#=VAL(N$): N1#=USR (N#): GOTO 500
380 GOTO 300
400 INPUT "             ENTER the string, please";S$   ' String routine
410 S1$=USR (S$)                ' Dummy argument
500 PRINT:PRINT                 ' Display FAC & its contents
510 PRINT "     MSB--------------------> LSBs ------------------> LSB"
520 PRINT RV$;"FAC          FAC-1   FAC-2   FAC-3   FAC-4   FAC-5   FAC-6   FAC-7";NV$
530 FOR J%=0 TO 7
540     F%=PEEK((&H8C35)+J%): F$=HEX$(F%): PRINT F$;"   ";
550 NEXT J%: PRINT
560 PRINT"  NOTES: 1. FAC contains exponent in 'excess 80' form. (Subtract 80H)."
570 PRINT"         2. Bit 7 of FAC-1 is the sign bit."
580 PRINT: PRINT"      ********   ALL    VALUES   IN   HEXADECIMAL   ********"
600 PRINT                       ' Display registers
610 H=256*PEEK(&H8C31): L=PEEK(&H8C30): HL=H+L: HL$=HEX$(HL)
620 D=256*PEEK(&H8C33): E=PEEK(&H8C32): DE=D+E: DE$=HEX$(DE)
630 A=PEEK(&H8C34):A$=HEX$(A)
640 PRINT"Register HL contains: ";HL$;".  It always points to FAC-3."
650 PRINT"Register DE contains: ";DE$;".  It points to the string descriptor."
660 PRINT"Register A  contains: ";A$;".    The  type of argument in the FAC, is:"
670 GOSUB 1000
680 IF VAL(A$)<>3 THEN 800
700 PRINT                       ' Display string descriptor
710 SL=PEEK(&H8C3D): SL$=HEX$(SL)
720 SD=256*PEEK(&H8C3F): SE=PEEK(&H8C3E): ST=SD+SE: ST$=HEX$(ST)
730 PRINT "The 3-byte STRING DESCRIPTOR, (pointed to by DE), holds:"
740 PRINT "   ";RV$;"String length: ";SL$;NV$;"   ";RV$;"Address of string: ";ST$;NV$
800 PRINT:INPUT "                                                       AGAIN (Y/N)";R$
810 IF R$="Y" THEN PRINT ES$+"E": GOTO 250
900 END
1000                            ' Display subroutine
1010 S2$=" 2=INTEGER(%);": S3$=" 3=STRING($);":S4$=" 4=SINGLE PRECISION(!);"
1020 S8$=" 8=DOUBLE PRECISION(#)"
1030 A=VAL(A$):ON A GOTO 1090,1050,1060,1070,1090,1090,1090,1080
1040 RETURN
1050 PRINT RV$;S2$;NV$;S3$;S4$;S8$: RETURN
1060 PRINT S2$;RV$;S3$;NV$;S4$;S8$: RETURN      ' Submitted by:
1070 PRINT S2$;S3$;RV$;S4$;NV$;S8$: RETURN      '       Herbert A. Friedman, M.D.
1080 PRINT S2$;S3$;S4$;RV$;S8$;NV$: RETURN      '
1090 RETURN
```

Guide to Setting Up Utilities as Device Drivers

Charles E. Horn, P.E.
1714 Patricia Lane
Garland, TX 75042

There are those who still regard the writing of device drivers in assembly language as a mysterious art, in spite of the publication of such excellent articles as "The HDOS Device Driver Programmer's Guide", by Al Dallas, Dale Lamm, and Tom Jorgenson (REMark, Issue 20, September 1981). This article is an attempt to illustrate how ANY utility program might be converted to a device driver, to be resident at all times for immediate call.

Our example shows how a particular utility program from the article, "Split Byte Octal/Decimal Addition/Subtraction", by Robert G. Traub (REMark, Issue 30, July 1982), can be so converted, with almost no modification to the original program.

The first process is to set up the program in PIC code format according to HDOS rules. This requires a header that does just that, as shown in our example. We will refer the reader to the noted Programmer's Guide for an explanation of what it all means. Our code is as brief as possible for our purpose, with little of the usual formality.

The second process is to provide a means to call the program as desired. Our procedure is to intercept every keyboard entry and look for the calling character. For the example, a CTRL-K is used. We do this by zapping the console interrupt vector at UIVEC3 with a substitute vector to our program, then sending the character on to the original vector address if it is not a CTRL-K.

When the CTRL-K is detected, we jump to the utility program, and depend upon HDOS to keep us out of trouble with the stack as the program exits. This way, we don't have to modify the original program much at all. In fact, a disassembled version of a utility program might just as well be tacked on with little knowledge about how it works or how it normally exits.

For our example, the following code will serve the purpose:

