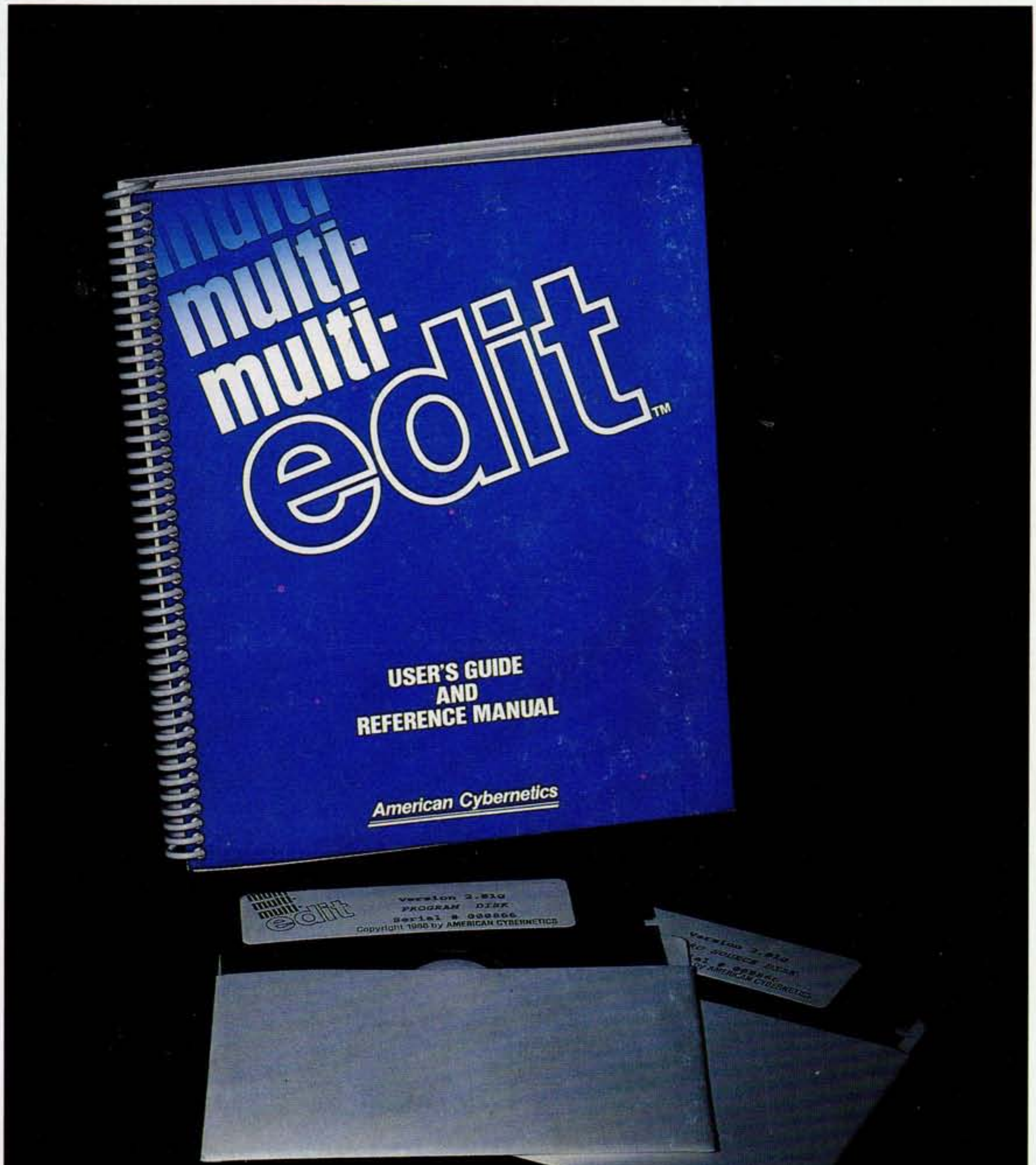
REMark®

Volume 9, Issue 10 • October 1988

P/N 885-2105 Issue 105

$2.50

Important Local Users' Group Survey
See Page 3



A Publication of Heath/Zenith

"I've Got *Your* Number Now . . ."

"Now you have no reason for us to be strangers. I know each and every one of you personally! My human gave me the entire HUG Database to study and now there's no need for **any** HUG member to register. Just have your ID Number handy when you call . . . I do! I'll bet you didn't know that my Bargain Centre has discounts on HUG products, as well as other computer **and** non-computer related items. And, did you know that I can renew your HUG membership . . . at a discount?!! I can give you over 35 megabytes of free software. All you have to do is take it! Now no-one will **ever** have to know about the two of us. Just set your modem to 300, 1200, or 2400 baud, 8-data bits, no-parity, and call me anytime your computer needs someone to talk to. I can be contacted at (616) 982-3956. Although no longer required, my human can be reached at (616) 982-3837.

FROM THE CPU...

JIM BUSZKIEWICZ

In my July 1988 Editorial, I talked about how you could add a third floppy (or 3-1/2" micro-floppy) disk drive to your H/Z-248/386 system. While in the process of experimenting with the hardware, I received an article from Mr. Mike Raick, which talked about the same basic topic, only his method of accomplishing the same goal, was somewhat simpler to implement. My idea was to switch the actual lines used to select drive B:. Splicing the tiny wires in the 34-conductor cable can be frustrating, especially if you've been out partying the night before! Mr. Raick's approach is much simpler. Switch the actual drive select jumpers on the drive itself. In this issue, you'll find Mike's complete article explaining how he did it.

You may have noticed that we're now using an advertising service for our ads. This includes ads for our new 'classified' section of the magazine. Not only should this improve service (less errors, reminders, etc.) to our Heath/Zenith related vendors, but should also provide for a better pricing structure. If you're interested in advertising in REMark, why not give Don Rupley, the president of Rupley's Advertising Service, a call. His number is (616) 983-4550.

Local User Groups, take note. We're coming up on the end of the year, and will be preparing to publish our annual list of local clubs in the January 1989 issue of REMark. This year, this list is going to be updated completely. We're finding too many errors with the current list. **IF YOU WOULD LIKE YOUR CLUB TO BE LISTED, YOU MUST SEND IN THE FORM (or a copy of it) LOCATED AT THE BOTTOM OF THIS PAGE!** We must receive this form BEFORE November 25th, 1988. If we don't get this form, your club will NOT be listed. Many times throughout the year, we're asked by users, where is the location of the nearest local users' group. Not only will this list provide us with that information, but will also help your membership since we will be able to steer that individual to your group!

Local HUG Club Survey

Club Name: ______________________________

Address: ______________________________

City/State/Zip: ______________________________

Voice Phone Number(s): ______________________________

BBS Phone Number(s): ______________________________

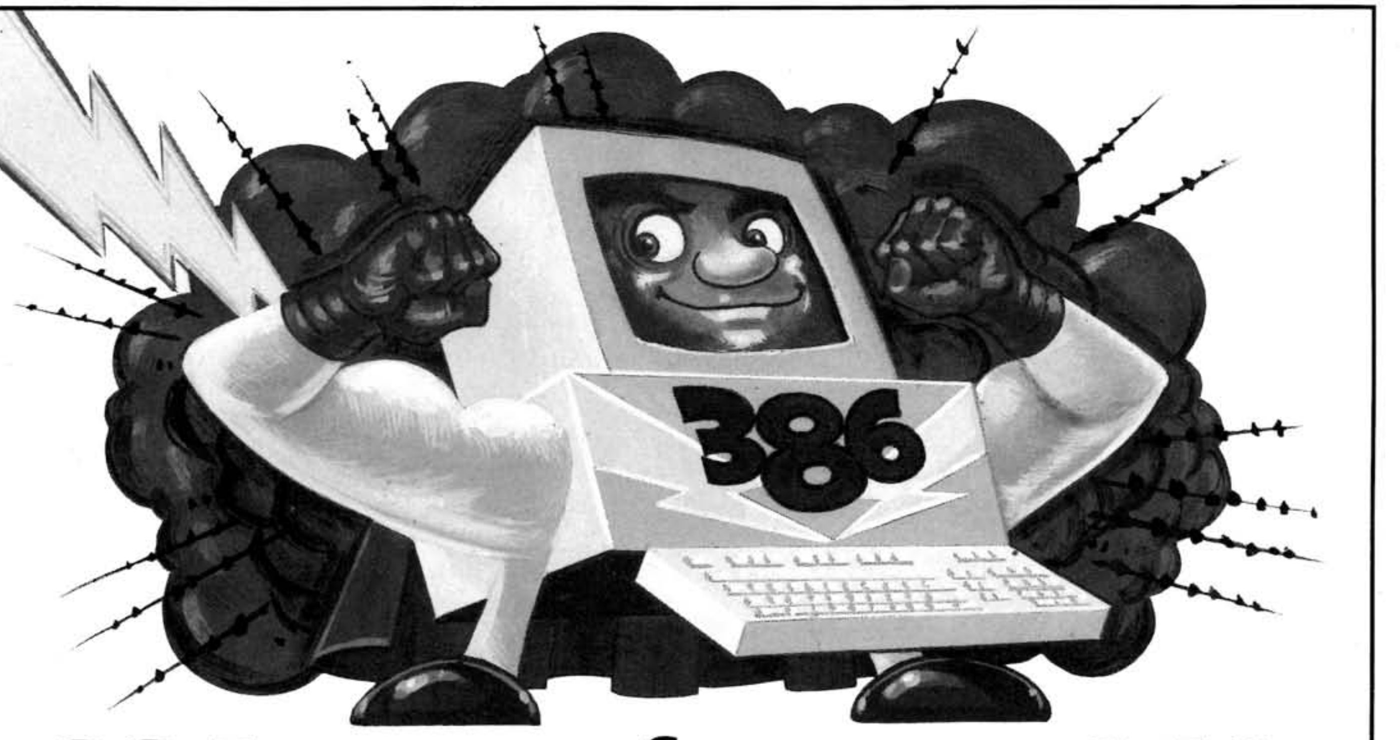
Person(s) to Contact: ______________________________

Meeting Times: ______________________________

Meeting Location: ______________________________

Mail a copy of this form or the equivalent information directly to:

HUG
P.O. Box 217
Benton Harbor, MI 49022-0217



386 power for your 248

Now there are two sensational Upgrade Kits to give your 286-based computer kit the power of an H-386 Desktop.

The HUG-386 Upgrade Kit lets you convert your H-241 or H-248 series computer to a full H-386. **ONLY $2699**

The HUG-386-C Upgrade Kit adds a dual floppy disk/hard disk controller card to the HUG-386 to let you upgrade H-248's that don't have a Winchester controller card, or H-241's that have controllers incompatible with the HUG-386. **ONLY $2849**

If you're a HUG member and kit builder, you may be eligible for special discounts. Call for further details.

Available NOW at Heath/Zenith Computers & Electronics Centers in the U.S.

ARIZONA – Phoenix, 85017
2727 W. Indian School Rd.
602-279-6247

Tucson, 85711
5616 E. Broadway
602-745-0744

CALIFORNIA – Anaheim, 92805
330 E. Ball Rd.
714-776-9420

San Jose (Campbell), 95008
2350 S. Bascom Ave.
408-377-8920

El Cerrito, 94530
6000 Potrero Ave.
415-236-8870

San Diego (La Mesa), 92041
8363 Center Dr.
619-461-0110

Los Angeles, 90007
2309 S. Flower St.
213-749-0261

Pomona, 91767
1555 N. Orange Grove Ave.
714-623-3543

Redwood City, 94063
2001 Middlefield Rd.
415-365-8155

Sacramento, 95825
1860 Fulton Ave.
916-486-1575

Woodland Hills, 91364
22504 Ventura Blvd.
818-883-0531

COLORADO – Denver (Westminster), 80003
8725 Sheridan Blvd.
303-429-2292

FLORIDA – Hialeah, 33012
4705 W. 16th Ave.
305-823-2280

Jacksonville, 32211
9426 Arlington Expressway
904-725-4554

Orlando (Altamonte Springs), 32714
445 W. State Hwy. 436
Suite 1037
407-682-9111

Plantation, 33317
7173 W. Broward Blvd.
305-791-7300

Tampa, 33614
4019 W. Hillsborough Ave.
813-886-2541

GEORGIA – Atlanta, 30342
5285 Roswell Rd.
404-252-4341

HAWAII – Honolulu (Pearl City), 96782
98-1254 Kaahumanu St.
808-487-0029

ILLINOIS – Downers Grove, 60515
224 Ogden Ave.
312-852-1304

Schaumburg, 60173
1213 E. Golf Rd.
312-330-1717

Skokie (Chicago), 60076
3606 W. Dempster
312-583-3920

INDIANA – Indianapolis, 46220
2112 E. 62nd. St.
317-257-4321

KANSAS – Kansas City (Mission), 66202
5960 Lamar Ave.
913-362-4486

KENTUCKY – Louisville, 40243
12401 Shelbyville Rd.
502-245-7811

LOUISIANA – New Orleans (Kenner), 70062
1900 Veterans Memorial Hwy.
504-467-6321

MARYLAND – Baltimore, 21234
1713 E. Joppa Rd.
301-661-4446

Rockville, 20852
5542 Nicholson Lane
301-881-5420

MASSACHUSETTS – Peabody, 01960
242 Andover St. (Rt. 114)
617-531-9330

Wellesley, 02181
165 Worcester Ave. (Rt. 9)
617-237-1510

MICHIGAN – Farmington Hills, 48018
29433 Orchard Lake Rd.
313-553-4171

East Detroit, 48021
18149 E. Eight Mile Rd.
313-772-0416

St. Joseph, 49085
2987 Lake Shore Drive
616-982-3215

MINNESOTA – Minneapolis (Hopkins), 55343
101 Shady Oak Rd.
612-938-6371

St. Paul, 55106
1645 White Bear Ave.
612-778-1211

MISSOURI – St. Louis (Bridgeton), 63044
3794 McKelvey Rd.
314-291-1850

NEBRASKA – Omaha, 68134
2311 N. 90th St.
402-391-2071

NEW JERSEY – Ocean, 07712
1013 State Hwy. 35
201-775-1231

Fair Lawn, 07410
35-07 Broadway (Rt. 4)
201-791-6935

NEW YORK – Amherst, 14226
3476 Sheridan Dr.
716-835-3090

Jericho, L.I. 11753
15 Jericho Turnpike
516-334-8181

Rochester, 14623
937 Jefferson Rd.
716-424-2560

N. White Plains, 10603
7 Reservoir Rd.
914-761-7690

NORTH CAROLINA – Greensboro, 27407
4620-C W. Market St.
919-299-5390

OHIO – Cincinnati (Springdale), 45246
131 West Kemper Rd.
513-671-1115

Cleveland, 44122
28100 Chagrin Blvd.
216-292-7553

Columbus, 43229
2500 Morse Rd.
614-475-7200

Toledo, 43615
48 S. Byrne Rd.
419-537-1887

OKLAHOMA – Oklahoma City, 73139
7409 South Western
405-632-6418

OREGON – Portland (Tigard), 97223
10115 S.W. Nimbus Ave.
Nimbus Center
503-684-1074

PENNSYLVANIA – Frazer, 19355
630 Lancaster Pike (Rt. 30)
215-647-5555

Philadelphia, 19149
6318 Roosevelt Blvd.
215-288-0180

Pittsburgh, 15235
3482 Wm. Penn Hwy.
412-824-3564

N. Pittsburgh (Wexford), 15090
Wexford Plaza
10624 Perry Hwy.
412-935-8555

RHODE ISLAND – Warwick, 02886
558 Greenwich Ave.
401-738-5150

TENNESSEE – Memphis, 38117
700 Mount Moriah
901-684-1161

TEXAS – Dallas, 75218
12022C Garland Rd.
214-327-4835

Fort Worth, 76116
6825-A Green Oaks Rd.
817-737-8822

Houston, 77008
1704 W. Loop N.
713-869-5263

San Antonio, 78216
7111 Blanco Rd.
512-341-8876

UTAH – Salt Lake City (Midvale), 84047
58 East 7200 South
801-566-4626

VIRGINIA – Alexandria, 22303
6201 Richmond Hwy.
703-765-5515

Virginia Beach, 23455
1055 Independence Blvd.
804-460-0997

WASHINGTON – Seattle, 98109
505 8th Ave. N.
206-682-2172

WISCONSIN – Milwaukee (Wauwatosa), 53226
845 N. Mayfair
414-453-1161

Phone orders accepted.

Heath ZENITH Computers & Electronics

Your TOTAL computer center for Service
- ***Support • Software***
- ***Accessories***
- ***User Training***
- ***Competitive Prices***

HZC-466

On the Cover: Need a good text editor? Check out Multi-Edit! Thomas Lisanti reviewed it and found it did everything he wanted, maybe it will do the same for you. See page 45.

REMark®

Managing Editor .... Jim Buszkiewicz
(616) 982-3837

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

Production Coordinator .... Lori Lerch
(616) 982-3794

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

HUG Bulletin Board ..... (616) 982-3956

HUG Parts Ordering ..... (616) 982-3838

Contributing Editor .... William M. Adney

Contributing Editor .... Joseph Katz

Printer .... Imperial Printing
St. Joseph, MI

	U.S. Domestic	APO/FPO & All Others
Initial	$22.95	$37.95*
Renewal	$19.95	$32.95*

* U.S. Funds

Limited back issues are available at $2.50, plus 10% shipping and handling — minimum $1.00 charge. Check HUG Product List for availability of bound volumes of past issues. Requests for magazines mailed to foreign countries should specify mailing method and appropriate added cost.

Send Payment to: Heath/Zenith Users' Group
P.O. Box 217
Benton Harbor, MI 49022
(616) 982-3838

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 Heath/Zenith Computers & Electronics Centers or Heath Technical Consultation.