```
        TITLE   'AS.DVD; ADDSUB Calculator, Converted to a DVD'
        STL     'by Charles Horn; 1982'

***     PROGRAM EQUATES
*
ENTCHR  EQU     'K'-100Q        CTL-K (OR WHATEVER)
UIVEC3  EQU     040045A         TT: INTERRUPT VECTOR
DC.LOD  EQU     9               LOAD REQUEST
AIO.DTA EQU     041053A         DEVICE TABLE ADDRESS
$TYPTX  EQU     031136A         TYPE TEXT (H-17 ROMSUB)

        CODE    PIC
REL     EQU     *-6

***     SET UP PIC FILE HEADER
*
        DB      307Q            PIC CODE FLAG
        DB      0               NO CAPABILITIES
        DB      1               MOUNTED UNIT MASK
        DB      1               MAX 1 UNIT
        DB      0,0,0,0,0,0,0,0 NO UNIT SUBCAPABILITY
        DS      002000A+REL-*   FIX DRIVER ENTRY POINT AT 002000A

***     I/O REQUESTS ENTER HERE AT THE DRIVER ENTRY POINT
*
        CPI     DC.LOD          SEE IF LOAD REQUESTED
LD.PAT  JNZ     IGNORE          IGNORE ANY OTHER REQUEST
        MVI     A,303Q          GET JMP INSTRUCTION
        STA     LD.PAT          PATCH PROGRAM - PREVENT SECOND LOAD
        LHLD    AIO.DTA         GET DEVICE TABLE ADDRESS
        INX     H
        INX     H
        MOV     A,M             GET LOAD STATUS FLAG
        ORI     2               ADD STATUS BIT
        MOV     M,A             MAKE LOAD PERMANENT

***     SET UP KEYBOARD INTERRUPT
*
        LHLD    UIVEC3+1        GET OLD TT VECTOR
        SHLD    TTVEC           STORE IT IN PROGRAM
        LXI     H,TTINT         GET NEW VECTOR TO PROGRAM ENTRY
        SHLD    UIVEC3+1        ZAP TT VECTOR
        CALL    $TYPTX
        DB      12Q,11Q,'Hit CTRL-K at any time to start CALCULATOR'212Q
        XRA     A               FLAG NO ERROR
        RET

***     IGNORE PROCESS FOR REQUESTS OTHER THAN LOAD
```

☞

```
IGNORE  XRA     A               FLAG NO ERROR
        RET

***     PROCESS KEYBOARD INTERRUPT
*
TTINT   PUSH    H               SAVE
        PUSH    PSW               REGISTERS
        IN      350Q            GET KEYBOARD ENTRY
        CPI     ENTCHR          SEE IF PROGRAM CALLED
        JZ      STARTP          IF SO - GO TO PROGRAM

*** AREA 1 - IGNORE THIS LINE FOR NOW

        POP     PSW             RESTORE
        POP     H                 REGISTERS
TTVEC   EQU     *+1             OLD TT VECTOR - PATCHED HERE ON LOAD
        JMP     *               NOT ENTCHR - BACK TO HDOS

*** AREA 2 - ALSO IGNORE THIS LINE FOR NOW

***     SHOW MESSAGE AND CLEAR OUT CONSOLE BY REQUIRING A RET HIT
*
STARTP  CALL    $TYPTX
        DB      12Q,11Q,'Hit Return to Start CALCULATOR',07Q,212Q
STARTP1 SCALL   .SCIN
        JC      STARTP1         WAIT FOR KEYSTROKE
        CPI     12Q             RETURN HIT?
        JZ      CLRSCR          GO CLEAR SCREEN AND START PROGRAM
        JMP     STARTP1         INSIST ON RETURN HIT

CLRSCR  CALL    $TYPTX
        DB      33Q,'E'+200Q    CLS
        JMP     (START LABEL)

***     APPEND THE UTILITY PROGRAM HERE
```

The original calculator program should be entered here. Look at the start label at the END statement of the program. Place that label in the above JMP instruction. In our example, that label is "START", and it is at the first statement in the program. In this case, the above JMP instruction could be deleted; however, not all programs are written this way.

It is advised that the original program be coded, assembled, and debugged in its original form first, to be sure that it will run. Debugging will be much easier this way. It should be noted that an error exists in the code as published. First, an origin statement:

```
        ORG     042200A
```

should be added just below the program equates. Second, missing code should be added at the bottom of the BEG2 routine:

```
        CPI     'Q'
        JZ      OCTIN
        JMP     ERROR
```

and the label OCTIN should be placed on the next line:

```
OCTIN   LXI     H,MESS8
```

After the program has been debugged, attach the assembly language source code to the bottom of the device driver header. Most text editors provide a means to merge files in this manner. Then, GO TO THE BOTTOM OF THE CODE AND REMOVE THE ENTRY LABEL "START" FROM THE END STATEMENT. PIC code does not require this label.

Of course, when creating device drivers in this manner, one must check the appended code very carefully for conflicts with labels that already exist in the DVD header. Also, duplication of program equates should be avoided. In our example, the appended program code was created without direct use of the normal ROMSUB and SYSCALL labels.

Now, about those lines in the code that we said to ignore; here is an optional addition. There are a few utility programs around that require use of the CTL-Z abort exit for termination. If you are operating your system in HDOS STAND-ALONE and have reset SY0 at any time, you will find that the CTL-Z exit will no longer work. Since our program intercepts keyboard entries anyway, we have an opportunity to overcome that problem. All we have to do is to look for the CTL-Z's and, when we see two in a row, exit to HDOS. After this program is loaded, it will process the CTL-Z's whether the utility part of the program is being run or not.

To set up the CTL-Z processor, enter the following lines of code at the line called "AREA 1":

```
        CPI     'Z'-100Q        IS IT ^Z?
        JZ      CTLZ            IF SO GO PROCESS IT
        MVI     A,0             IF NOT ^Z
        STA     ZCOUNT            ZERO THE ^Z COUNTER
```

Now, enter all of the following code at the line called "AREA 2":

```
###     THE CTL-Z ENTRY IS PROCESSED HERE
#
CTLZ    LXI     H,ZCOUNT        POINT TO THE ^Z COUNTER
        MOV     A,M             GET THE COUNT
        CPI     1               SECOND ^Z?
        JNC     CTLZXIT         IF SO - EXIT TO HDOS
        INR     M               IF NOT - BUMP COUNT
        CALL    $TYPTX
        DB      '^Z','?'+200Q   ECHO ^Z? TO SCREEN
        POP     PSW             RESTORE REGISTER
        LHLD    TTVEC           GET OLD TT VECTOR
        XTHL                    POP H - PUT TT VECTOR
        RET                                     ON STACK
###     THE SECOND ^Z COMES TO HERE
#
CTLZXIT CALL    $TYPTX
        DB      '^','Z'+200Q    ECHO ^Z TO SCREEN
        MVI     A,0             ZERO THE
        STA     ZCOUNT            ^Z COUNT
        POP     PSW             CLEAR
        POP     H                 STACK
        XRA     A               FLAG NO ERROR
        SCALL   .EXIT           EXIT TO HDOS

ZCOUNT  DB      0               ^Z COUNT STORED HERE
```

As one last note, remember that HDOS will permit a maximum of seven files of the form xx.DVD to exist on the system diskette. Secondly, TT.DVD and SY.DVD belong to the system; hence, there is a limit to how far we can go with resident utilities. Also, note that device drivers become part of the resident system, and occupy RAM space that is no longer available for user programs.

ETLOAD Correction!

The following is a correction to the ETLOAD program that was submitted in REMark Issue #39, page 41.

```
B6 0E E1 81 FF 26 01 83 BD F6 5B BD F7 E5 3D 0E
7E 77 BD 0F 86 7E 97 2D CE 00 B6 DF 2E 0E 8D 3C
81 AA 26 FA 8D 36 B7 00 C2 8D 31 B7 00 C1 8D 2C
B7 00 C4 8D 27 B7 00 C3 FE 00 C3 FF 00 C5 FE 00
C1 8C 00 00 26 05 FE 00 C5 6E 00 09 FF 00 C1 8D
0B FE 00 C3 A7 00 08 FF 00 C3 20 E2 86 00 B7 00
C7 86 80 B7 C2 20 F6 00 C7 C1 00 27 F9 86 FF 4A
26 FD B6 C2 A0 39 86 00 B7 C2 20 C6 01 F7 00 C7
39 00 00 00 00 00 00 00
```

Basically Speaking

Reading Disk Files While in MBASIC

(Note: The following material was taken from DENHUG, The Denver Heath Users Group, Inc. Newsletter. DENHUG, P. O. Box 20422, Denver, CO 80220.)

Bob Eson
9350 Green Ct.
Westminster, CO 80030

Not all information needs to reside in memory during the running of a program. Screen Instructions, Documentation, Graphics, e.g., anything that can be stored in a disk file (ASCII) can be read to the screen while in the MBASIC interpreter.

Create the 'Screen File' with a text editor, typing the file just as you would have it appear on the screen. It would be advisable to limit the file to 23 lines to prevent scrolling problems at the 24th line.

At the point in the program where you wish to call the FILE to the screen, you can branch to a call routine. By branch, I mean set up an IF-THEN statement "DO YOU WISH TO SEE THE SCREEN FILE (Y) <N> ?". IF Q$="Y" THEN GOSUB 10000 REM *** SCREEN FILE CALL.