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

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

Features

Columns

Series

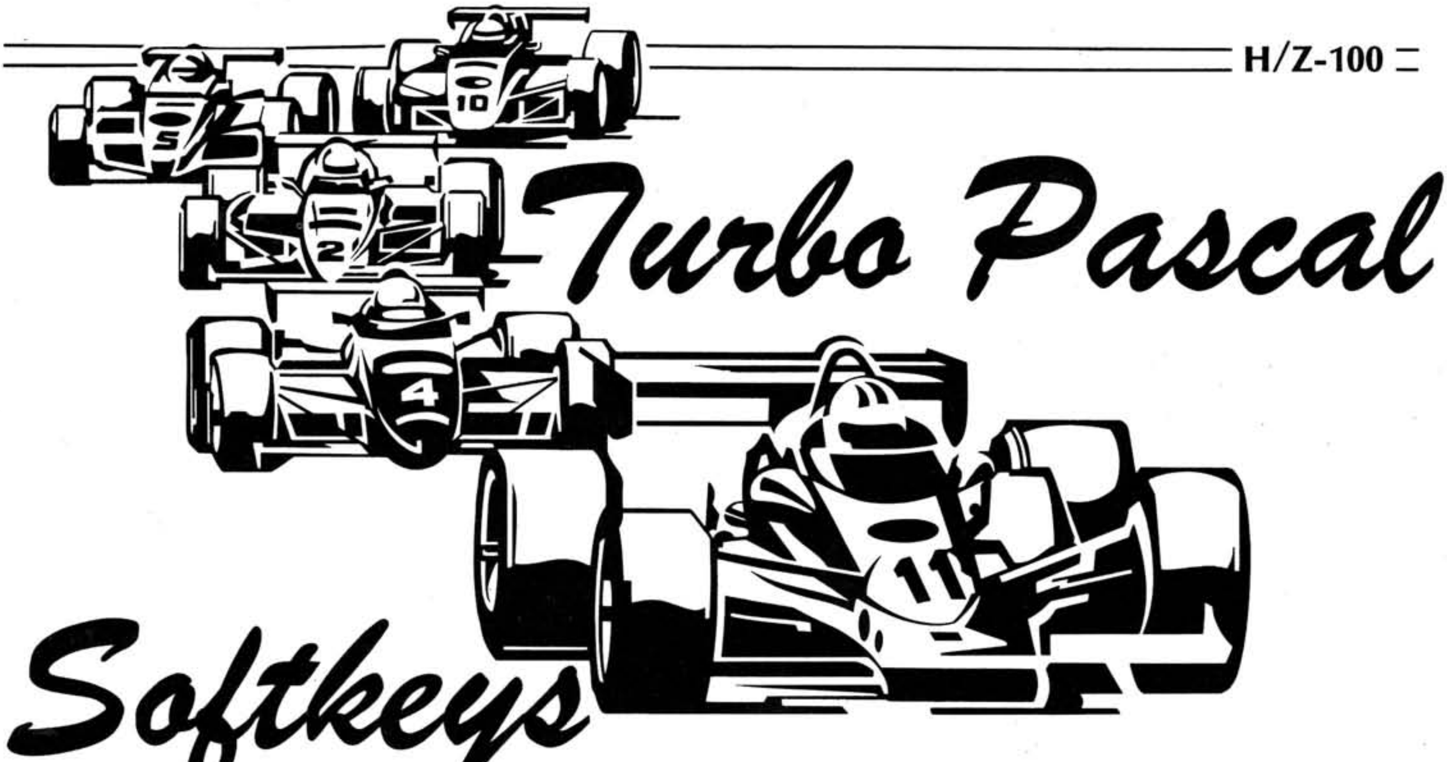
Reader Service No.		*Page No.*
104	*FBE Research Company*	*81*
105	*First Capitol Computer*	*15,16*
105	*First Capitol Computer*	*40*
*****	*HUG MCP*	*84*
*****	*HUG Members Only*	*83*
*****	*HUG PBBS*	*2*
107	*Paul F. Herman*	*20*
108	*Hogware Company*	*61*
111	*KEA Systems*	*26,81*
117	*Payload Computer Services*	*52*
121	*Scottie Systems*	*35*
*****	*Veritechnology Elec. Corp.*	*4*

Resources

Attention Advertisers!!

Advertising for REMark Magazine will now be handled through:

Rupley's Advertising Service
Dept. REM, 240 Ward Avenue
P.O. Box 348
St. Joseph, MI 49085
(616) 983-4550



Turbo Pascal Softkeys for your Z-100

Ronald J. Perrella
Rose-Hulman Ins. of Tech.
5500 Wabash Avenue
Terre Haute, IN 47803

Introduction

Have you ever wanted to have programmable function keys under Turbo-3 with your Z-100? Did you want to have them under complete program control? Did you want to do this without using a TSR (Terminate-Stay-Resident) program? Well, guess what. This is how you go about doing this.

This article describes a module that provides any Turbo-Pascal V3.x programmer with "soft-keys." I call "soft-key" any key that can be reprogrammed to return another value (or set of values). What I have done is write a package of routines that allow you to turn any key into a soft-key. Indeed, all that is required is that you enter in the listing and use the {$I Softkey .inc} compiler directive to include these routines into your program. But this article will do more for you than that. It will describe the inner workings of this package, as well as the Turbo Pascal internals this package "hooks" into.

Soft-keys can be very useful. With this package, you can make ordinary programs a little "jazzier" and give Turbo Pascal programs the capability that all ZBASIC and GWBASIC programmers have had for years with their *KEY* commands. But before we can use them, let's . . .

Begin at the Beginning

Let's begin by examining the specifications that this package was designed to meet:

1. Run on Zenith Z-100.

2. Permit mapping of any ASCII character to a string of up to 255 characters.

3. Key macros will not allow recursion.

4. Existing programs need not be modified to use the package.

The first criterion comes from the fact that I own a Z-100. This package will not function properly on an IBM PC or compatible. That is because this package maps 1 character to 1 string and the IBM function keys are represented by 2 characters. The second criterion allows me to map *any* character. Therefore, cursor keys and other such items can also be mapped. This is possible because I use the Z-100 "disable key expansion" mode that returns all keys as a 1 character code. For more information on this, check your Z-100 Users' Manual, Appendix B.

The third requirement is that the mapping is non-recursive. This is to make programming easier. With a possible string length of 255 characters, it seems unlikely that a user would need to resort to such tricks to save space.

The fourth and final requirement comes from the fact that I wish to add this feature. to programs I have already written. Therefore, this package does not require you to call special input routines. Ordinary *read* and *readln* statements are used. Even *keypressed* will work correctly! In fact, Turbo Pascal has no way of knowing what is actually going on!

Revelation

How is this miracle accomplished? Well, through the use of "User Written I/O

Drivers" (page 241 in the Turbo Pascal Manual.) In other words, we will replace Turbo's standard I/O routines with our own!

Here are the two functions we will "hook" into:

Function	**BDOS Call**	**Pointer**
Function ConSt: Boolean	6	ConStPtr
Function ConIn: Char	6	ConInPtr

The table above will take a little explanation. The pointer column indicates the predefined pointer variable which contains the address of the ConSt (console Status) and ConIn (Console Input) functions. These pointers, by default, contain the addresses of routines which call BDOS (Basic Disk Operating System) function #6. These routines use Turbo Pascal calling conventions. This means that we can replace them with routines written in Turbo Pascal! There is no need to use Assembler! That is a powerful feature that is significantly better than many other systems.

What we are going to do is "patch" these routines with our own. We will call these routines *softkeyGet* and *softkeySt*. These will be what we shall call "invisible routines." Your Turbo Pascal programs will not access *softkeyGet* and *softkeySt*.

What your programs will access are called "visible routines", obviously. They are *softkeyAssign*, *softkeyDelete* *softkeyEnable*, *softkeyDisable*,, *softkeyInit*, and *softkeyExit*. Athough their functions seem intuitive, let's go over each one.

The Means

The first, *softkeyAssign* is used to map one character to any given string. It is "gorilla-proof" and prevents you from re-mapping a key without first deleting the previous mapping. This is important since we are using dynamically allocated memory.

The second, *softkeyDelete* is used to perform just the opposite. It is used to remove the mapping of a key and releases previously allocated memory, if any.

The next two routines, *softkeyEnable* and *softkeyDisable*, are used to respectively enable and disable the softkey feature during runtime. We will discuss various uses of these two routines during the explanation of the sample program included with this article.

Finally, the last two routines, *softkeyInit* and *softkeyExit* are used to initialize and de-initialize the softkey environment. They are called once and only once in *every* program. *SoftkeyInit* is called at the very beginning of a program. *SoftkeyExit* is called at the very end.

Play-by-Play

Let's begin studying this package. This is how you will gain insight into the "User-written I/O Drivers." First, take a look at *Var* section. Notice that only four variables are declared. All except one are preceded by the prefix "sk" to distinguish them from the calling program's own declarations. Look at *sk*. This is a pointer to a table of pointers to strings. Whew! That was complicated. Confused? Maybe the following drawing will clear things up. (see Figure 1) It really is not that bad.

Although you cannot really see it here, each string will really be of different length! But that will be explained later on. Always in the *Var* section, notice the *skprevIn* and *skprevSt* variables. These variables contain the addresses to the original I/O vectors. We may need to restore these later so we store them here. We will skip the *translating* variable at this time.

Now, have a look at perhaps the most important procedure here: *softkeyGet*. This procedure is the key to it all. As mentioned before, this procedure will substitute the standard ConIn function and provide us with the Softkey capability. For all you Computer Science types out there, notice that I have used a *Label* declaration. That means that I will be using a *Goto* statement somewhere along the line! Horror! Yes, I know. A Junior majoring in Computer Science should not do such a thing. Perhaps I felt a need for provocation. Well, if anybody out there thinks they have a cleaner and faster way to do this, please send me your solution. Enlighten me. Meanwhile, the rest of us will content ourselves with this tiny little "evil". (By the way, I do not believe that even limited recursion is the right way to do this.)

Continuing on, we notice that the *softkeyGet* routine really has two "states." Either it is "translating" or it is not. If it is not translating, it means that a character must be read from the keyboard. If it is, it means that a character must be read from within a softkey string. At this point, you should recognize a dilema: I need to have variables that have already been initialized! Do not despair! Turbo Pascal provides such an animal. But its implementation might be quite a surprise. Indeed, we are forced to use "Typed Constants." This form of constant is, in fact, not constant at all! Examine the *Const* declarations of *softkeyGet*. Three variables are declared: *Index*, *len*, and *key*. Notice that each line is of the form:

```
<identifier> : <type> = <value>
```

That is how "typed Constants" are defined. This facility is extremely useful in cases just like this one. It also means that when the procedure (or function) is called again, the values of constants declared in this manner are *intact*!

Assuming that *translating* is initialized to false (a quick glance of *SoftkeyInit* will

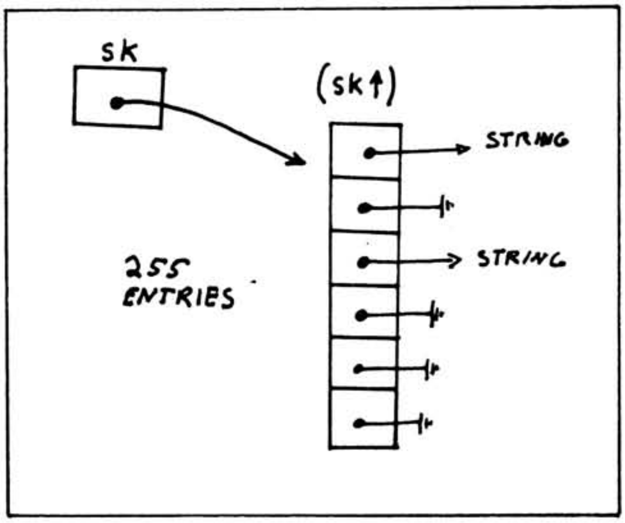


Figure 1

prove it), we see that the *then* clause will be executed. First, a character is read from the console device using BDOS function #6. This is necessary since we have already patched Turbo's I/O drivers and do not wish to have indefinite recursion. So we do the console input ourselves. Next, the character in *key* is looked up in the *sk*. If a NIL pointer is present, we just return the value of *key*. Otherwise, we set up some variables and change the state to *translating* equal to true. We now want execution to continue in the true state. We do this by using a *goto*. It causes the routine to start executing the code of the outer *else* statement.

The outer *else* statement controls how each character of a softkey definition is passed back to Turbo. Basically, it dishes out one character per call until the end of the string is reached. At that point, it changes the *translating* state back to false and performs a *goto* back to the top of the routine. This causes the outer *then* statement to be executed.

I recognize that this may be a little confusing. If you are having trouble following this, make a little drawing that describes what is going on. If you think about it a little, you will see that only one *goto* is executed per call.

The *softkeySt* routine is supposed to return the status of our new driver. It merely checks to see if we are translating. If we are, it returns true. If not, it does the actual BDOS calls to check if a character is waiting to be read. The reasons for using a BDOS call are the same as those evoked in the paragraph on *softkeyGet*.

Meet the "Visibles"

At this point, we arrive at a higher level in the package. We will now discuss the storage of the softkey definitions. As we have seen before, each softkey is an entry in the *sk* pointer table. The pointer contains either NIL or a valid pointer to a string. As mentioned before, these strings are *not* of fixed length. This is done by using both Turbo's string capability and Turbo's *getmem* and *freemem* procedures. We all know (or should know) that Turbo stores strings as an array of characters where the length of the string is contained in the first byte of the array. What I have done here is use that length to determine how much actual storage that string really needs.

Take a look at *softkeyAssign*. It takes a string and stores it in the *sk* table according to the *skey* character that was passed in the argument list. In order to save space, it allocates a number of bytes equal to the length of the string plus one using *getmem*. That extra byte contains the string length. Then it stores the string in that newly allocated piece of memory.

Later, *softkeyDelete* takes a character argument, finds the corresponding string, looks at its length, adds one to it and deallocates the string storage using *freemem*.

If you want to learn more about *getmem* and *freemem*, try reading the Turbo Pascal V3.0 manual on page 125.

But wait, there's more . . .

The Example

You probably have all kinds of new concepts milling around in your head. So let's

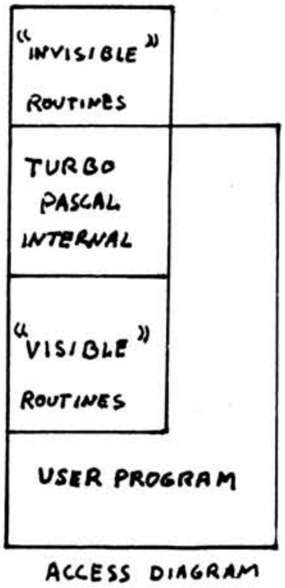


Figure 2

Listing 1

```
Program SoftKey_Example;
Const Esc=#27;          {Ascii Escape character}
Var C:Char;

{$I b:SOFTKEY.INC}      {Softkey module}
procedure Present;
begin
     Writeln('Enter ^C to exit, ESC to redefine keys, anything else to see.');
end;

procedure Redefine;
var
c,             {Redefined key}
t,             {Terminator key}
k:char;        {Input key}
s:longstring;  {string[255] as defined in softkey.inc}
begin
        ClrScr;
        NormVideo;
        SoftkeyDisable;
        Write('Enter key you wish to redefine:');
        Read(Kbd,c);
        case c of
        #0..#31: Writeln('^',chr(ord('@')+ord(c)))
        else Writeln(c)
        end;

        if C=ESC then begin writeln('cannot redefine escape key.'); exit end;
        Write('Enter terminator key:');

        Read(Kbd,t);
        case t of
        #0..#31: writeln('^',chr(ord('@')+ord(t)))
        else writeln(t)
        end;

        Writeln('Now, enter string:');
        s:='';
        Repeat
              Read(Kbd,k);
              if k<>t then begin
                 case k of
                 #0..#31: write('^',chr(ord('@')+ord(k)))
                 else write(k)
                 end;
```

```
                s:=s+k
            end;
        Until (k=t) or (length(s)=255);
        SoftkeyAssign(c,s);
        ClrScr;
        LowVideo;
        Present;
        SoftkeyEnable;
end;

begin
     SoftkeyInit;
     Write(#27,'y?',#27,'x1');
                            {Z100 disable key expansion and enable 25th line}
     ClrScr;
     LowVideo;
     Present;
     repeat
           Read(Kbd,c);
           if c = ESC then redefine else write(c);
     until c=^C;
     Writeln(#27, z );               {Z100 Reset}
     SoftkeyExit;
end.
```

Listing 2

```
{*****************************************************************
  softkey: this file contains the Turbo Pascal 3.0 definitions
  necessary for creating, deleting, and using programmable keys.
  ****************************************************************
                      This module by:
                   Ronald J. Perrella
                     Public Domain 1987

This assumes that a coding scheme is used to convert function
keys into single character codes.  On the Z100, use the following:

write(#27,'y?'); (* disable key-expansion *)

}

Type
longstring = string[255];
softkeyptr = ^softkey;
softkey    = longstring;

softkeyTblPtr = ^softkeyTbl;
softkeyTbl    = Array[#0..#255] of Softkeyptr;

Var
sk          : softkeyTblPtr;      {To conserve Data segment memory}
translating : Boolean;
SkprevIn,
SkprevSt    : Integer;           {Previous values of I/O vectors}

(*
 *           =-=-=-= HIDDEN PROCEDURES AND FUNCTIONS =-=-=-=
 *)

(* softkeyGet function replaces the standard ConIn function*)

function softkeyGet:char;
Type
regpack     = record
                    ax,bx,cx,dx,bp,si,di,ds,es,flags:integer
                        end;
Var
register    : regpack;


Const {Static Variables}
Index       : Byte = 0;
len         : Byte = 0;
key         : char =    ;
```

stabilize everything by trying out a sample program that uses this package (see Listing 2).

The program is called "softkey_Example" and allows you to try out all of the features of the softkey package. Jump down to the main program. Notice that all executable code is between a *softkeyInit* and *softkeyExit*. This is necessary to use the package. The next statement is a *write* and it is used to disable Z-100 key expansion and enable the 25th line. The screen is cleared and a loop is entered to allow you to type any character you wish. However, if an ASCII escape character is pressed, the *redefine* procedure is called. All other characters are echoed to the screen.

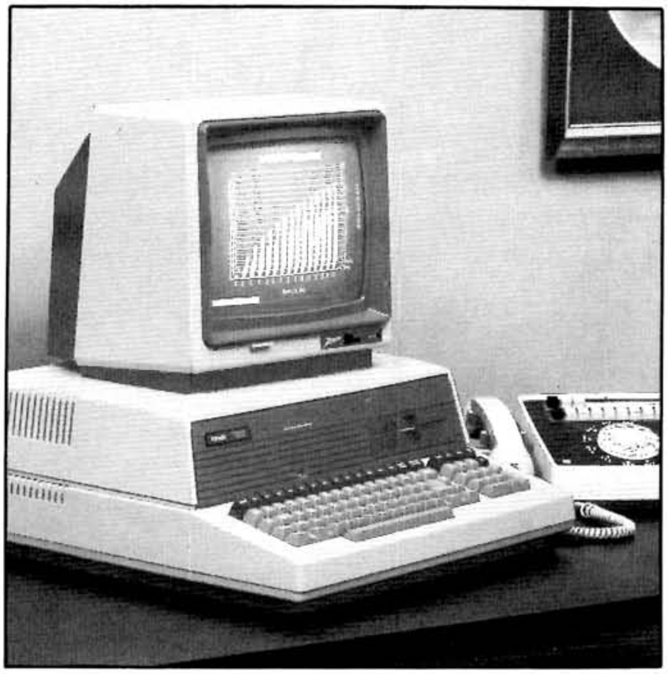
Check out the *redefine* procedure. It is used to allow you to redefine keys on the fly! *Redefine* lets you enter the key you wish to redefine (which can be anything but the escape character), and then the string you wish to have that key correspond to. Probably the easiest way to understand that procedure is to type in the sample program and watch it run. When you try to redefine a key, keep the listing handy and see how the program reacts. Everything is really much easier than it looks.

I hope that you will find this package useful and I wish you the best of luck.

Ronald J. Perrella is a Junior at Rose-Hulman Institute of Technology majoring in Computer Science. His interests include playing violin, karate, and programming, of course. He will be happy to answer any questions about his package.

```
Label top;  {necessary evil}
begin
     top:
     if not translating then begin
        {perform a DOS function to read keyboard without echo}
        {Can t use ''read(kbd,c)   anymore }
         with register do begin
              ax := $0700;        {Direct console input}
              Msdos(register);    {Run through MSDOS}
              key:=chr(lo(ax))    {key := AL register}
         end;

         if sk^[key] = Nil then          {There is no entry in table so}
            softkeyGet:=key              {just return input key}
         else begin {we have a string to translate...}
             Translating :=True;         {We are now translating}
             Index :=1;                  {Start at 1st character}
             Len := ord(sk^[key]^[0]);   {Only need this once per string}
             Goto Top                    {Makes it easy}
         end
     end else {translating} begin
         if (index <= len) and (index >0)  then begin
            softkeyGet:=sk^[key]^[index];   {get character at index in string}
            index:=succ(index)              {increment index}
         end else begin
            translating:=false;          {Stop translation}
            Goto Top                     {see above}
         end
     end
end;

(* softkeySt function replaces the standard ConSt function *)

function SoftkeySt:boolean;
Type
regpack      = record
                     ax,bx,cx,dx,bp,si,di,ds,es,flags:integer
               end;
var
register : regpack;
begin

     if Translating = True then
          softkeySt:=true    {a character is waiting}
     else
          with register do begin
            ax := $0B00;        {Check keyboard Status}
            Msdos(register);    {Can t use KeyPressed here}
            if (lo(ax)=$FF) then softkeySt:=true else softkeySt:=false;
          end
end;

(*
 *      =-=-=-=-= VISIBLE FUNCTIONS AND PROCEDURES =-=-=-=-=
 *)

(*
 * The softkeyDelete procedure removes a softkey definition
 * by de-allocating the memory and finally setting the softkey
 * to its original value. If there is no existing definition,
 * no action is taken.
 *)

procedure softkeyDelete(skey:char);
begin
     if sk^[skey] <> nil then begin
          freeMem(sk^[skey],length(sk^[skey]^)+1 ); {+1 for length byte}
          sk^[skey]:=nil
     end
end;

(*
 * The softkeyAssign procedure assigns a string to a key thus
 * making it a softkey. It allocates a piece of memory just
 * big enough to contain the string.  If a softkey is all ready
 * existing, it is deleted and the new one replaces the previous
 * definition.
 *)

procedure softkeyAssign(skey:char; s:longstring);
begin
     softkeyDelete(skey);
     getmem(sk^[skey],length(s)+1); {+1 for length byte}
     sk^[skey]^:= s    {move string}
end;

(*
 * The softkeyEnable procedure resets the vectors and enable softkey
 * capability.
 *)

procedure softkeyEnable;
begin
     memw[seg(ConInPtr):ofs(ConInPtr)] := ofs(SoftkeyGet);
     memw[seg(ConStPtr):ofs(ConStPtr)] := ofs(SoftkeySt)
end;

(*
 * The softkeyDisable procedure resets the vectors and disables softkey
 * capability.
 *)

procedure softkeyDisable;
begin
     memw[seg(ConInPtr):ofs(ConInPtr)] := skprevin;
     memw[seg(ConStPtr):ofs(ConStPtr)] := skprevst
end;

(*
 * The SoftkeyInit procedure initializes all variables, allocates
 * the memory for the softkeyTbl and redefines the Input/Output
 * drivers.
 *)

procedure softkeyInit;     {Replace Turbo s ConIn AND ConSt routines.}
var c:char;
begin
     new(sk);                                        {Make table}
```

Continued on Page 26



Pat Swayne
HUG Software Engineer

ZPC Update #23

This is the twenty-third in a series of articles in support of ZPC, a program that allows you to run IBM PC software in H/Z-100 (dual processor) computers. ZPC is available from HUG as part no. 885-3037-37. An upgrade disk for ZPC is also available as part no. 885-3042-37.

In this ZPC Update, I will present a patch to ZPC itself that makes it run faster in the monochrome (640×200) graphics mode. I will also present patches for a release of Generic CADD not previously seen, and a program that allows FoxBase Plus to run under the normal version of ZPC.

Faster Monochrome Graphics

This improvement to ZPC was sent in by J. C. Collins and was modified by myself. It is apparently similar to a patch done by Paul Herman that is part of his modifications for using ZPC with a mouse and PFS: First Publisher Software.

The modifications presented here must be done to the file ZPC.ASM, and ZPC must be reassembled after they are added. For the first change, locate the first CLD instruction in the procedure CHKSCRN, and add these lines:

```
        CMP     MODE,6
        JNE     CHKN6
        CALL    COPYGM
        JMP     CHKEX
CHKN6:  CLD             ;SCAN FORWARD
```

Locate the label UPDX, and the lines before it that look like this:

```
        CMP     MODE,6          ;HI-RES MODE?
        JNZ     CHKMED          ;NO
        MOV     ES,GRAM         ;ELSE, POINT TO GREEN RAM
        PUSH    AX              ;SAVE PIXEL PATTERN
        MOV     AL,WHITEB
        OUT     VRPORT,AL       ;SELECT WHITE
        POP     AX
        MOV     ES:[BX],AL      ;UPDATE SCREEN
UPDX:   POP     ES
```

Delete all of the lines shown above except for the last one (the one with the label UPDX), and add a jump before the label, so that you have:

```
        JMP     CHKMED          ;JUMP TO MED RES CODE
UPDX:   POP     ES
```

Following the procedure CHKSCRN, add new procedures COPYGM and DOLINE as follows:

```
;       COPY MODE 6 GRAPHICS TO Z-100 VIDEO
COPYGM  PROC    NEAR
        MOV     AL,WHITEB
        OUT     VRPORT,AL
        MOV     ES,GRAM             ;USE GREEN RAM
        MOV     DS,PCRAM            ;POINT TO PC RAM
        XOR     DI,DI
        MOV     SI,DI
        MOV     BX,SI
        CLD
NEXT2L: CALL    DOLINE              ;DO A LINE
        ADD     SI 1FB0H
        CALL    DOLINE              ;DO ODD LINE
        SUB     SI,2000H
        CMP      BX,396              ;200 LINES DONE? (2 COUNTS/LINE)
        JB      NEXT2L
COPYGM  ENDP

DOLINE  PROC    NEAR
        MOV     DI,CS:LOCTBL[BX] ;GET LOCATION IN Z-100
        MOV     CX,40
        REP     MOVSW               ;MOVE A LINE
        ADD     BX,2                ;INCREMENT TABLE POINTER
DOLINEX:RET
DOLINE  ENDP
```

After you make these changes, reassemble ZPC. It will now work faster in the monochrome graphics mode (video mode 6).

Generic CADD Patches

Here is a patch for Generic CADD version 3.0, dated 11-9-87, and a patch for the companion DOTPLOT program, dated 12-18-87.

```
GENERIC CADD version 3.0 11-9-87
Insert the disk containing CADD.EXE.
CADD.EXE
1B7A5,90
1B7AA,90
1B7B5,90
1B7B9,90
1BAD9,B0
1BB16,B0
1BB7F,B0
1BBD0,0,0
1BC38,0,0
1BCB3,0,0
z
GENERIC CADD DOTPLOT 3.0 12-18-87
Insert the disk containing DPLOT.EXE.
DOTPLOT.EXE
150C5,B0
15176,B0
151B3,B0
1521C,B0
1526D,0,0
152D5,0,0
15350,0,0
z
```

Running FoxBase Plus

In ZPC Update #21, I mentioned that FoxBase Plus is not compatible with the normal version of ZPC in the way it sizes memory, and it overwrites ZPC's emulated video memory if the work it is doing requires a lot of memory. I presented a fix in that update that allowed you to run FoxBase Plus under the small memory version of ZPC, which does not emulate video memory, but that is not a very good solution to the problem. I have since come up with a better solution. It is a program that reserves the top 64k of memory using special DOS memory allocation functions, and then runs FoxBase as a child. This solution allows you to run FoxBase Plus under the normal version of ZPC, and to run it without any patches.

The program that runs FoxBase as a child is called RUNPROG.COM, and you can create it by typing in and running this BASIC program:

```
10 PRINT "CREATING RUNPROG.COM
20 OPEN "O",1,"RUNPROG.COM":L=100
30 FOR I=1 TO 87 :C=0:FOR J=1 TO 12
40 READ B:C=C+B:PRINT #1,CHR$(B);:NEXT J:READ S
50 IF S<>C THEN PRINT "TYPING ERROR IN LINE";L:STOP
60 L=L+10:NEXT I:CLOSE #1:SYSTEM
100 DATA 186,149,3,232,152,1,180,48,205,33,60,2,1251
110 DATA 115,8,186,238,2,232,138,1,205,32,198,6,1361
120 DATA 92,0,0,198,6,108,0,0,252,191,93,0,940
130 DATA 176,32,185,11,0,243,170,191,109,0,185,11,1313
140 DATA 0,243,170,190,128,0,191,17,6,185,64,0,1194
150 DATA 243,165,190,17,6,172,10,192,117,8,186,243,1549
160 DATA 3,232,82,1,205,32,50,228,139,200,232,62,1466
170 DATA 1,116,239,191,66,3,172,60,32,116,13,60,1069
180 DATA 13,116,11,60,47,116,5,170,226,240,235,78,1317
190 DATA 78,65,73,198,5,0,116,70,81,86,136,14,922
200 DATA 128,0,191,129,0,243,164,94,89,232,15,1,1286
210 DATA 116,52,191,92,0,184,0,41,86,205,33,94,1094
220 DATA 138,4,60,32,116,17,60,44,116,13,60,59,719
230 DATA 116,9,60,61,116,5,70,226,235,235,25,70,1228
240 DATA 73,116,15,232,229,0,116,10,191,108,0,184,1274
250 DATA 0,41,205,33,235,6,199,6,128,0,0,13,866
260 DATA 250,188,17,6,251,186,195,2,180,26,205,33,1539
270 DATA 51,237,191,66,3,176,46,185,80,0,242,174,1451
280 DATA 117,29,129,61,67,79,117,9,128,125,2,77,940
290 DATA 116,59,233,93,255,129,61,69,88,117,247,128,1595
300 DATA 125,2,69,116,44,235,239,69,191,66,3,50,1209
310 DATA 192,185,80,0,242,174,79,199,5,46,67,199,1468
320 DATA 69,2,79,77,186,66,3,185,0,0,180,78,925
330 DATA 205,33,115,47,198,69,1,69,199,69,2,88,1095
340 DATA 69,185,0,0,180,78,186,66,3,205,33,115,1120
350 DATA 26,186,36,3,232,107,0,11,237,116,8,199,1161
360 DATA 5,13,10,198,69,2,36,186,66,3,232,89,909
370 DATA 0,205,32,14,31,187,98,0,180,74,205,33,1059
380 DATA 114,57,184,1,88,187,2,0,205,33,114,47,1032
390 DATA 180,72,187,255,15,205,33,114,38,184,1,88,1372
400 DATA 51,219,205,33,161,44,0,163,181,2,140,14,1213
410 DATA 185,2,140,14,189,2,140,14,193,2,186,66,1133
420 DATA 3,187,181,2,184,0,75,205,33,115,6,186,1177
430 DATA 50,3,232,13,0,205,32,128,60,32,117,3,875
440 DATA 70,226,248,11,201,195,86,139,242,252,172,10,1852
450 DATA 192,116,251,60,36,116,8,138,208,180,2,205,1512
460 DATA 33,235,239,94,195,0,0,128,0,0,0,92,1016
470 DATA 0,0,0,108,0,0,0,0,0,0,0,0,108
480 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0
490 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0
500 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0
510 DATA 0,0,13,10,7,84,104,105,115,32,112,114,696
520 DATA 111,103,114,97,109,32,114,101,113,117,105,114,1230
530 DATA 101,115,32,77,83,45,68,79,83,32,118,101,934
540 DATA 114,115,105,111,110,32,50,32,111,114,32,97,1023
550 DATA 98,111,118,101,46,13,10,36,13,10,67,97,720
560 DATA 110,39,116,32,102,105,110,100,32,36,13,10,805
570 DATA 67,97,110,39,116,32,101,120,101,99,117,116,1115
580 DATA 101,32,0,0,0,0,0,0,0,0,0,0,133
590 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0
600 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0
610 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0
620 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0
630 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0
640 DATA 0,0,0,0,0,0,0,0,0,0,13,10,23
650 DATA 36,13,10,82,85,78,80,82,79,71,32,86,734
660 DATA 101,114,115,105,111,110,32,49,46,48,46,13,890
670 DATA 10,67,111,112,121,114,105,103,104,116,32,40,1035
```

```
680 DATA 67,41,32,72,101,97,116,104,47,90,101,110,978
690 DATA 105,116,104,32,85,115,101,114,115,39,32,71,1029
700 DATA 114,111,117,112,32,49,57,56,55,46,32,32,813
710 DATA 65,108,108,32,82,105,103,104,116,115,32,82,1052
720 DATA 101,115,101,114,118,101,100,46,13,10,36,13,868
730 DATA 10,84,104,105,115,32,112,114,111,103,114,97,1101
740 DATA 109,32,114,117,110,115,32,97,110,111,116,104,1167
750 DATA 101,114,32,112,114,111,103,114,97,109,32,97,1136
760 DATA 115,32,97,32,99,104,105,108,100,44,13,10,859
770 DATA 119,105,116,104,32,116,104,101,32,116,111,112,1168
780 DATA 32,54,52,107,32,111,102,32,115,121,115,116,989
790 DATA 101,109,32,109,101,109,111,114,121,32,114,101,1154
800 DATA 115,101,114,118,101,100,46,13,10,10,84,111,923
810 DATA 32,117,115,101,32,116,104,105,115,32,112,114,1095
820 DATA 111,103,114,97,109,44,32,101,110,116,101,114,1152
830 DATA 13,10,10,32,32,82,85,78,80,82,79,71,654
840 DATA 32,60,112,114,111,103,114,97,109,62,32,91,1037
850 DATA 60,97,114,103,117,109,101,110,116,115,62,93,1197
860 DATA 13,10,10,87,104,101,114,101,32,60,112,114,858
870 DATA 111,103,114,97,109,62,32,105,115,32,116,104,1100
880 DATA 101,32,112,114,111,103,114,97,109,32,121,111,1157
890 DATA 117,32,119,97,110,116,32,116,111,13,10,114,987
900 DATA 117,110,44,32,97,110,100,32,60,97,114,103,1016
910 DATA 117,109,101,110,116,115,62,32,97,114,101,32,1106
920 DATA 97,110,121,32,99,111,109,109,97,110,100,32,1127
930 DATA 108,105,110,101,32,97,114,103,45,13,10,117,955
940 DATA 109,101,110,116,115,32,114,101,113,117,105,114,1247
950 DATA 101,100,32,98,121,32,116,104,101,32,112,114,1063
960 DATA 111,103,114,97,109,46,13,10,36,0,0,0,639
```

This program is fairly large, so I will place it (RUNPROG.COM, that is) on the HUG bulletin board (616-982-3956). To use it, enter

```
RUNPROG <PROGNAME> [<ARGUMENTS>]
```

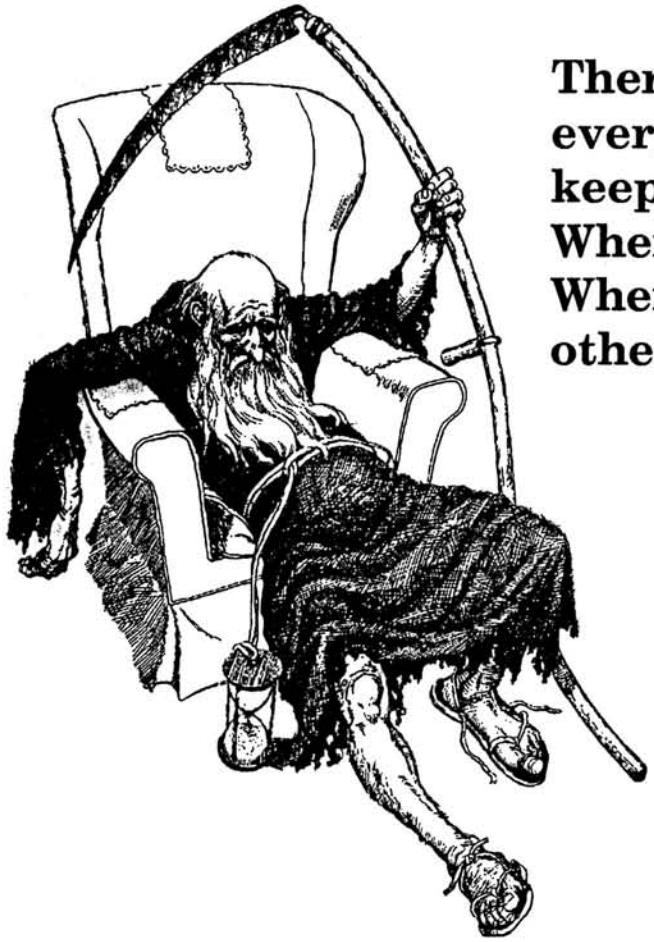
Where <PROGNAME> is the name of the program you want to run, including a path description if it is not in the default path; and [<ARGUMENTS>] are any optional arguments required by the program.