```
10000 PRINT (CLEAR SCREEN) :I = 1
10010 OPEN "I",#1, "READFILE.SCN"
10020 LINE INPUT#1,A$
10030 IF EOF(1) GOTO 10050
10040 PRINT A$
10050 GOTO 10020
10060 LINE INPUT "ENTER <CR> WHEN READY TO PROCEED";Q$
10070 RETURN
```

Anyway, that kind of shows the basics of how it is done. The LINE INPUT command looks for a complete string with a delimiter to end it. Most ASCII files, created with an editor or word processor end each line with a <cr>. All you have to do is create your screen display with an editor and store it under a standard Dev:FNAME.EXT.

Oh yes, you may wonder what line 10060 is for. When you return to the main program, lines following may insert line feeds and start to scroll your display. The LINE INPUT holds the screen display until you are ready to continue.

This is a fun routine to play around with; the applications can be fascinating.

A SHORT TIP: When handling long strings with HDOS you can extend a string past the end of the physical line by using the '@' character. For example:

```
100  PRINT E1$;TAB(10);RV$;"NOW IS THE TIME FOR ALL GOOD MEN@
     TO COME TO THE AID ";ER$
```

When doing the same thing with CP/M BASIC you extend the string using the 'Linefeed' key in the same manner.

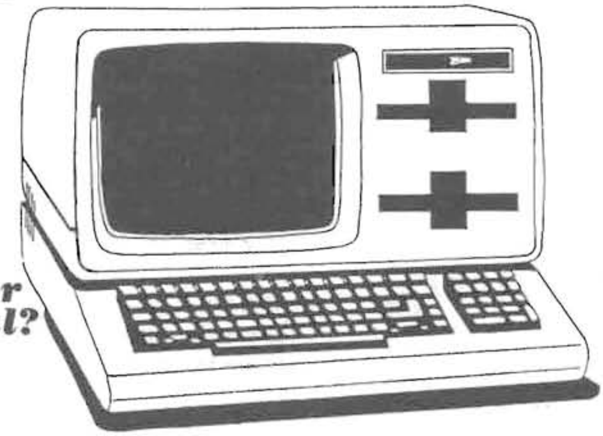
Can your Z-100 Computer pass these eight tests?

1. Is there COLOR in your life?

If not, you may be lacking either video RAM or a color display. Wizardry handles several top-quality displays as well as having the best price around on video RAM.

2. Does your Z-100 socialize?

Even computers need friends, our ZLYNK modem utility will allow your system to talk with others — and even exchange files! (Wait'll you see our LOW modem prices!!)



3. How's your Z-100's memory?

It takes 192K of CPU RAM to assemble your BIOS. We carry the best (fully-guaranteed) RAM chips for the Z-100 at probably the best price anywhere.

4. Can you read your writing?

Nothing is more sad than a Z-100 whose penmanship is poor. If you can't read your letters, neither can anyone else. Adding a letter or correspondence-quality printer can change all that.

5. Does your computer have growth potential?

We carry several types of compatible S-100 boards to expand your horizons, but NOT deflate your wallet. Ask about the Widget-100.

6. Is your Z-100 disk-hungry?

Our prices on disks will satisfy its appetite no matter how hungry it may be. We carry the disks (the best) that we use on our own systems: Memorex.

7. Can your Z-100 tell time?

If not, we can teach it. We produce a program that uses the Hayes Chronograph to get the time at bootup, or it will use the last date and time if missing.

8. Have you control over your Z-100's character?

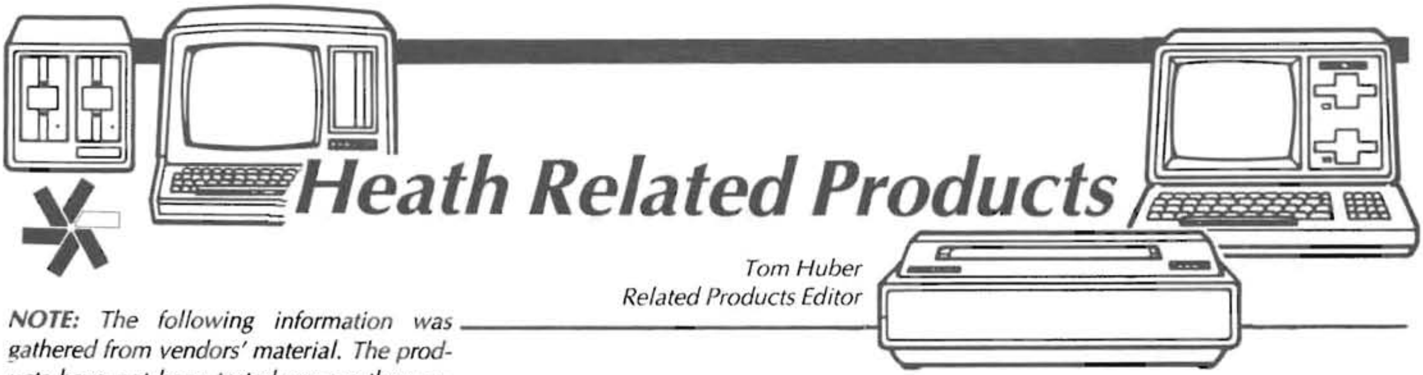
Our ALTCHR program allows creation of new characters as well as editing those you already have. Use it to create graphic symbols or whatever you desire.

Call or write for our FREE price list. Software Wizardry, Inc. is both an authorized Zenith Data System dealer (we carry all Heath/Zenith assembled computer products and software) and a computer product developer. Unlike computer salesmen, we can answer your toughest questions about the Heath/Zenith computer series. Come to us for your next computer purchase and we will meet or beat your best price on any product we carry (if humanly possible).

VISA MasterCard

122 YANKEE DRIVE
ST. CHARLES, MO. 63301
(314) 946–1968

SOFTWARE WIZARDRY, INC.



Heath Related Products

Tom Huber
Related Products Editor

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

Software Products for HDOS, CP/M, and Z-DOS

Software Wizardry, Inc., has announced a number of products for Heath/Zenith Data Systems computer products. Here is a brief listing of those we have received product notices for.

The MDS Spooler is a package of seven device drivers which will allow you to send files to printers (or other devices) at the same time you are using the computer for normal HDOS operation. Requires HDOS 2.0, 24K RAM, and one disk drive (any format).

The Excalibur Data Base System is a data base file manager incorporating menu-driven operation, user-friendly, H19-type display, and features formerly available in higher priced data base systems. Requires 64K, Microsoft BASIC, CP/M® 2.2 on an 8080/Z-80 type microprocessor running at 2 or 4 MHz, and one disk drive.