Here is the assembly source code for RUNPROG.COM.

```
        PAGE    ,132
;       THIS PROGRAM IS A SHELL THAT RUNS A PROGRAM AND
;       RESERVES THE TOP 64K OF SYSTEM MEMORY SO THAT THE
;       CHILD PROGRAM WILL NOT USE IT
;
;       TO USE THIS PROGRAM, ENTER
;
;       RUNPROG <PROGRAM> [<ARGUMENTS>]
;
;       WHERE <PROGRAM> IS THE COMPLETE PATH NAME DESCRIBING
;       THE PROGRAM YOU WANT TO RUN, AND <ARGUMENTS> ARE ANY
;       ARGUMENTS REQUIRED BY THE PROGRAM.
;
;       COPYRIGHT (C) HEATH/ZENITH USERS  GROUP 1988. ALL RIGHTS RESERVED.
;
;       BY PATRICK SWAYNE, HUG SOFTWARE ENGINEER 28-JUN-88

CODE    SEGMENT
        ASSUME  CS:CODE,DS:CODE,ES:CODE,SS:CODE
        ORG     0
        ZERO    LABEL   NEAR
        ORG     2CH
ENVSEG  LABEL   WORD                    ;ENVIRONMENT SEGMENT
        ORG     5CH
FCB1    LABEL   BYTE                    ;FIRST FCB
        ORG     6CH
FCB2    LABEL   BYTE                    ;SECOND FCB
        ORG     80H
ARG     LABEL   BYTE                    ;ARGUMENT AREA
        ORG     100H

;       SET UP FOR EXECUTION OF PROGRAM

START:  MOV     DX,OFFSET SIGNON
        CALL    PMSG                    ;PRINT SIGN-ON
        MOV     AH,30H
        INT     21H                     ;GET DOS VERSION
        CMP     AL,2                    ;TEST IT
        JNB     VERSOK                  ;VERSION OK
        MOV     DX,OFFSET BADVER
        CALL    PMSG                    ;SAY ''BAD VERSION
        INT     20H
VERSOK: MOV     FCB1,0                  ;ZERO FCB1 DRIVE
        MOV     FCB2,0                  ;AND FCB2 DRIVE
        CLD
        MOV     DI,OFFSET FCB1+1
        MOV AL,' '
        MOV     CX,11
        REP     STOSB                   ;CLEAR FCB1
        MOV     DI,OFFSET FCB2+1
        MOV     CX,11
        REP     STOSB                   ;CLEAR FCB2
        MOV     SI,OFFSET ARG           ;POINT TO USER ARGUMENT
        MOV     DI,OFFSET ENDADR        ;POINT TO FREE SPACE
        MOV     CX,80H/2
        REP     MOVSW                   ;COPY ARGUMENT THERE
        MOV     SI,OFFSET ENDADR        ;NOW, POINT TO IT HERE
        LODSB   ;GET COUNT
        OR      AL,AL                   ;ANY COUNT?
        JNZ     GOTINP                  ;YES, WE HAVE USER INPUT
```

```
NOINP:  MOV     DX,OFFSET NPMSG
        CALL    PMSG                    ;ELSE, SAY "NO PROGRAM"
        INT     2ØH

;       GET THE PROGRAM NAME

GOTINP: XOR     AH,AH                   ;MAKE COUNT A WORD
        MOV     CX,AX                   ;COUNT TO CX
        CALL    SOS                     ;SKIP SPACES
        JZ      NOINP                   ;NO USER INPUT
        MOV     DI,OFFSET PRGNAM         ;PUT PROGRAM NAME HERE
MOVPGM: LODSB                          ;GET A CHARACTER
        CMP     AL,' '                   ;END OF PROGRAM NAME?
       JZ      GOTSW                   ;YES
       CMP     AL,13                   ;COULD END IN CR
        JZ      GOTEND
        CMP     AL,'/'                  ;COULD END IN SWITCH
        JZ      GOTSW
        STOSB                           ;STORE IT
        LOOP    MOVPGM                  ;ELSE, LOOP
        JMP     NOARG                   ;RAN OUT OF CHARACTERS, NO ARGUMENT
GOTSW:  DEC     SI                      ;BACK UP TO SWITCH
        INC     CX                      ;CANCEL NEXT DEC
GOTEND: DEC     CX                      ;COUNT THE SEPARATOR
        MOV     BYTE PTR [DI],Ø         ;NULL END OF PROGRAM NAME
        JZ      NOARG                   ;NO ARGUMENT SPECIFIED

         SET UP THE COMMAND ARGUMENT AND THE FCB S

        PUSH    CX                      ;SAVE COUNT
        PUSH    SI                      ;AND LOCATION
        MOV     ARG,CL                  ;SAVE ARGUMENT LENGTH
        MOV     DI,OFFSET ARG+1          ;PUT PROGRAM ARGUMENT HERE
        REP     MOVSB                   ;MOVE ARGUMENT
        POP     SI                      ;RESTORE POINTER
        POP     CX                      ;AND COUNT
        CALL    SOS                     ;SKIP SPACES
        JZ      NOARG
        MOV     DI,OFFSET FCB1          ;POINT TO FIRST FCB
        MOV     AX,29ØØH
        PUSH    SI                       ;SAVE POINTER
        INT     21H                     ;PARSE FIRST NAME
        POP      SI
FNDTRM: MOV     AL,[SI]                 ;GET NEXT CHARACTER
        CMP     AL,' '                  ;SPACE?
        JZ      GOTTRM
        CMP     AL,' '                  ;COMMA?
        JZ      GOTTRM
        CMP     AL,';'                  ;SEMI?
        JZ      GOTTRM
        CMP     AL'='
        JZ      GOTTRM
        INC     SI
        LOOP    FNDTRM                  ;LOOK FOR TERMINATOR
        JMP     SHORT GOTARG            ;NONE FOUND, NO 2ND ARG
GOTTRM: INC     SI                      ;PASS OVER TERMINATOR
        DEC     CX                      ;COUNT IT
        JZ      NOARG                   ;THAT WAS THE END
        CALL    SOS                     ;SKIP ANY SPACES
```

```
        JZ      NOARG                   ;ONLY SPACES LEFT
        MOV     DI,OFFSET FCB2          ;POINT TO SECOND FCB
        MOV     AX,29ØØH
        INT     21H                     ;PARSE SECOND NAME
        JMP     SHORT GOTARG
NOARG:  MOV     WORD PTR ARG,ØDØØH      ;INSERT NULL COUNT, CR
GOTARG: CLI
        MOV     SP,OFFSET ENDADR        ;PUT STACK HERE
        STI

        PROCESS PROGRAM NAME

        MOV     DX,OFFSET FNDBUF
        MOV     AH,1AH
        INT     21H                     ;SET DTA TO FIND BUFFER
        XOR     BP,BP                   ;CLEAR EXTENSION FLAG
        MOV     DI,OFFSET PRGNAM        ;POINT TO PROGRAM NAME
        MOV     AL,'.'
        MOV     CX,8Ø
        REPNZ   SCASB                   ;LOOK FOR "."
        JNZ     NOEXT                   ;NO EXTENSION SPECIFIED
        CMP     WORD PTR [DI],'OC'      ;LOOK FOR "COM"
        JNZ     NOTCOM                  ;NOT "COM"
        CMP     BYTE PTR 2[DI],'M'
        JZ      LOOK2                   ;WE HAVE COM
NOINPJ: JMP     NOINP                   ;EXPLAIN PROGRAM
NOTCOM: CMP     WORD PTR [DI],'XE'      ;LOOK FOR "EXE"
        JNZ     NOINPJ                  ;NOT "EXE"
        CMP     BYTE PTR 2[DI],'E'
        JZ      LOOK2                   ;EXTENSION SPECIFIED, LOOK FOR FILE
        JMP     NOINPJ
NOEXT:  INC     BP                      ;FLAG EXTENSION SUPPLIED
        MOV     DI,OFFSET PRGNAM
        XOR     AL,AL
        MOV     CX,8Ø
        REPNZ   SCASB ;ELSE, LOOK FOR END
        DEC     DI                      ;POINT TO IT
        MOV     WORD PTR [DI],'C.'      ;MOVE IN .COM
        MOV     WORD PTR 2[DI],'MO'
        MOV     DX,OFFSET PRGNAM
        MOV     CX,Ø
        MOV     AH,4EH
        INT     21H                     ;LOOK FOR PROGRAM
        JNC     GOTPRG                  ;GOT IT
        MOV     BYTE PTR 1[DI],'E'      ;ELSE, MAKE PROGRAM .EXE
        MOV     WORD PTR 2[DI],'EX'
LOOK2:  MOV     CX,Ø
        MOV     AH,4EH
        MOV     DX,OFFSET PRGNAM
        NT     21H                     ;LOOK FOR PROGRAM
        JNC     GOTPRG                  ;GOT IT
        MOV     DX,OFFSET NOFIND
        CALL    PMSG                    ;ELSE, SAY "CAN T FIND"
        OR      BP,BP                   ;EXTENSION SUPPLIED?
        JZ      NOEXS                   ;NO
        MOV     WORD PTR [DI],ØAØDH     ;ELSE, INSERT CR,LF
        MOV     BYTE PTR 2[DI],'$'
NOEXS:  MOV     DX,OFFSET PRGNAM
        CALL    PMSG                    ;PRINT FILE NAME
        INT     2ØH
GOTPRG:
;       PREPARE TO RUN THE PROGRAM

PRPRUN: PUSH    CS
        POP     DS                      ;FIX DS
        MOV     BX,(ENDADR-ZERO+15) SHR 4 ;POINT TO END PARA.
        MOV     AH,4AH
        INT     21H                     ;FREE UP MEMORY FOR PROGRAM TO RUN
        JC      CNTEX                   ;SOMETHING S WRONG
        MOV     AX,58Ø1H
        MOV     BX,2
        INT     21H                     ;SET ALLOCATION TO LAST FIT
        JC      CNTEX
        MOV     AH,48H
        MOV     BX,ØFFFH
        INT     21H                     ;RESERVE 64K AT TOP
        JC      CNTEX
        MOV     AX,58Ø1H
        XOR     BX,BX
        INT     21H                     ;RESTORE ALLOCATION TO NORMAL
        MOV     AX,ENVSEG
        MOV     PBENV,AX                ;SET UP ENV. SEGMENT ADDRESS
        MOV     PBCMD+2,CS              ;SET UP SEGMENT FOR CMD LINE
        MOV     PBFCB1+2,CS             ;SET UP FCB1 SEGMENT
        MOV     PBFCB2+2,CS             ;SET UP FCB2 SEGMENT

;       RUN THE PROGRAM

        MOV     DX,OFFSET PRGNAM        ;POINT TO PROGRAM NAME
        MOV     BX,OFFSET PBLOCK        ;AND TO PARAMETER BLOCK
        MOV     AX,4BØØH                ;PROGRAM EXEC FUNCTION
        INT     21H                     ;TRY TO EXECUTE PROGRAM
        JNC     EXECOK                  ;IT WAS OK
CNTEX:  MOV     DX,OFFSET NOEXEC
        CALL    PMSG                    ;SAY "CAN T RUN"

;       ALL DONE, EXIT

EXECOK: INT     2ØH

;       SUBROUTINES
;       SKIP OVER SPACES

SOS:    CMP     BYTE PTR [SI],' '       ;SPACE?
        JNZ     NOTSP                   ;NO
        INC     SI                      ;ELSE, SKIP IT
        LOOP    SOS
NOTSP:  OR      CX,CX                   ;SET FLAG ON COUNT
        RET

;       PRINT MESSAGES, SKIPPING ZEROS

PMSG:   PUSH    SI
        MOV     SI,DX                   ;POINT TO MESSAGE
        CLD
PMSGLP: LODSB                           ;GET A CHARACTER
        OR      AL,AL                   ;ZERO
        JZ      PMSGLP                  ;SKIP IT
        CMP     AL,'$'                  ;END?
        JZ      PMSGX
        MOV     DL,AL
        MOV     AH,2
        INT     21H                     ;PRINT CHARACTER
        JMP     PMSGLP
PMSGX:  POP     SI
        RET

;       DATA AREA

;       PARAMETER BLOCK USED TO EXECUTE PROGRAM

PBLOCK  LABEL   NEAR
PBENV   DW      Ø                       ;ENVIRONMENT SEGMENT
PBCMD   DW      8ØH,Ø                   ;COMMAND LINE ADDRESS
PBFCB1  DW      5CH,Ø                   ;FCB1 ADDRESS
PBFCB2  DW      6CH,Ø                   ;FCB2 ADDRESS

FNDBUF  DB      43 DUP (Ø)              ;FIND BUFFER

BADVER  DB      13,1Ø,7,'This program requires MS-DOS version 2 or '
        DB      'above.',13,1Ø,'$'
NOFIND  DB      13,1Ø,"Can t find $"
NOEXEC  DB      13,1Ø,"Can t execute "
PRGNAM  DB      8Ø DUP (Ø)              ;NAME OF PROGRAM TO RUN
        DB      13,1Ø,'$'
SIGNON  DB      13,1Ø,'RUNPROG Version 1.Ø.',13,1Ø
        DB      "Copyright (C) Heath/Zenith Users' Group 1987. "
        DB      'All Rights Reserved.',13,1Ø,'$'
NPMSG   DB      13,1Ø,'This program runs another program as a child,',13,1Ø
        DB      'with the top 64k of system memory reserved.',13,1Ø,1Ø
        DB      'To use this program, enter',13,1Ø,1Ø
        DB      '   RUNPROG <program> [<arguments>]',13,1Ø,1Ø
        DB      'Where <program> is the program you want to',13,1Ø
        DB      'run, and <arguments> are any command line arg-',13,1Ø
        DB      'uments required by the program.',13,1Ø,'$'

ENDADR  EQU     $+256                   ;END OF THIS PROGRAM

CODE    ENDS
        END     START
```



ZPC Update Index

Name of Program	ZPC Update No.	REMark Issue
FIXES FOR ZPC	Fixes for ZPC	June 88
PROGRAMS THAT RUN UNDER ZPC	16	May 87
Best Menu Version 1.2	17	June 87
CED V3.20	Buggin' Hug	November 87
Certificate Maker	19	September 87
Dbase III	2	December 85
Dbase III+	8	August 86
Dbase III+ V1.1	12	December 86
Dbase III+	17	June 87
Edix	3	February 86
Enable	3	February 86
Enable Version 2	15	April 87
Fox Base Plus	21	April 88
Framework	2	December 85
Framework	13	February 87
Framework II in Color	15	April 87
GEM (Digital Research)	14	March 87
Generic CADD	12	December 86
Generic CADD	14	March 87
Generic CADD	15	April 87
GW-Basic Compiler (LPRINT Statement)	4	March 86
GW-Basic	8	August 86
IBM Basic Compiler (LPRINT Statement)	4	March 86
Javelin Version 1.1	13	February 87
Javelin Version 1.1	15	April 87
Javelin Version 1.1	17	June 87
LogiCADD V2.02	20	December 87
LogiCADD V3.00	21	April 88
Logitech (PFS) Publisher V1.0	20	December 87
Lotus 1-2-3	2	December 85
Lotus 1-2-3	10	October 86
Lotus 1-2-3, Release 2	8	August 86
Lotus 1-2-3, Release 2	10	October 86
Lotus Freelance Plus	18	July 87

Lotus Symphony	9	September 86
Lotus Symphony	10	October 86
Maxthink Version 3.3	16	May 87
Microsoft Chart V2.02	16	May 87
Microsoft Word	3	February 86
Microsoft Word	9	September 86
Microsoft word Version 3.1	17	June 87
Microsoft Word (Ctrl -) Key Combo	Fixes for ZPC	June 88
Multimate	3	February 86
Multimate	4	March 86
Multimate	7	July 86
Multimate	8	August 86
Multiplan V3.0	20	December 87
Newsroom Pro	19	September 87
PC File	4	March 86
PC File Plus V1.0	21	April 88
PC Pallette	3	February 86
PC Write	4	March 86
PC Write Version 2.6	15	April 87
PFS First Publisher	20	December 87
Print Master	4	March 86
Quick Basic V1.0 (LPRINT Statement)	4	March 86
Quick Basic V1.0	7	July 86
Quick Basic V2.0	11	November 86
Quick Basic V2.0	16	May 86
Quick Basic V2.1	17	June 87
Quick Basic V4.0	21	April 88
Quick Basic C V1.0	21	April 88
Quick DOS V1.21	13	February 87
Quick DOS V2.00	21	April 88
Sublogic Jet	18	July 87
Supercalc 3	2	December 85
Supercalc 3	5	April 86
Supercalc 4	21	February 87
Turbo C	20	December 87
Turbo Editor Tool Box	15	April 87
Turbo Pascal 3.01	4	March 86
Volkswriter Deluxe	3	February 86
Word Finder	19	September 87
Word Perfect 4.1	4	March 86
Word Perfect 4.1	9	September 86
Wordstar Version 4.0	19	September 87
Watfor Fortran	7	July 86

ZHS Hardware Support

Original Circuit	5	April 86
Improvement	9	September 86
Woes	10	October 86
Solution	11	November 86
Improved Circuit	12	December 86
Video Port Help	13	February 87
ZHS can be made to work!	Buggin' HUG	October 87

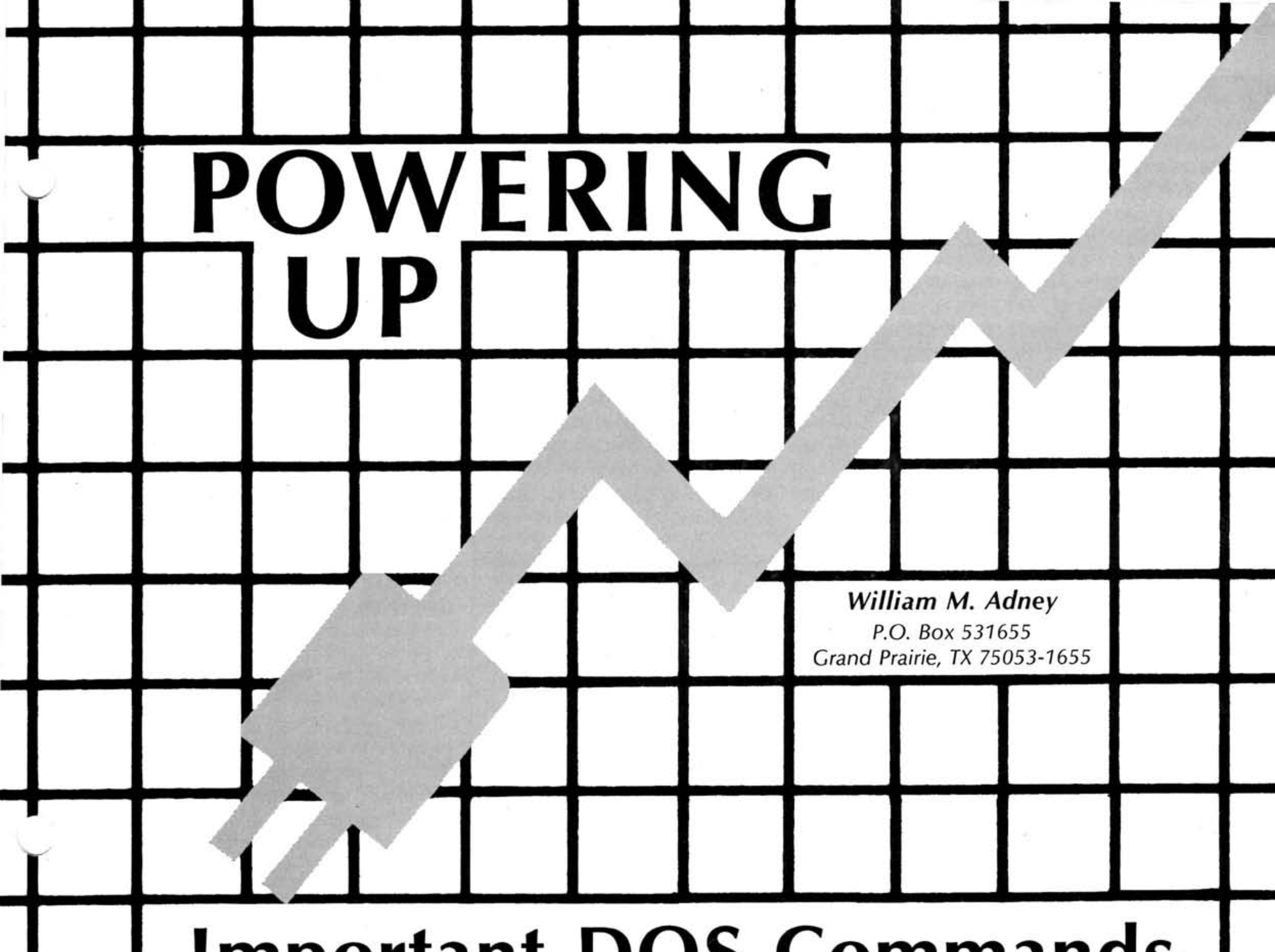
Source Codes (Patches, etc...)

Name of Program	To Fix/Patch	Update No.	REMark Issue
BLINK ASM	Non Blinking Cursor	9	September 86
BLINK.BAS	Non Blinking Cursor	9	September 86
DOS3.DAT	ZPC3.COM 9/15/85	3	February 86
DOS3.DAT	ZPC3.COM 9/9/85	3	February 86
DOS3.DAT	ZPC3.COM 9/11/85	3	February 86
DOS3.DAT	ZPC3.COM 9/18/85	3	February 86
DOS3.DAT	ZPC3.COM <9/19/85	2	December 85
DOS3.DAT	ZPC3.COM 9/19/85	2	December 85
DOS3.DAT	ZPC3.COM 9/19/85	3	February 86
DOS3.DAT	ZPC3.COM 10/4/85	2	December 85
DOS3.DAT	ZPC3.COM 10/4/85	3	February 86
FIXBRK.DAT	Control/Break Patch	8	August 86
FIXCON.ASM	Console I/O	7	July 86
FIXIFF.ASM	Interrupt FF	7	July 86
FIXLTS.ASM	Lotus	8	August 86
FIXLTS.BAS	Lotus	8	August 86
FIXLTS.ASM	Lotus	10	October 86
FIXLTS.BAS	Lotus	10	October 86
FIXMEM.ASM	Unprotect upper 64K	21	April 88
FIXMEM.BAS	Unprotect upper 64K	21	April 88
FIXWI.BAS	Width of Screen	13	February 87
FIXZPC.DAT	ZPC3.COM 9/5/85	3	February 86
FIXZPC.DAT	ZPC3.COM 9/9/85	3	February 86
FIXZPC.DAT	ZPC3.COM 9/11/85	3	February 86
FIXZPC.DAT	ZPC3.COM 9/18/85	3	February 86
FIXZPC.DAT	ZPC3.COM <9/19/85	2	December 85
FIXZPC.DAT	ZPC3.COM 9/19/85	2	December 85
FIXZPC.DAT	ZPC3.COM 9/19/85	3	February 86
FIXZPC.DAT	ZPC3.COM 10/4/85	3	February 86
FIXZPC.DAT	ZPC3A.COM	3	February 86
INT14.ASM	Interrupt 14	14	March 87
KEY.ACM	Ctrl (-) Key Combo Fixes for ZPC	June 88	
KEYINT.BAS	Keyboard Interrupt	5	April 86
KEYINT.ASM	Keyboard Interrupt	5	April 86
M2TO3	Map Video Mode 2 to 3	15	April 87
PATCHES	ZPC3.COM <10/4/85	2	December 85
PRNFIX.BAS (Concerns LPRINT in compiled BAS)		4	March 86
PRNFIX.ASM (Concerns LPRINT in compiled BAS)		4	March 86
RUNFOX	Run Foxbase +	21	April 88
VIDCON.BAS	Video Control Port	13	February 87
ZHSTST.ASM	Tests ZHS Board	9	September 86
ZHSTST.BAS	Tests ZHS Board	9	September 86

Bugs, Problems and General Information

Subject	Update No.	REMark Issue
8259's Slow	9	September 86
Control/- Keystroke Combination	Fixes for ZPC	June 88
Control/Break Patch	8	August 86
Control/Break Patch Correction	10	October 86
Cursor-Steady/Non-Blinking	9	September 86
Memory usage discussion	3	February 86
Mice and Serial Ports	10	October 86
Patcher Bug	7	July 86
Port 60 Is Bad	9	September 86
Program Hangs Up	9	September 86
Serial Ports and Mice	10	October 86
SETZPC Bug	7	July 86
SETZPC Bug Correction from Update #7	9	September 86
Video Control Port	13	February 87
Video Mode-Map Mode 2 to Mode 3	15	April 87
Video Port Bad	9	September 86
Width of Screen	13	February 87
Wild Interrupts	9	September 86
ZPC Bug Patches	15	April 87
ZPC Version 2.1 Fixes/Ctrl (-) Key Combo	Fixes for ZPC	June 88

✱



POWERING UP

William M. Adney
P.O. Box 531655
Grand Prairie, TX 75053-1655

Important DOS Commands You Must Know

If you have been reading this series of articles, you already know some of the important DOS commands: DIR, DEL, PATH, and various commands used with subdirectories. Up to this point, all of these commands have been part of the operating system (i.e., DOS), and we have called these internal commands for that reason. These commands cannot be found on a disk because they are included in the files loaded into memory when DOS is booted. Since they are always available (the technical term is "resident" in memory), you need not worry about preceding an internal command with a drive or path specification.

In this article, we will concentrate on some important external commands that you MUST know in order to run your computer system. These commands are: FORMAT, CHKDSK, and DISKCOPY. In addition, you will learn about some details that are necessary to use these commands correctly and at the proper time. For one reason or another, files and sometimes entire disks get "corrupted", and you will learn how to fix some of those problems with the CHKDSK command. You will also learn various ways to use the internal COPY command that can provide a number of ways to accomplish various tasks. Of all commands mentioned thus far, there is one command that you must know — the FORMAT command.

The FORMAT Command

FORMAT is the most important of all the external commands. It is the single command that every DOS user must know because it is used to initialize new disks — both floppy and hard disks. How you enter the FORMAT command also determines whether you will have a bootable disk (i.e., a system disk) or a data disk that has been discussed in a previous article. The basic FORMAT command syntax is shown in Figure 1.

```
FORMAT d:          (Data disk)
FORMAT d:/S        (Bootable disk)
```

Figure 1
FORMAT Command

The first syntax shown in Figure 1 is used to initialize a data disk for use with any application program, such as a word processor or a spreadsheet. For example, you can use this syntax to FORMAT a disk in drive B by entering the following command:

```
A:\ ==>FORMAT B:
```

Although most FORMAT programs included with most DOS versions do not require the drive specification on the command line, I strongly recommend that you get into the habit of always entering the drive letter as shown. The reason is that some DOS versions will FORMAT the default drive (A, in this example) and destroy everything on your system disk. If you get into the habit of always entering the drive letter on the command line, it forces you to think about what drive you really want to FORMAT and helps avoid the problem of losing programs or data on the default drive.

The second line of the syntax in Figure 1 is used to initialize a system (i.e., bootable) disk. Note that the /S switch is used to tell the FORMAT program that you want to create a SYSTEM disk that can be used to boot your computer. To initialize a system disk on drive B using this syntax, you would enter the command as follows:

```
A:\ ==>FORMAT B:/S
```

That is all there is to using the FORMAT command. Remember that all external commands, like FORMAT, may be preceded by an optional drive and/or path specification as mentioned in the previous article. Or, you may have a PATH command that includes the path specification(s) to be searched by DOS.

Now that you know how to use FORMAT, let's take a look at when you need to use it.

When to Use the FORMAT Command

A friend called me one time a few years ago and told me that he was losing files on his computer. Every time he started up his computer and inserted the data disks, files that he had created in the last session had disappeared. I asked him to describe what he was doing, and from his description, it sounded like he was doing the proper things in the correct order. But he was still losing his data files. What happened?

The problem was that he thought a data disk had to be initialized with the FORMAT command EVERY time it was inserted into the floppy drive B. Each time he started his computer system, he formatted his data disk and destroyed all of the files created in the previous session. It is this kind of misunderstanding that I often see which convinces me that a little technical knowledge can save a whole lot of grief and time. The answer to this question is based on an understanding of when you must FORMAT a disk.

When you buy a brand new box of floppy disks, all of them MUST first be initialized with the FORMAT command before they can be used in your computer system. That is the ONLY time that you must initialize a disk — once a disk is initialized, there is normally no reason why you need to FORMAT it again because the original FORMAT is usually good for the life of the disk. But there is one exception to this.

As you use your computer system, you may find that at some point you will want to consolidate files from two or more disks on to one disk. Once those files have been copied to a single disk, there is probably no reason to keep them scattered on the other disks. One easy way to "erase" those files is to simply use the FORMAT command on those old disks. The important point to remember is that FORMAT completely DESTROYS all existing files on a floppy disk so you want to be sure that is what you want before you start.

What FORMAT Really Does

The FORMAT program is used to initialize a new disk, but what does that really mean? During the FORMAT process, the program actually writes a "skeleton" of basic information to the disk. It is important to understand what this skeleton consists of because DOS can generate a number of seemingly cryptic error messages if you don't know what it is trying to tell you. Perhaps you have heard someone refer to a "bad sector", and although it may have been clear that the term referred to a "bad spot" on the disk, you may not have been sure exactly what the implications were. To understand what FORMAT really does, let's start with some terminology for a disk. We'll assume a standard 5.25" double-sided, double-density (DS/DD) floppy disk with the usual 360 kilobytes (360 KB) of storage capacity for this discussion although the same principles apply to all disks.

For data storage purposes, each side of the disk is divided into a series of concentric circles that are called TRACKS. Our standard floppy disk has 40 tracks on each side, and you will sometimes see a specification that says these are 48 TPI (Tracks Per Inch) disks. This is sometimes confusing, but since we only use 40 tracks, the actual data recording surface is 40/48" or 5/6" wide on the disk. By convention, these tracks are numbered beginning with zero so the last track is actually 39.

If you are using a current DOS version (e.g., 3.20), you will find that FORMAT displays an information message that continually changes during formatting and looks like the following:

```
Head: Ø   Cylinder: 22
```

Since this is a double-sided disk, the two sides are numbered zero and one. During the FORMAT, the Head number will continually change between zero and one indicating which side is being formatted. The bottom side of the floppy is side zero, and the top is side 1. That's fine so far, but what is a cylinder?

Remember that there are 40 tracks on each side of our example disk. If you consider an imaginary surface drawn between tracks of the corresponding number (e.g., track 0) on each side, that imaginary surface can be described as a CYLINDER. Cylinders follow the same numbering convention as tracks — that is, they start at zero, and in this example, the last cylinder is 39. If you carefully watch the FORMAT display, you will find that it will complete formatting when you see:

```
Head: 1   Cylinder: 39
```

This technique, called cylinder mapping, is used because the read/write heads on a disk drive (including a hard disk) are attached to a single movable arm so that when one head moves, they all move. Although this is not too important for floppy disks, it can be very important on a hard disk because you can minimize head movement (and improve file access speed) by using a program that "optimizes" (i.e., reorganizes) the hard disk. You will learn more about that in the article on "Selecting and Using a Hard Disk".

In addition, the FORMAT program divides each track into a pie-shaped wedge called a SECTOR. Our standard 360 KB floppy disk is divided into 9 sectors, and each sector can store 512 bytes (i.e., characters) of data. The number of sectors can vary depending on the type of disk being formatted — a typical hard disk, for example, has 17 sectors. Each sector is numbered for identification purposes.

As you may know, magnetism is used to record data on a disk. Sometimes the magnetic properties of the disk can be "scrambled" so that data recorded in a specific place (i.e., a sector) cannot be read. When this happens, the disk is said to have a "bad sector" because the computer cannot read or write data to that specific location.

At this point, you have seen that FORMAT actually sets up and defines each cylinder, track, and sector; but that is only the beginning. Now that the basic "format" of tracks, cylinders, and sectors is established; the FORMAT program also writes some other special information to the disk so that DOS will be able to use it. This special information is written in an "outline" format such that the disk is divided into four separate areas as shown in Figure 2.

```
I.   Boot Sector
II.  File Allocation Tables (FATs)
III. Directory
IV.  File (or data) area
```

Figure 2
Disk Areas Created by FORMAT

The BOOT SECTOR is exactly one sector long (i.e., 512 bytes) and contains information about the exact type of disk format that is being used among other things.

The File Allocation Table (called a FAT) is used to define the various storage locations of data and programs on the disk. There are two FATs on the disk so that, if one becomes unreadable or otherwise unusable, DOS can use the other FAT. The fact that there are two FATs — each is an exact duplicate of the other — illustrates the point that the FAT is critical in locating data on a disk, and that's why there is a "backup" copy of the FAT.

The Directory is a list of all of the files on the disk which includes the file name, date and time when the file was last written, and the starting location of the file on the disk among other things.

Last, but not least, is the Data area which is where all of the file data is actually stored.

As you can see, the FORMAT program really does a lot of things. First, it defines and sets up tracks, cylinders, and sectors. Then, it creates the outline format for the disk in the form of the boot sector, FATs, Directory, and Data area.