CRASH is an HDOS utility utilized for recovering corrupted files on hard-sectored disks and includes a short course on disk structures to assist the user. Requires HDOS 2.0, 32K RAM, an 8080/Z-80 type microprocessor, and a hard-sectored controller (it does not support the Z-89-37 soft-sectored controller).

Chronologic has the ability to automatically read a hardware clock such as the Hayes Chronograph™ from Z-DOS™ on bootup. Requires H/Z-100, Z-DOS, and one disk drive. Configured for the Hayes Chronograph but may be modified for other clocks (source code included).

The HDOS Toolkit is a package of nine assembly language utilities provided in source code for customization to suit your own needs. The programs included are: INSTALL, to update old programs; CMP, for file comparison; WFL, a FLAGS program that accepts wild cards; CRC, a file checksum generator; DSKMAP, displays hidden file characteristics; CRYPTO, a file encryptor and decoder; FILTER, filters spurious control codes from text; MLC, provides upper to lower-case conversion; and ASMFIX "pretty prints" source files. Requires HDOS 2.0, 32K RAM, any disk format.

ZLYNK is an intelligent HDOS Modem program that requires 32K RAM, HDOS 2.0, an 8080/Z-80 processor at 2 or 4 MHz, and one disk drive.

For details on any of these programs, contact the vendor.

Vendor: Software Wizardry, Inc.
122 Yankee Drive
St. Charles, MO 63301
Phone: (314) 946-1968

Prices:	MDS Spooler	$24.95
	Excalibur DBS	$49.95
	CRASH.	$24.95
	CHRONOLOGIC	$19.95
	HDOS Toolkit	$29.95
	ZLYNK.	$24.95

Soft-Sectored Floppy Disk Source

Mensa Media, Inc., carries a full line of 5.25 and 8-inch soft- sectored disks with a one year warrantee against defects in materials or workmanship. Contact the vendor for pricing.

Vendor: Mensa Media, Inc.
365 Orchard Road
P.O. Box 394
Wyckoff, NJ 07481
Phone: (201) 891-4029

Software Products for CP/M, Z-DOS

Single Source Solution™ is a new software company that is offering a number of software packages. Here is a brief list of those that we have received product notices for.

PORTMAP™ is a program that allows investors to analyze an existing investment portfolio that contains stocks, bonds, moneymarket instruments, and other securities. Schedule D is supported for reporting taxes. Requires MS-DOS (Z-DOS) or CP/M and Microsoft BASIC (specify operating system).

BUYSEL is a menu-driven package of mathematical and statistical routines for supporting stock, commodities, and options markets. Requires CP/M, 64K memory, and one disk drive.

The STUDENT RECORD SYSTEM™ is a file manager to aid in classroom management of students and their grades. Requires CP/M or UCSD Pascal™ and 32K minimum (43K standard) RAM.

MEMBERS™ is a mailing list management system that requires CP/M and Microsoft BASIC-80.

QUICKSCREEN™ is a software utility that allows use of the CRT screen as a blackboard and then will create a program that will create a program that draws a screen identical to the original display. Requires 8080/Z-80/8086/8088 based microprocessor, 48K RAM, CP/M 2.x, one disk drive, and a cursor addressable 24x80 CRT. Resulting programs can be written in Microsoft BASIC, CBASIC™, dBASE II™, or FMS80.

Contact the vendor for details on any of these programs and/or a complete catalog of over 100 software products.

Vendor: Single SOURCE Solution
2699 Clayton Road
Concord, CA 94519
Phone: (415) 680-0202

Prices:	PORTMAP.	$79.95
	BUYSELL.	$149.95
	STUDENT RECORD SYSTEM	$89.95
	MEMBERS.	$99.50
	QUICKSCREEN	$19.95

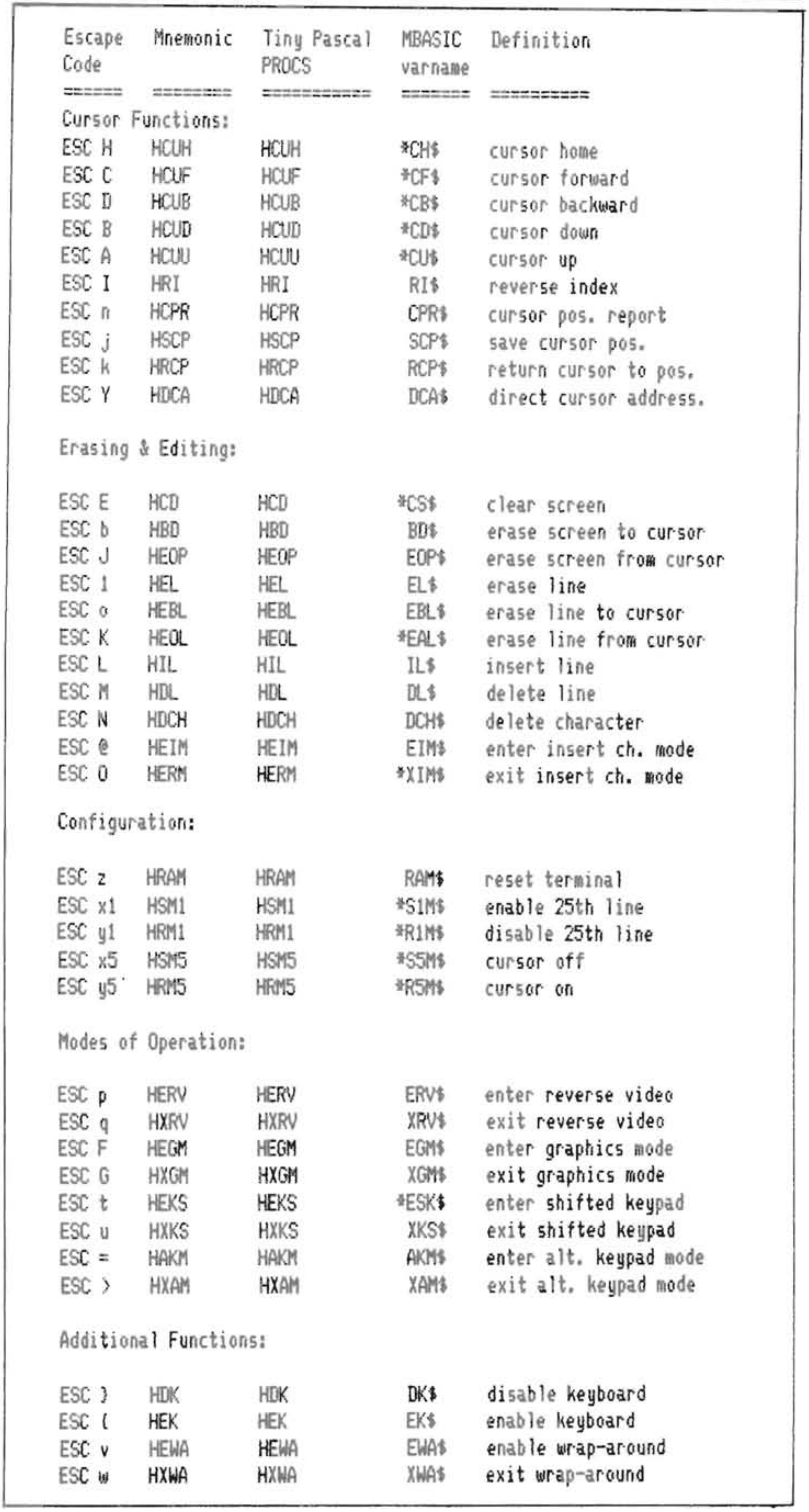
Price Reduction on Printer From MPI

Micro Peripherals, Inc. (MPI) has announced a retail price reduction of their popular PrintMate™ 99 to $599. The MPI PrintMate 99 features MPI's AP-PAK™ 100 cps bi-directional printing, friction and tractor-feed, serial and parallel interfaces, and a 1K buffer.

Vendor: Micro Peripherals, Inc.
4426 South Century Drive
Salt Lake City, UT 84107
Phone: 1-800-821-8848
(toll free, US)