By now, you may be wondering why all of this information is important. The answer is there are a number of things that can go wrong with a disk (such as a bad sector), and if you understand some of these terms, you will be able to understand some of the information and error messages generated by DOS programs, and you may even be able to actually "fix" some problems using the CHKDSK command.

The CHKDSK Command

CHKDSK is probably the second most important command to know because it can tell you about the status of a disk, some information about your system, and correct some common disk problems. Like FORMAT, CHKDSK is also an external command, and its syntax is shown in Figure 3.

```
CHKDSK [d:]
```

Figure 3
CHKDSK Command

If you don't enter the optional drive letter, CHKDSK will check the default drive's directory and FATs for errors, and report the information to you. For example, you could enter the following command:

```
A:\ ==>CHKDSK B:
```

If you have a system disk in drive A, you might see something like the following:

```
362496 bytes total disk space
362496 bytes available on disk

655360 bytes total memory
583184 bytes free
```

This example disk was initialized using the "FORMAT B:" command without the /S (to transfer the system) switch. This is an example of the storage capacity of a DATA disk that we have discussed. Also notice that CHKDSK reports the total amount of system memory, as well as the number of bytes (free) that can be used by a program. My system has the 640 KB maximum memory as reported by CHKDSK. If you are wondering why CHKDSK reports 655,360 bytes total memory, it is important to know that one kilobyte (KB) is defined as 1024 bytes (not 1,000) for both disk and memory storage capacity. If you divide 655,360 by 1,024, you will find that the result is exactly 640 KB as advertised.

Now let's see what CHKDSK reports when we take a look at a disk initialized with the "FORMAT B:/S" command. We will use the "CHKDSK B:" command as before, and the report will be:

```
362496 bytes total disk space
 48128 bytes in 2 hidden files
 24576 bytes in 1 user files
289792 bytes available on disk

655360 bytes total memory
583184 bytes free
```

Notice that there are only 289,792 bytes available on this disk (because we used the /S switch to create a bootable disk) as compared with 362,496 bytes on the previous data disk example. This illustrates why you do not want to use the /S switch to create a data disk — the system files required to make the disk bootable require a total of 72,704 bytes of valuable space that could better be used for data. Also notice that CHKDSK tells you that 48,128 bytes are used by 2 hidden files (the BIOS and the System Kernel), and 24,576 bytes in 1 user file (COMMAND .COM).

If you decide to try this exercise on your own system, you may find that the numbers in the examples used above will vary depending on what DOS version you are using. Zenith's MS-DOS version 3.20 was used to create these examples, but you will find similar numbers in your DOS version.

You may run CHKDSK sometime and find that the following report is displayed:

```
362496 bytes total disk space
 10240 bytes in bad sectors
352256 bytes available on disk

655360 bytes total memory
583184 bytes free
```

As you can see, this floppy disk has over 10,000 bytes in bad sectors, and if you see this kind of display (FORMAT has a similar one) for a brand new disk, I recommend filing it in the circular file. Floppy disks are quite inexpensive, and I don't like to take the risk of using one that has obvious defects when it is new — I do not trust it to safely store my data. Yes, I really do throw bad disks away, but I have kept this one (which is prominently marked as BAD) to use for demonstration purposes such as this.

Up to now, we have only looked at the usual kind of CHKDSK report that indicates all is well with the data or programs stored on a disk. CHKDSK can also display a number of error messages reporting problems that it has found on a disk, and we need to take a look at a few of them.

CHKDSK Error Messages

The CHKDSK program verifies and can report on (and sometimes fix) various disk-related problems. For example purposes, I will only mention a few of the most common error messages, such as:

```
3 lost clusters found in 2 chains.
Convert lost chains to files (Y/N)?
```

One of the more common causes of this problem is when a computer is rebooted (i.e., CTRL-ALT-DEL) while some application program (e.g., word processor, spreadsheet, etc.) is being run. That is why I mentioned in a previous article that you should never reboot a computer until you return to the DOS command prompt except as an absolute LAST resort. When you see this kind of error message, you are faced with the nearly certain prospect that one or more files can be lost. Although CHKDSK can start you on the path to actually fixing the problem, it is essential to understand some terminology and what has happened. To see what is going on, let's quickly review how the DOS filing system works.

We use an application program to create and name a file that is stored on a disk. When that is done, a directory entry containing that file name is added to the Directory on a disk (see Figure 2) which also contains the starting location on the disk.

You have already seen that disks are divided into sectors that contain 512 bytes of data. Because many files are much larger than 512 bytes (and also for other technical reasons), DOS manages file space in "groups" (or blocks) of sectors called CLUSTERS. The actual number of sectors which comprise a cluster can vary from one (512 bytes) to 16 sectors (8,192 bytes) depending on the type and size of disk, as well as other things.

When a disk is formatted, it is arranged in a long stream of numbered sectors by the FORMAT program. But when disk space is actually defined (the technical term is "allocated") for a given file, DOS knows the actual file location on the disk by using a cluster number (i.e., address). And when a disk is completely empty, its sequential clusters of storage space are available for storing data. As files are created and deleted on the disk, a "filing system" is needed to keep track of which clusters are used by existing files and which clusters are still available for allocation to new or changed files. In DOS, this filing system is called the File Allocation Table, or FAT, for short.

Think of the disk space as being a string of sequential cluster numbers with one entry for each cluster on the disk. The first FAT entry corresponds to the first available cluster, and the last FAT entry corresponds to the last available cluster. In short, the FAT is just a list of the clusters on a disk.

When you create a new file, DOS must first find a free cluster on the disk, and it searches through the FAT to find an empty table entry that corresponds to an unused cluster. When the file is actually saved to disk, DOS records the starting cluster number in the directory entry (along with the file name and other data) and writes a unique "end-of-file CHAIN" marker in the FAT entry (actually, both FAT entries) to show that cluster is in use. In this special case — where a file is shorter than one cluster —, it consists of a CHAIN of only that one cluster. Then DOS actually records the data in that cluster location on the disk.

That process works just fine until the file becomes too large for a single cluster, so DOS needs to allocate a second cluster for the file. Again, it searches through the FAT for a free cluster. Then DOS goes back to the first cluster's FAT entry where the old "end-of-file CHAIN" marker was and replaces it with the number of the file's second cluster followed by a new "end-of-file CHAIN" marker. This process "links" the cluster numbers together to form a chain and continues until enough file space has been allocated to store the file.

If you followed this description, you can see that the process creates a CHAIN of clusters for a file — each cluster entry in the FAT points to the next one (like a map), and the last entry for a file contains the "end of file CHAIN" marker. To be technically accurate, the directory and the FAT entries are not actually updated until the file is "closed" when it is saved to disk. If a computer is rebooted when a file is still displayed on the screen, the directory and/or the FAT may not have been updated correctly (or at all), and CHKDSK reports that problem.

Now that you see how files are allocated, let's take another look at the error message:

```
3 lost clusters found in 2 chains.
Convert lost chains to files (Y/N)?
```

First of all, there is a problem in the list of clusters associated with two chains. Remember a chain is just a list of cluster numbers associated with a given file in the FAT, so CHKDSK is telling us that there is a problem with the linkage of 3 clusters in 2 chains that are not correctly associated with a file name. In other words, there are 3 clusters of "stuff" on the disk that DOS cannot access, and that usually means one or more files is whacked up on the disk because it was not written and saved properly. Now that you see what happened, let's try to use CHKDSK to fix it.

Using CHKDSK to Fix Disk Problems

Even though the error message used in the example above shows a Y/N prompt, CHKDSK will not really try to fix any problem unless you enter a /F (Fix) switch on the command line. If the problem was reported on a disk in drive B, you would have to enter the following command:

```
A:\ ==>CHKDSK'B:/F
```

When you enter a "Y" in response to the "Convert lost chains to files" prompt, CHKDSK will create two files (one for each chain or 2 in this example), such as FILE0001.CHK. The CHK file type indicates that the file was created by the CHKDSK program.

Now that CHKDSK has fixed the disk problem, you can sometimes use the TYPE command or your word processor to look at each file with the CHK file type. The success of this trick depends on what program originally created the file. If it was a word processing file, you can generally tell what data is "missing" from which file by just looking. With some word processors, you may be able to salvage that data by simply reading that CHK file into a current document, but success with that technique depends on which word processor you use. In most cases, you can at least see what is "missing", although you may have to retype a paragraph or two. Hopefully, the original file was not corrupted, but if it was, you may also have to restore it from a backup disk.

If the file contained data from a spreadsheet or database program, or if the file is part of a program, you probably will not see too much intelligible information — in this instance, the original file may be so corrupted that you cannot use it. Unfortunately, it is also likely that you will not find the problem until you attempt to use the program or data, and the only real cure for the problem is to copy the appropriate file(s) from a backup disk.

There are a number of other directory and FAT-related problems that CHKDSK can fix. If CHKDSK displays the following:

```
Entry has bad size
```

CHKDSK with the /F switch will correct the file size in the disk directory and display the "*filename.typ* Allocation error. Size adjusted" message to indicate that it has done so.

Now that you know how DOS allocates files and how CHKDSK works, consider the following error messages:

```
filename.typ First cluster number is invalid. Entry truncated
filename.typ Has invalid cluster, file truncated
```

Using your knowledge of DOS file allocation, it should be obvious by now that all data in the file name displayed in the first error message are lost. In the second error message, CHKDSK simply truncated (i.e., cut off) the file's chain at the first invalid cluster number. These are examples of what can happen when a computer is rebooted at the wrong time or an application program has a bug, and they all cause the same problem — loss of data. Now let's look at a couple of commands that can help protect you from losing much, if any, data or program files.

The DISKCOPY Command

DISKCOPY is a very useful external command that is used to make an EXACT copy of a diskette, and its command syntax is shown in Figure 4.

```
DISKCOPY old-d: new-d:
```

Figure 4
DISKCOPY Command

Although various versions of DOS have different requirements for the DISKCOPY command, the above syntax will work with all versions. More importantly, I recommend that you always enter the old drive letter (old-d) that contains the source (original) disk and the new drive letter (new-d) that contains the new backup disk.

Most of the DISKCOPY programs furnished with many DOS versions will only allow you to use the program to copy files to IDENTICAL media. That is, you cannot use DISKCOPY to copy files from a hard disk to a 5.25" disk and vice versa.

For the most part, I suggest that you only use DISKCOPY to make exact backups of various programs that you buy, including the DOS distribution disks. Perhaps the best reason for that suggestion is that DISKCOPY creates an exact duplicate of the source disk, and an exact duplicate also means that bad sectors will also be "copied" (if there are any) from the source disk. Another reason for that suggestion is that it is usually much faster to use the COPY command to copy individual files to a backup disk.

The COPY Command

Unlike the other commands discussed in this article, COPY is an internal DOS command which allows you to use it any time regardless of the current drive or directory path. The syntax for the command is shown in Figure 5.

```
COPY [d:][\oldpath]oldafn

             [d:][\newpath]newafn
```

Figure 5
Basic COPY Command

Notice one of the features of the COPY command is that you can use an ambiguous file name (afn) in both the old and new file specifications. For example, you can use the following command to backup all DOC file types on one disk to a corresponding BAK file type on another:

```
A:\ ==>COPY A:*.DOC B:*.BAK
```

Or, you can use that syntax to effectively "rename" a file during the COPY process such as:

```
A:\ ==>COPY A:MYFILE.DOC B:YOURFILE.BAK
```

There is an easier way if you use an abbreviated form of the command as shown in Figure 6.

```
COPY [d:][\oldpath]oldafn d:[\newpath]
```

Figure 6
Abbreviated COPY Command

When you use this form of the command, it is not necessary to specify a new file name because the old file name(s) are copied to the destination disk with the same names. For example, you might want to copy all of your DOC files to a backup disk by using:

```
A:\ ==>COPY *.DOC B:
```

You can use wildcards in any legal combination to copy files to any valid destination — another disk or another subdirectory. This is by far the most common and most useful form of the COPY command that I have found.

There are some occasions when it is necessary to concatenate (i.e., join together) a group of files to create a single file. The best example of this technique is perhaps when a book is written and each chapter is stored as a single file name: e.g., CHAP1, CHAP2, etc. Once the book is written, it may be necessary to create a single large file when you want to run an indexing program to create an index for the book so that the page numbering comes out right. You can use the COPY command to concatenate files using the syntax shown in Figure 7.

```
COPY ufn1+ufn2+...+ufnn newufn
```

Figure 7
Concatenating Files with the COPY Command

Although each unambiguous file name (ufn) may be preceded by the optional drive and/or path specification, they have been omitted here for clarity. In order to create a single file for a book that contains a number of chapters, you could use a command like:

```
A:\ ==>COPY'CHAP1+CHAP2+CHAP3+

       CHAP4+CHAP5+CHAP6'MYBOOK
```

In this example, the new file MYBOOK will be created as a combination of the six chapters — in the EXACT order listed —, and each of the original files (e.g., CHAP1, etc.) remains intact.

There is one other interesting trick that you can do with the COPY command that may prove useful when you want to create a really short file and don't want to start your word processor to do it. For example, you want to create a CONFIG.SYS file with the following lines:

```
BUFFERS=30
FILES=25
```

It's hardly worth trying to locate your word processing disk to create that file, and here's all you need to enter:

```
A:\ ==>COPY CON CONFIG.SYS  (Press RETURN to execute)
       BUFFERS=30          (Type the line, press RETURN)
       FILES=25^Z          (Type the line, press CTRL-Z, RETURN)
          1 File(s) copied         (Information message)
```

This form of the COPY command will copy from the CON (console-CRT) device to a file which, in this case, is CONFIG .SYS. Since you don't have any real editing features (other than the Backspace key) on a line, you cannot change a line after you press RETURN. When you have typed the last line in the file, press CTRL-Z (or F6 on a PC-compatible) and RETURN to create the file.

In Summary

If you really know how to use the commands discussed here, plus the internal commands mentioned in previous articles, you can do just about anything with your system. FORMAT is used to initialize system and data disks, CHKDSK is used to verify a disk and correct some common problems, DISKCOPY is used to make backup copies of software distribution disks, and COPY can be used for a variety of different functions.

I have not attempted to show all possible variations and switches that can be used with these commands because, in some cases, there are differences depending on which DOS version you are using. If you want to know about other capabilities and uses for these commands, be sure to refer to the DOS manual for your specific version. If you would like to see more examples of the various uses of these commands and switches, along with descriptions of other error messages, you might find the MS-DOS FlipFast Reference Guide (available from HUG) of significant help.

Next Time

Few things can be quite as frustrating as trying to get a new peripheral (e.g., a printer) to work with your computer. Sometimes it is not obvious what the problem is, and next time you will learn about connecting printers, modems, and other peripherals (e.g., a mouse) to your system. We'll also take a look at how to use the Zenith CONFIGUR command, as well as the MODE and PRINT commands.

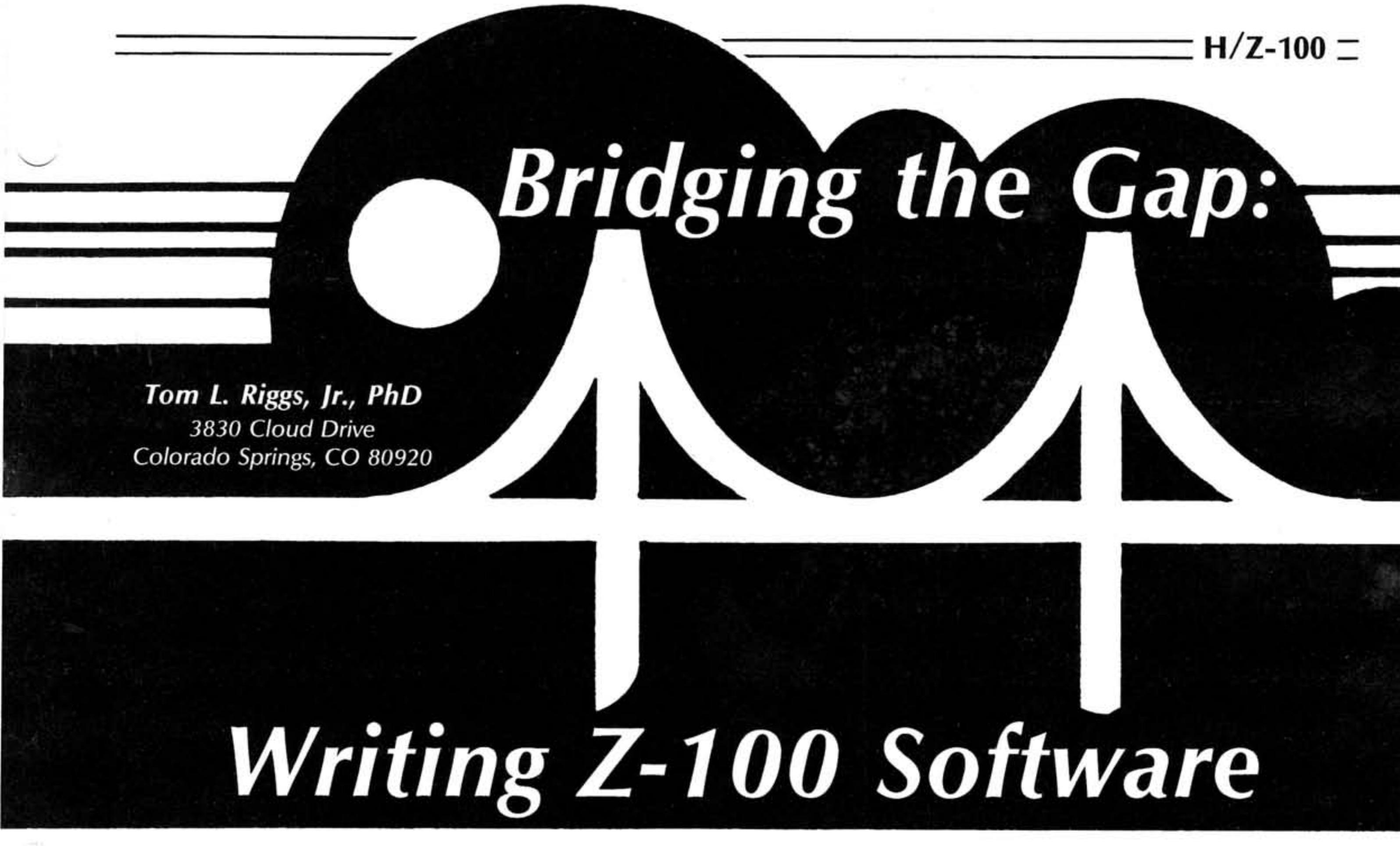
If you have any questions about anything in this column, be sure to include a self-addressed, stamped envelope (business size preferred) if you would like a personal reply to your question, suggestion or comment.

✱

Continued from Page 10

```
     skprevIn:= Memw[seg(ConInPtr):ofs(ConInPtr)];  {Save previous vectors}
     skprevSt:= Memw[seg(ConStPtr):ofs(ConStPtr)];
     softkeyEnable;            {set vectors}
     for c:=#0 to #255 do
         sk^[c] := Nil;        {Initialize all pointers to nil}
     translating := False
end;

(*
 * The softkeyExit procedure frees all of the storage that was allocated
 * during the program use and resets all vectors to their original values.
 * If you wish to use softkeys again, you must call SoftkeyInit once again.
 *)

procedure softkeyExit;
var c:char;
begin
     for c:=#00 to #255 do
         softkeydelete(c);     {Delete and free all storage}
     softkeyDisable;
     dispose(sk)               {sk is not needed any more...}
end;

(*
 * =-=-= End of Softkey.Inc =-=-=
 *)
```

✱



Bridging the Gap:

Writing Z-100 Software with Turbo Pascal 4.0

Tom L. Riggs, Jr., PhD
3830 Cloud Drive
Colorado Springs, CO 80920

Since Borland International introduced Turbo Pascal some four years ago, the language has truly become a standard. It's standard is based on popularity, a popularity that is derived from one simple principle, offer the public a widely useful and reasonably priced product of high quality. Borland has taken a language that was invented to teach computer programming and turned it into a language that appeals not only to the novice and his teacher but also to the expert programmer.

I first bought Turbo Pascal about a year after I bought my first computer, an H-100. That was five years ago. I still have and use my H-100, although presently, it's a far cry from the original kit. Just as my computer has grown in capability over these years, so have my programming skills. Turbo Pascal has been a fundamental part of my computer development. There is a key reason for this; Turbo Pascal offers the programmer the capability to access the inner workings of any computer and write high performance routines that fully exploit the power of the hardware. This capability, though not standard Pascal, is, in my mind, what makes Turbo Pascal so widely used. The high speed integrated development environment that has become the claim to fame for all of the Borland products is simply icing on the cake. Combine the flexibility of Turbo Pascal with the excellent documentation that came with the H-100 and the wealth of information provided by the HUG and you have the makings of true computer power.

Recently, Borland introduced Turbo Pascal version 4.0. In keeping with the past, this latest release is faster and more powerful than earlier versions. Unfortunately, Turbo 4.0 is designed around that other standard that has emerged over these past few years, the IBM-PC standard. The PC standard is difficult to ignore. There is simply too much neat software written for the PC. Though I and many others consider the H/Z-100 the best computer, it has one important deficiency; it is not 100% PC compatible. To correct this shortcoming, I bought the UCI EasyPC system and installed it in my H-100. Now I have the best of both worlds. Since I now dabble in both worlds, I want my software to work in both worlds. Further, I want my software to work at levels of performance not possible through strict MS-DOS conventions. To accomplish this I need some way to bridge the gap between the Z-100 world and that other world. Turbo 4.0 offers me the way. I now easily create such programs through the use of customized modules called Turbo Pascal Units (TPU's).

Turbo Pascal Units (TPU)

Turbo Pascal 4.0 provides total modularity through the use of the Turbo Pascal Unit (TPU). The TPU is simply a collection of Pascal modules that can be precompiled and used by other Pascal modules. From a functional standpoint, it makes sense to group together related and commonly used functions and routines into a single library file. Programmers have been creating and using library files for many years. However, the Turbo 4.0 TPU carries this philosophy one step further. The TPU also provides the capability to define public variable types, public constants, and even public variables. These public definitions can be optionally used by outside modules.

The structure of the TPU is very similar to the structure of a Pascal program. The general structure of a TPU is shown below.

```
UNIT Unit_Name;          (TPU Header)
INTERFACE                (Interface Section Begins Here)
    Const
                         (Any Public Constants Can Be Defined)
    Type
                         (Any Public Types Can Be Defined)
    Var
                         (Any Public Variables Can Be Defined)

    (All Public Functions and Procedure Headers are Defined Here)

IMPLEMENTATION
                         (The actual implementation of the Procedures
                         and Functions of the TPU go in this Section)

BEGIN
                         (This is the Optional Initialization Section)
END.
```

There are four parts to a Unit; the unit header, the interface section, the implementation section and the initialization section. The unit header begins with the keyword "UNIT" followed by the name of the unit. The next part of the unit, the interface section, is a new construct to Pascal. The purpose of this section is to define how the unit interfaces with the outside world. The interface section contains all the public definitions and a list of all the procedures and functions within the TPU that will be accessible from outside modules. If another unit is required to support this unit then that support unit can be declared by the keyword "USES" followed by the name of the support unit. The next part of the unit is the meat of the TPU. This part is the implementation section. The implementation section contains all the procedures and functions that you wish to place in the TPU. As a minimum, this section must contain all the code implementing the functions and procedures that were declared in the interface section. However, this section can also contain constants, types, variables, functions, and procedures that are private to the TPU. If used, these private elements are invisible to the outside world. The last section is the initialization section. This section is blocked between the reserved words "BEGIN" and "END.". This section is very similar to the main program code section in a Pascal program; however, it is optional. If the unit requires no initialization then this section will only contain the reserved word "END.".

Borland provides seven TPU's with the Turbo Pascal 4.0 compiler. The seven units are SYSTEM, DOS, CRT, PRINTER, GRAPH3, TURBO3, and GRAPH. These units are contained in a single module called a Turbo Pascal Library (TPL) and named TURBO.TPL. The TPL is constructed like any library and can be managed with a utility provided with the compiler. Though not necessary, any unit can be added or deleted from a TPL. Further, a unit does not have to reside in a TPL in order to be used. The TPL is simply another tool of convenience available to the programmer.

Of the seven units provided with the compiler only two are generic to all MS-DOS machines, SYSTEM and DOS. The other five are hardware dependent and designed specifically for PC compatible machines. This article will present a unit that will support many of the hardware functions of the Z-100 and give us the capability to write useful programs for the Z-100. In order to fully exploit the speed and capability of the Z-100, this unit will redefine how output is handled by Turbo Pascal.

Customized Input/Output Drivers

Turbo Pascal 4.0 allows the programmer to define and use his or her own hardware device drivers. To provide generality and

Listing 1

```
unit zcon;          {Turbo Pascal Unit Designed for the H/Z-100

                    Written by Tom L. Riggs, Jr.
                                3830 Cloud Drive
                                Colorado Springs, CO 80920}

interface
uses DOS;

   {Public Constants}

const                                    {Z-100 Key Codes}
    up = $0A5;
    down = $0A6;
    left = $0A8;
    right = $0A7;
    ret = $0D;
    hom = $0A9;
    can = $18;
    bell = #07;
    sht_rt = $0E7;
    sht_lt = $0E8;
    ins = $0A3;
    del = $7F;
    d_line = $0E4;
    ins_line = $0E3;
    F1 = $97;
    F2 = $98;
    F3 = $99;
    F4 = $9A;
    F5 = $9B;
    F6 = $9C;
    F7 = $9D;
    F6 = $9E;
    F9 = $9F;
    F0 = $0A0;
    blk = 0;                             {Z-100 Colors}
    red = 2;
    grn = 4;
    yel = 6;
    blu = 1;
    mag = 3;
    cyn = 5;
    wht = 7;

    {Public Variables}

var
   LST : text;  {Public Variable for Writing to the Printer
                 Useage:  Write(Lst,...) or Writeln(Lst,...)  }

   {Public Routines}

   {Cursor Control Routines}
      procedure gotoxy(x,y : integer);  {Position Cursor}
      procedure crsoff;                 {Turn Cursor Off}
      procedure crson;                  {Turn Cursor On}
      procedure blkcrs;                 {Set Cursor to Block Cursor}
      procedure lnecrs;                {Set Cursor to Line Cursor}

   {Screen Control Routines}
      procedure clrscr;                 {Clear the Screen and Home Cursor}
      procedure clreol;                 {Clear to End of Line}
      procedure clreop;                 {Clear to End of Page}
      procedure set_text_color(for_color, back_color : integer);
      procedure rv;                     {Reverse Video}
      procedure nv;                     {Normal Video}
      procedure on25;                   {Turn 25th line on}
      procedure off25;                  {Turn 25th line off}
      procedure graphmode;              {Turn on H-19 Graphic Characters}
      procedure nographmode;            {Turn off H-19 Graphic Characters}

   {Miscellaneous Routines}
      procedure box(lt, top, rt, bot: Integer); {Draw a Box}
      procedure getkey(var Key : Integer);      {Return Key Code}
      function keypressed : Boolean;           {True if Key has been Hit}
      function Prn_OK : Boolean;               {Checks for Printer Ready}
      procedure alarm;                          {Rings the Bell}
      procedure Go_DOS;                         {Exits to DOS}

implementation
var
   prn_stat : byte;   {Used by Prn_OK function}

{$R-,S-}

   procedure gotoxy(x,y : integer);
   begin
      write(#27'Y',chr(y+31),chr(x+31))
   end;

   procedure crsoff;
   begin
      write(#27'x5')
   end;

   procedure crson;
   begin
      write(#27'y5')
   end;

   procedure blkcrs;
   begin
      write(#27'x4')
   end;

   procedure lnecrs;
   begin
      write(#27'y4')
   end;

   procedure clrscr;  {Fast ClearScreen routine.  Reference Z-100
                       Tech Reference Manual. Even works in 640x400
                       Interlace Mode}
```

control, Borland defines several public record types and constants in the DOS unit. By using the DOS unit, these special definitions are made available to the programmer. Some of these definitions will be used in writing the customized output routines for the Z-100. The defined constants that will be used are

```
          fmOutput = $D7B1
          fmClosed = $D7B3
```

The defined types that will be used are

```
TextBuf = Array[0..127] of Char;

TextRec = Record
          Handle : word;
          Mode : word;
          BufSize : word;
          Private : word;
          BufPos : word;
          BufEnd : word;
          BufPtr : ^TextBuf;
          OpenFunc : pointer;
          InOutFunc : pointer;
          FlushFunc : pointer;
          CloseFunc : pointer;
          UserData : Array[1..16] of Byte;
          Name : Array[1..79] of Char;
          Buffer : TextBuf;
      End;
```

Although there are several other public types and constants defined in the DOS unit that are useful for other specialized tasks, the above definitions are the only ones needed to write the custom device driver.

In the SYSTEM unit there are two public variables that can be used to override the standard input/output drivers. These variables are standard Turbo Pascal Text files and are named INPUT and OUTPUT. The variable OUTPUT will be reassigned to the custom screen device driver.

In Turbo Pascal, hardware devices are treated as files and the software used to control the devices are called Text File Device Drivers. In Turbo Pascal 4.0, a text file device driver is a set of four functions that completely define and control the flow of data between Turbo's file system and the hardware device. The four functions are Open, InOut, Flush, and Close. Open is used to open the device and ready it for communication. InOut is used to transfer data to and/or from the device. Flush is used to handle the data buffer and Close is used to close the device once there is no longer a need to commu-

```
 var
    i, stat : word;

 begin
    stat := port[$0D8];                        {Save Status}
    port[$0D8] := $0F;
    port[$0db] := port[$0db] and $F7;
    port[$0d9] := port[$0D9] and $F7;
    i := 1;
    repeat i := i + 1 until i > 10000;          {Delay}
    port[$0d9] := port[$0d9] or 8;
    port[$0d8] := stat;                        {Reset to Entry Status}
    gotoxy(1,1);                               {Home Cursor}
 end;


 procedure clreol;
 begin
    write(#27'K')
 end;


 procedure clreop;
 begin
    write(#27,'j');       {Save Cursor Position}
    write(#27'J');        {Clear to End of Page}
    on25;gotoxy(1,25);  {Clear the 25th Line}
    clreol;
    write(#27,'k');       {Restore the Cursor Position}
 end;


 procedure set_text_color(fore_color, back_color : integer);
 begin
    write(#27,'m',chr(fore_color+48),chr(back_color+48));
 end;


 procedure rv;
 begin
    write(#27'p')
 end;


 procedure nv;
 begin
    write(#27'q')
 end;


 procedure on25;
 begin
    write(#27'x1')
 end;


 procedure off25;
 begin
    write(#27'y1')
 end;


 procedure GraphMode;                  {Use H-19 Graphics Characters}
 begin
    write(#27'F')
 end;


 procedure NoGraphMode;                {Use Standard Characters}
 begin
    write(#27'G')
 end;


 procedure Box(lt, top, rt, bot: Integer);

{ This routine draws a Box using H-19 Graphics Characters.
 The H-19 Graphics characters are provided in ALTCHAR.SYS
 file that comes with the DOS disks. ALTCHAR.SYS must be
 present on the Boot Disk at Boot-up.}


 var
    i: Integer;
 begin
    GraphMode;
    GotoXY(lt, top); write(chr(102));          {Left-Top Corner}
    for i:=lt+1 to rt-1 do Write(chr(97));      {Top}
```

nicate with the hardware. The OpenFunc, InOutFunc, FlushFunc, and CloseFunc pointers in TextRec are used to direct Turbo to the appropriate functions as the need arises when accessing the hardware. The unit presented in this article will use two custom drivers, both of which will access the Z-100 hardware through BIOS. One driver will provide screen output and the other will provide printer output.

ZCON.PAS
The Z-100 TPU Source Code

Listing 1 is the source code for the Z-100 TPU. Notice that the Unit is named ZCON. This will be important when writing the main programs that will use this TPU. Let's examine the ZCON TPU. The interface section first declares that another TPU named DOS will be used by ZCON. As stated above, the DOS TPU is required for custom I/O routines. Following the "uses DOS" statement is a list of constants that are made public for convenient outside use. These constants are keycodes and colors that I most often use in my programs. If one wanted to enlarge or reduce this list then the list can be easily altered.

Following the public constant definition is a public variable definition, "LST". This variable will be assigned to the other custom device driver and will provide fast printer access through the Z-100 BIOS. Once LST is assigned to the driver, output can be sent to the printer using Pascal write statements in the same way output was directed to the printer in previous releases of Turbo Pascal.

The last set of declarations in the interface section is a list of all the routines in ZCON that will be accessible by other modules. There are 21 public routines; five for cursor control, ten for screen control, and six miscellaneous routines. The comments in Listing 1 (the code in curly brackets) indicate their purpose.

The next major section of the TPU is the implementation section. Most of the routines in this section use the convenient Z-100 escape sequences to perform the operations. These escape sequences are easy to use, relatively fast and well documented in the Z-100 user's manual, technical reference manual and in the Programmer's Utility Pack. I'd like to discuss several routines that do not use escape sequences.

The first one is clear screen, CLRSCR. The clear screen routine goes directly to hard-

```
   Write(chr(99));                                {Right-Top Corner}
    for i:=top+1 to bot-1 do
    begin
       GotoXY(lt , i);  Write(chr(96));           {Left Side}
       GotoXY(rt, i); Write(chr(96));            {Right Side}
    end;
    GotoXY(lt, bot);
    Write(chr(101));                                  {Left-Bottom Corner}
    for i:=lt+1 to rt-1 do Write(chr(97));        {Bottom}
    Write(chr(100)); {Right-Bottom Corner}
    NoGraphMode;
  end  { character box };


  Procedure Getkey(var Key : Integer);

{ This routine checks if a key has been hit and if so, returns
  the keycode else returns a keycode of zero.}

  var
     reg : registers;
  begin
     REG.AX := $0600;
     REG.DX := $00FF;
     MSDos(reg);
     if (reg.flags and 64) = 0 then
        key := reg.al
     else
        key := 0;
  end;


  Function KeyPressed : Boolean;
{ This routine checks to see if a key has been hit. If so,
  Keypressed is True else Keypressed is False. This routine
  does not alter the key buffer. It simply determines if a key
  is available. }

  var
     reg : registers;
  begin
     Reg.AX := $0B00;
     MSDos(reg);
     if reg.Al = $FF then
        keypressed := True
     else
        keypressed := False;
  end;


  procedure ckprn(var pr_st : byte);
    begin
     inline
       ($C4/$BE/PR_ST/          { LES DI,BP[PR_ST] }
       $B8/$00/$02/             { MOV AX,0200H - Check Printer Status}
       $57/                     { PUSH DI }
       $9A/$4B/$00/$40/$00/     { CALL 0040:004B - BIOS PRN_FUNC }
       $5f/                     { POP DI }
       $88/$25);                { MOV [DI],AH }
  end;


  function Prn_OK : Boolean;  { This function determines if the
                                printer is on-line and ready.
                                If so, Prn_OK is True else
                                       Prn_OK is false.}


  begin
     ckprn(prn_stat);
     ckprn(prn_stat);                {Do it Twice for Best Results}
     prn_stat := prn_stat and $80;
     if (prn_stat = 0) then
        Prn_OK := False
     else
        Prn_OK := True;
  end;


  procedure alarm;  {One Ringy-Dingy}
  begin
     write(bell);
  end;
```

ware to accomplish this commonly needed task. The algorithm is provided in the Z-100 Technical Reference Manual. I prefer this method over the escape sequence method for two reasons. First, it is extremely fast and second, it works when the Z-100 is in the 640x400 interlace mode.