PROCs for Tiny Pascal to the HUGBB on CompuServe. These, I think, could also be applied to MBASIC since this dialect of the language allows variable names of any length, although only the first two characters are significant. Some minor adjustments would have to be made for the latter in order to keep the names for each string variable from conflicting, while still reasonably conforming to the function mnemonics laid out in section 12 (or any other appropriate manual section) of the H88 Operations Manual.

The first minor adjustment would be the stripping of the initial "H" on the mnemonics since this is common to all. Further adjustments are noted with an asterisk in the following table.



Escape Code	Mnemonic	Tiny Pascal PROCS	MBASIC varname	Definition
======	========	===========	=======	==========
Cursor Functions:				
ESC H	HCUH	HCUH	*CH$	cursor home
ESC C	HCUF	HCUF	*CF$	cursor forward
ESC D	HCUB	HCUB	*CB$	cursor backward
ESC B	HCUD	HCUD	*CD$	cursor down
ESC A	HCUU	HCUU	*CU$	cursor up
ESC I	HRI	HRI	RI$	reverse index
ESC n	HCPR	HCPR	CPR$	cursor pos. report
ESC j	HSCP	HSCP	SCP$	save cursor pos.
ESC k	HRCP	HRCP	RCP$	return cursor to pos.
ESC Y	HDCA	HDCA	DCA$	direct cursor address.
Erasing & Editing:				
ESC E	HCD	HCD	*CS$	clear screen
ESC b	HBD	HBD	BD$	erase screen to cursor
ESC J	HEOP	HEOP	EOP$	erase screen from cursor
ESC l	HEL	HEL	EL$	erase line
ESC o	HEBL	HEBL	EBL$	erase line to cursor
ESC K	HEOL	HEOL	*EAL$	erase line from cursor
ESC L	HIL	HIL	IL$	insert line
ESC M	HDL	HDL	DL$	delete line
ESC N	HDCH	HDCH	DCH$	delete character
ESC @	HEIM	HEIM	EIM$	enter insert ch. mode
ESC O	HERM	HERM	*XIM$	exit insert ch. mode
Configuration:				
ESC z	HRAM	HRAM	RAM$	reset terminal
ESC x1	HSM1	HSM1	*S1M$	enable 25th line
ESC y1	HRM1	HRM1	*R1M$	disable 25th line
ESC x5	HSM5	HSM5	*S5M$	cursor off
ESC y5	HRM5	HRM5	*R5M$	cursor on
Modes of Operation:				
ESC p	HERV	HERV	ERV$	enter reverse video
ESC q	HXRV	HXRV	XRV$	exit reverse video
ESC F	HEGM	HEGM	EGM$	enter graphics mode
ESC G	HXGM	HXGM	XGM$	exit graphics mode
ESC t	HEKS	HEKS	*ESK$	enter shifted keypad
ESC u	HXKS	HXKS	XKS$	exit shifted keypad
ESC =	HAKM	HAKM	AKM$	enter alt. keypad mode
ESC >	HXAM	HXAM	XAM$	exit alt. keypad mode
Additional Functions:				
ESC }	HDK	HDK	DK$	disable keyboard
ESC {	HEK	HEK	EK$	enable keyboard
ESC v	HEWA	HEWA	EWA$	enable wrap-around
ESC w	HXWA	HXWA	XWA$	exit wrap-around

[Some of these can be dangerous; the most obvious is HDK (disable keyboard)! So use caution.] But I think you get the idea at this point. The reason, of course, is to make the listing more readable (and therefore understandable). This will be particularly the case if the person perusing the listing is already familiar with the mnemonics. I have, by the way, no quibble with Mr. Warnick's use of E$ for ASCII 27; I am just uneasy about his following the usual practice of using E1$ to E9$ (and whatever else) for escape coding. That may be necessary for Extended Benton Harbor BASIC, but not MBASIC!

One final note: if someone would like a listing of Frank's Tiny Pascal PROCs, I would be happy to supply them, provided that he or she sends me an SASE.

Kirk L. Thompson
Box 24, Rte. 1
Oxford, IA 52322

Bug in Z-DOS Operating System

Dear HUG,

A minor bug was found in the generic format module for the Z-DOS operating system. This module is used by FORMAT, MAKE, DSKCOPY, and BACKUP, and thus the problem appears in all four programs.

The problem occurs when a user has MAPped his drives' logical names. If he then runs one of the above programs and attempts to format a mapped drive, the disk in the drive is formatted properly. Then a system call is executed, but the drive number given is not mapped, thus the system attempts to go to the real drive instead of the logical drive. This IN NO WAY harms either the newly formatted disk, or a disk in the real drive, but it is an inconvenience (if no disk is in the real drive, you will get a NOT READY error message).

The problem was discussed with Doc Campbell on the telephone several days ago, and he decided to put patches for the bugs on the HUG bulletin board. The problem discussed above exists in ALL versions of the 1.10 bios/utilities that were ever shipped. It has never been corrected. Users may see Doc's messages on the board and attempt to get fixed software by calling in. There is no "fixed software". The patches to fix the software are given below.

Program	Address
FORMAT.COM	OE1A
MAKE.COM	18EA
DSKCOPY.COM	152A
BACKUP.EXE	601A

The procedure for patching a .COM file is:

```
A:DEBUG  xxxxxxxx.COM
-ESC:aaaa
ssss:aaaa   CD.90   21.90
-W
-Q
```

You must replace the two bytes at the given address from CD 21 to 90 90.

The procedure for patching an .EXE. file is:

```
A:RENAME  xxxxxxxx.exe  xxxxxxxx.bin
A:DEBUG
-Nxxxxxxxx.BIN
-Lssss:O
-Essss:aaaa
ssss:aaaa   CD.90   21.90
-Wssss:O
-Q
A:RENAME  xxxxxxxx.bin   xxxxxxxx.exe
```

Above, the value for ssss in the L command is chosen by first executing an R command and noting the value for the DS register. Add 100H to this value to get a safe value for ssss.

Skip Gwyer
Software Consultation Group

Corrections to "A Faster Benton Harbor BASIC"

Dear Walt,

Please refer to "A Faster Benton Harbor BASIC", REMark #39, pg. 11. After typing in the entire program without a mistake (a first for me), I tried to assemble it 4 or 5 times without success. I kept coming up with "U" errors for ALL the SCALL's in the program.

A few minutes of research led me to notice there was no XTEXT or EQUates for the SCALL's in the program. Either I am doing something wrong or Dahl Metters forgot to include it in his listing.

Please investigate the program and either let me know where I am wrong OR add the following to the list at the beginning of the listing:

```
         XTEXT    HOSDEF
USERFWA EQU       42200A
etc....
```

[Ed: After checking, I find that you are right. Thanks for bringing this to our attention.]

David Orosz

Cooling Fan for H/Z-89

Dear HUG,

In light of the cooling fan article found in REMark Issue 37, I would like to pass a novel idea discovered at out last LAHUG meeting.

KRES Engineering gave a great demonstration of their hardware. They had brought in an H-89 with a clear acrylic top cover, showing their fine expansion board and internal fan. After their demonstration, we flooded them with questions. When asked about the clear top, they described the observation of smoke during air circulation tests. Then, someone asked, "Why didn't their clear top have ventilation slots?". They simply said, "It runs cooler without them.". Apparently, after installing a fan, the convection cooling slots become a hindrance to efficient air removal.

Thus intrigued, I had to try a simple test of my own. Months ago, I removed the slot dividers directly above my fan and covered the hole with a nice chrome grill. My test started by measuring the exhaust temperature. After fifteen minutes of warm up, my trusty Weston Photo Thermometer read 96 degrees. Then, I used duct tape to seal off the vents inside the top. Wouldn't you know it, the temperature went down to 89 degress, a drop of seven degrees. Now, fresh air is being pulled up through the bottom slots, over the boards, and out the top. Who knows, maybe someday we will be able to chill ice inside the H/Z-89!!

Larry Fina

YOU ARE INVITED TO THE SECOND NATIONAL HUG CONFERENCE

see details on page 7 of this issue

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

CUT ALONG THIS LINE

HUG MEMBERSHIP RENEWAL FORM

When was the last time you renewed?

Check your ID card for your expiration date.

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

Name ______________________________

Address ______________________________

City-State ______________________________

Zip ______________________________

REMEMBER — ENCLOSE CHECK OR MONEY ORDER

CHECK THE APPROPRIATE BOX AND RETURN TO HUG

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

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

Heath
Users'
Group

Hilltop Road
Saint Joseph, Michigan 49085

Volume 4, Issue 6