The next routine that should be highlighted is the box drawing routine, BOX. This routine uses the H-19 graphics characters defined in the ALTCHAR.SYS file that comes with the Z-100 MS-DOS operating system. In order for BOX to work properly, ALTCHAR.SYS must be present when the system is booted.

The GETKEY routine is a nifty procedure that I like to use in interactive programs. The routine checks the DOS keyboard input buffer to see if a key is available. If a key is available, then the keycode is removed from the buffer and returned, otherwise, a keycode of zero is returned immediately. In order to get a unique keycode for each key combination on the keyboard, the Z-100 key expansion mode must be disabled. Disabling the key expansion mode is accomplished during initialization by the Init_Con_Out routine.

The Prn_OK function is one of my favorites whenever I write code that will access the printer. By using the BIOS printer function, Prn_OK checks to see if the printer is on-line and ready. If the printer is OK, Prn_OK is set to TRUE, otherwise, it is set to FALSE. Before accessing the printer for any printer output operation, I test for printer ready to prevent a nonrecoverable situation. If the printer is not ready, I alert the user by an audio alarm and an appropriate message.

The Go_DOS procedure provides an organized exit from the program. In this routine, I've chosen to clear the screen and reset the Z-100 to its default boot-up condition. Ideally, one would like to return the Z-100 to the state it was in when the program was envoked; however, there is no way to determine the state of the Z-100. If you have a favorite mode of operation for the Z-100 that is different from its default state, then you will have to modify this routine to place the Z-100 in your desired state condition.

The last routines that I will cover are the ones that really make things happen in the TPU. These are the routines that define the custom output drivers. There are

```
  Procedure Go_Dos;
  begin
     clrscr;
     write(#27,'z');  {Reset to Initial Mode}
     Close(Output);
     Assign(Output,");
     Close(LST);
     halt
  end;


{Custom Output Driver - Sends Output to Z-100 Screen through BIOS}
  procedure outchar(out_c : char);  {Sends Character to Screen
                                     via Z-100 BIOS}


  begin
   inline
     ($8a/$46/$04/                        {MOVe out_c to AL register}
     $FC/                                {CLD}
     $9a/$19/$00/$01/$FE)                {Call 0FE01:0019}
  end;


{$F+}   {Force Far Call}

  function outdone(var F:textrec):integer;  {Do Nothing routine for Closing
                                             Access to Z-100 BIOS}


  begin
     outdone := 0;
  end;



  function outstr(var F:textrec): integer;  {Sends the Characters to Screen
                                             from the Character Buffer}


  var
     i : word;
  begin
     with F do begin
        i := 0;
        while i < BufPos do begin
           outchar(Bufptr^[i]);
           inc(i);
        end;
        BufPos := 0;
     end;
     outstr := 0;
  end;



  function outopen(var F:textrec): integer;  {Tell Turbo Pascal which
                                               functions to use for Screen
                                               Output String Handling}


  begin
     With F do begin
        Mode := fmoutput;
        InOutFunc := @outstr;
        FlushFunc := @outstr;
        CloseFunc := @outdone;
     end;
     outopen := 0;
  end;


{$F-}   {Let Turbo Decide}

  procedure AssignOut(var F : Text);  {Tell Turbo how to set up
                                       Custom Screen Driver}
  begin
     With TextRec(F) do begin
        Mode := fmClosed;
        Bufsize := sizeof(Buffer);
        Bufptr := @Buffer;
        OpenFunc := @outopen;
        Name[0] := #0;
     end;
  end;


  procedure init_con_out; {Initialize the Custom Screen Output Driver}
  begin
```

six routines for each driver; four that are required by Turbo and two support routines that I've added for coding convenience. The two output drivers, one for screen output and one for printer output, are structurally the same, so I'll only discuss the screen output driver. The screen output driver consists of the procedure OutChar, the functions OutDone, OutStr, OutOpen, and the procedures AssignOut and Init_Con_Out. The procedure OutChar is responsible for sending a character to the screen. It accomplishes this task by accessing the Z-100 BIOS. I've chosen to access BIOS not at the standard entry point, but rather as far into the BIOS as possible without getting into trouble.

By doing this, screen output is sped-up considerably but at the risk of losing compatibility with future releases of the Z-100 BIOS. If compatibility is a problem then this routine can be modified to access the BIOS at the standard entry point, absolute address 0040:0009 (hex). To do this, change the third line of the inline code to

```
$9a/$09/$00/$40/$00)    (Call 0040:0009)
```

The next four routines are the ones required by Turbo. The OutDone routine is a do-nothing module that is coded simply to satisfy Turbo. This routine is called when the Turbo intrinsic function CLOSE() is envoked. The OutStr routine manages the output buffer. The Turbo file handling procedures that implement Write and Writeln fill the output buffer with the output data. The OutStr routine then takes the data that is in the buffer and sends it one caharacter at a time to the OutChar procedure. Once the buffer is empty, the mission is accomplished. The OutOpen function is used to tell Turbo which routines are used for the InOut, Flush, and Close operations. The AssignOut routine defines the parameters for installing the custom driver and tells Turbo that OutOpen is to be used to open access to the screen output driver. Init_Con_Out initializes the output driver by assigning the standard Turbo Output file to the AssignOut function and then opening the output driver for communication by envoking the intrinsic routine, Rewrite(Output). The last task of Init_Con_Out is to disable the Z-100 key expansion so that the keyboard processor will generate a unique keycode for each key.

The initialization section of the TPU performs two basic tasks. It initializes the cus-

```
     assignout(Output);
     rewrite(Output);
     write(#27,'y?');      {Disable Key Expansion}
  end;

{Custom Printer Driver through Z-100 BIOS}


  procedure lstchar(outch : char);  {Sends Character to Printer
                                     via Z-100 BIOS}
  begin
      inline
         ($8a/$46/$04/                  {MOVe Outch to AL register}
          $fc/                          {CLD}
          $9a/$0c/$00/$40/$00)          {Call 0040:000C}
end;


{$F+}  {Force Far Calls}
  function lstdone(var F:textrec):integer;  {Do Nothing routine for Closing
                                             Access to Z-100 BIOS}
  begin
  lstdone := 0;
  end;


  function lststr(var F:textrec): integer; {Sends the Characters to
                                            Printer from the Char Buffer}
  var
     i : word;


  begin
     with F do begin
        i := 0;
        while i < BufPos do begin
           lstchar(Bufptr^[i]);
           inc(i);
        end;
        BufPos := 0;
     end;
     lststr := 0;
  end;



  function lstopen(var F:textrec):  integer; {Tell Turbo Pascal which
                                              functions to use for Prn
                                              Output String Handling}
  begin
     With F do begin
        Mode := fmoutput;
        InOutFunc := @lststr;
        FlushFunc := @lststr;
        CloseFunc := @lstdone;
     end;
     lstopen := 0;
  end;


{$F-}   {Let Turbo Decide}

  procedure AssignLst(var F : text);  {Tell Turbo how to set up
                                       Custom Printer Driver}
  begin
     With TextRec(F) do begin
        Mode := fmClosed;
        Bufsize := sizeof(Buffer);
        Bufptr := @Buffer;
        OpenFunc := @lstopen;
        Name[0] := #0;
     end;
  end;


  procedure init_lst_out;  {Initialize the Custom Printer Output Driver}
  begin
     assignLst(Lst);
     rewrite(Lst);
  end;


  begin                    {Initialize Turbo for Z-100 Output Drivers}
     init_con_out;
     init_lst_out;
  end.
```

tom screen driver by calling Init_Con_Out and it initializes the custom printer driver by calling Init_Lst_Out. Now that the TPU is completely defined let's use it.

ZCONDEMO.PAS
A Demonstration Program of the ZCON TPU

Listing 2 is a small program that I've written to demonstate some of the capability of the ZCON TPU. The demonstation program will create a menu that provides context sensitive on-screen help. The menu will provide options to get the current time, the current date and exit the program. The program provides some useful routines that you might want to modify for your own software creations. Enjoy.

Using Turbo Pascal 4.0 on a Z-100

Turbo Pascal 4.0 comes with two compilers; one containing the total integrated development environment and one that is a stand-alone compiler. The integrated development environment version requires a true PC compatible machine. This is not a problem for those of you, like me, that have bitten the bullet and purchased PC compatibility hardware like the UCI EasyPC system. However, for those of you that have withstood the PC onslaught, the stand-alone compiler is your solution. The stand-alone compiler will run on any MS-DOS machine. The only drawback is you will have to use your own text editor to create your source code. The stand-alone compiler, though not as convenient as an integrated development environment, is very nice in that it is extemely fast and it provides detailed diagnostics. It will compile the ZCON TPU from a floppy disk in about 6 seconds. This is considerably faster than any compiler I've used. The speed and flexibility of Turbo Pascal 4.0's compilers give you access to both the Z-100 and PC worlds and the ZCON TPU will let you bridge the gap between them.

References

1. **Turbo Pascal 4.0**, Reference Manual, Borland International, 1987.
2. **Z-100 Technical Manual**, Zenith Data Systems, 1983.
3. **MS-DOS Programmer's Utility Pack**, Zenith Data Systems, 1984.

Listing 2

```
Program ZCON_Demonstration;
        {Program that Demonstrates the ZCON TPU

        Authored by :  TOM L. RIGGS, JR.
                       3830 Cloud Drive
                       Colorado Springs, CO 80920 }
Uses DOS, ZCON;

type
    str18 = string[18];
    str78 = string[78];
    opt_vec = array[1..12] of str18;
    col_ptr = array[1..12] of integer;
    opt_help = array[1..12] of str78;
    opt_cmd = array[1..12] of char;
    opt_rec = record
                 options : opt_vec;
                 col_loc : col_ptr;
                 row_loc : Integer;
                 num_col : Integer;
                 help : opt_help;
                 cmd : opt_cmd;
              end;

var
   Main_Opt : Opt_Rec;
   Col : Integer;
   Command : Char;

   procedure Make_Menu(Gen_Rec : Opt_Rec);
   var i : integer;
   begin
      with Gen_Rec do
         for i := 1 to num_col do begin
             gotoxy(col_loc[i],row_loc);
             write(options[i])
         end;
   end;

   procedure Center(row : Integer; w_str : Str78);
   begin
       gotoxy(2,row);
       write(   :78);
       gotoxy(40 - (length(w_str) div 2), row);
       write(w_str);
   end;

   procedure getcmd(Gen_Rec : Opt_Rec; Var Colindex : Integer);

   Var
   i, key : Integer;
   colval : Integer;

   begin
      crsoff;
      With Gen_Rec do begin
         repeat
            center(2,help[colindex]);
            colval := col_loc[colindex];
            gotoxy(colval,row_loc);
            rv;write(options[colindex]);nv;
            gotoxy(colval,row_loc);
            repeat
               getkey(key);
            until key <> 0;
            if key = hom then key := ord( Q );
            write(options[colindex]);
            case key of

               left : if colindex > 1 then colindex := colindex - 1
                          else colindex := Num_col;

               right : if colindex < Num_col then colindex := colindex + 1
                          else colindex := 1;

               else
                       begin
                          i := 1;
                          repeat
                             if UpCase(Chr(key)) = cmd[i] then begin
                                colindex := i;
                                key := ret
                             end;
                             inc(i);
                          until (Key = Ret) or (i > Num_Col);
                       end;

            end; {Case of Key}

         until (key = ret);
      end;
   end;

   procedure Main_Menu(var Main_Opt : Opt_Rec);
   var
      i : integer;
   begin
      With Main_Opt do begin
         for i := 1 to 12 do begin
             Options[i] :=    ;
             Help[i] :=    ;
             Cmd[i] := chr(0);
         end;
         Num_col := 3;
         Col_Loc[1] := 2;
         Col_Loc[2] := 10;
         Col_Loc[3] := 18;
         Row_Loc    := 4;
         Options[1] :=   <T>ime   ;
         Options[2] :=   <D>ate   ;
         Options[3] :=   <Q>uit   ;
         Cmd[1] :=  T ;
         Cmd[2] :=  D ;
         Cmd[3] :=  Q ;
         Help[1] :=  Get and Display the Current Time ;
         Help[2] :=  Get and Display the Current Date ;
      Help[3] :=  Quit and Return to DOS ;
   end;
end;

Procedure Get_DOS_Time;
Var
   C_sec,
   Sec,
   Min,
   Hour : Word;
   Key : Integer;

begin
   GetTime(Hour,Min,Sec,C_Sec);
   Gotoxy(30,6);
   Write( The Time is  ,Hour:1, : ,Min:1, : ,Sec:1);
   Alarm;
   Center(2, Hit any Key to Continue  );
   Repeat
      GetKey(Key)
   Until Key <> 0;
   Gotoxy(1,6);
   Clreop;
end;

Procedure Get_DOS_Date;
Const
   Days : Array[0..6] of String[3] =
          ( Sun , Mon , Tue , Wed , Thu , Fri , Sat );
Var
   Day_of_Week,
   Date,
   Month,
   Year : Word;
   Key : Integer;

begin
   GetDate(Year,Month,Date,Day_of_Week);
   Gotoxy(30,6);
   Write( Today is  ,Days[Day_of_Week],    ,Month:1, / ,Date:1, / ,Year:4);
   Alarm;
   Center(2, Hit any Key to Continue  );
   Repeat
      GetKey(Key)
   Until Key <> 0;
   Gotoxy(1,6);
   Clreop;
end;
```

```
begin
   crsoff;
   Set_Text_Color(wht,blk);
   Clrscr;
   Box(1,1,80,3);
   gotoxy(37,1);
   rv; write(  HELP  ); nv;
   Main_Menu(Main_Opt);
   Make_Menu(Main_Opt);
   Col := 1;
   repeat
      getcmd(Main_Opt, col);
      Command := Main_Opt.Cmd[Col];
      Case command of

           T  : Get_DOS_Time;
           D  : Get_DOS_Date;

      End;

   until (command =  Q );
   Crson;
   Go_Dos;
end.
```

About the Author

Dr. Tom Riggs holds a PhD in Electrical Engineering from Auburn University. He currently is an associate professor for Astronautical Engineering at the U.S. Air Force Academy where he teaches courses in digital and modern control theory. Dr. Riggs enjoys developing engineering software and is the author of the EZPlot function plotting program that is distributed through the Heath Users' Group.

A Bootable EPROM Disk for the H-100 Part 1

Robert F. Hassard
E3466 Tice Creek Drive, #4
Walnut Creek, CA 94595

When I purchased my H-100 kit, one of the primary considerations was the S-100 bus. In fact, assembling the disk controller board was the highlight of constructing the computer. I had constructed two S-100 (Z-80) computers previously and for me S-100 boards are a passion. It was a disappointment that so few boards have been offered for the H-100, and that all that were available were assembled and tested only. Had any of them been available as kits, I would surely have bought them.

I was lucky to be able to obtain a P-SST bare board with documentation. My computer now signs on orally with the date and time whenever it is first booted. Also, I have constructed Pat Swayne's ZPC support board (not once, but three times). But in the three years plus that I have had my H-100, that's all the S-100 board work I have been able to do.

Finally, I have given up all hope for another kit. So the next best thing seems to be to create a board. I have always been impatient with the time it takes for the H-100 to boot and I have had problems with the amount of disk space that is occupied with overhead. I know that the best answer would be a hard disk, but I really have no need for one. If they were available as kits, I would have one nevertheless.

So that is what this article is all about: how to construct a port addressable EPROM board that simulates a hard disk and not only is bootable, but holds all of the system overhead. Because this is a complicated task, this will have to be a three part series. This part, Part I, will cover the construction of the board. Part II will cover the software required, including the revision of IO.SYS, and Part III will cover the revision of the MTR-100 EPROM so that the board will automatically boot.

There are some essential requirements that should be covered at the outset. The MS-DOS Programmers' Utility Pack is required, as well as a soldering iron and board assembly tools. And, most important, access to an EPROM Burner is a must. If your H-100 BOOT ROM is NOT Version 2.9, you might have a problem with the monitor modification.

Port address 0CEH was selected for the board. That is an address that is used by the H-100 Trainer. But the Trainer does not use S-100 boards. If you want to mock this up on a Trainer, then you must choose a different address.

Figure 1 shows the schematic drawing of the board. As you will see, it is a simple straightforward design. The Board Address (0CEH) is hard wired into U11 (74LS30), which sends a board enable Low to U14 (74LS32) when the proper address is received.

On an I/O write, pin 3 of U14 goes Low and enables the EPROM high address latch U12 (74LS374), allowing it to read. It also resets to zero the chained counters U8, U9, & U13 (74LS161), which provide the low EPROM address. An I/O write transmits the desired Sector Number. The high three bits go to U10 (74LS138) and select the proper EPROM. The low five bits go to the high five bits of the EPROM address.

On an I/O read, pin 8 of U14 enables the EPROM outputs. Pin 6 of U16 provides the same signal to do three things. (1) It enables U10 to select and activate an EPROM. (2) It enables U19 (74LS244) to transmit a byte from the EPROM to the data bus. (3) At the end of the signal (low to high) it causes the address counters to increment by one to the address of the next byte.

The 74LS161 used for the three address counters was designed to overflow to the next counter at the start of a count. However, this board must count after each read cycle in order to function properly. Therefore, the overflow from pin 15 must be inverted through gates in U16 (74LS 366).

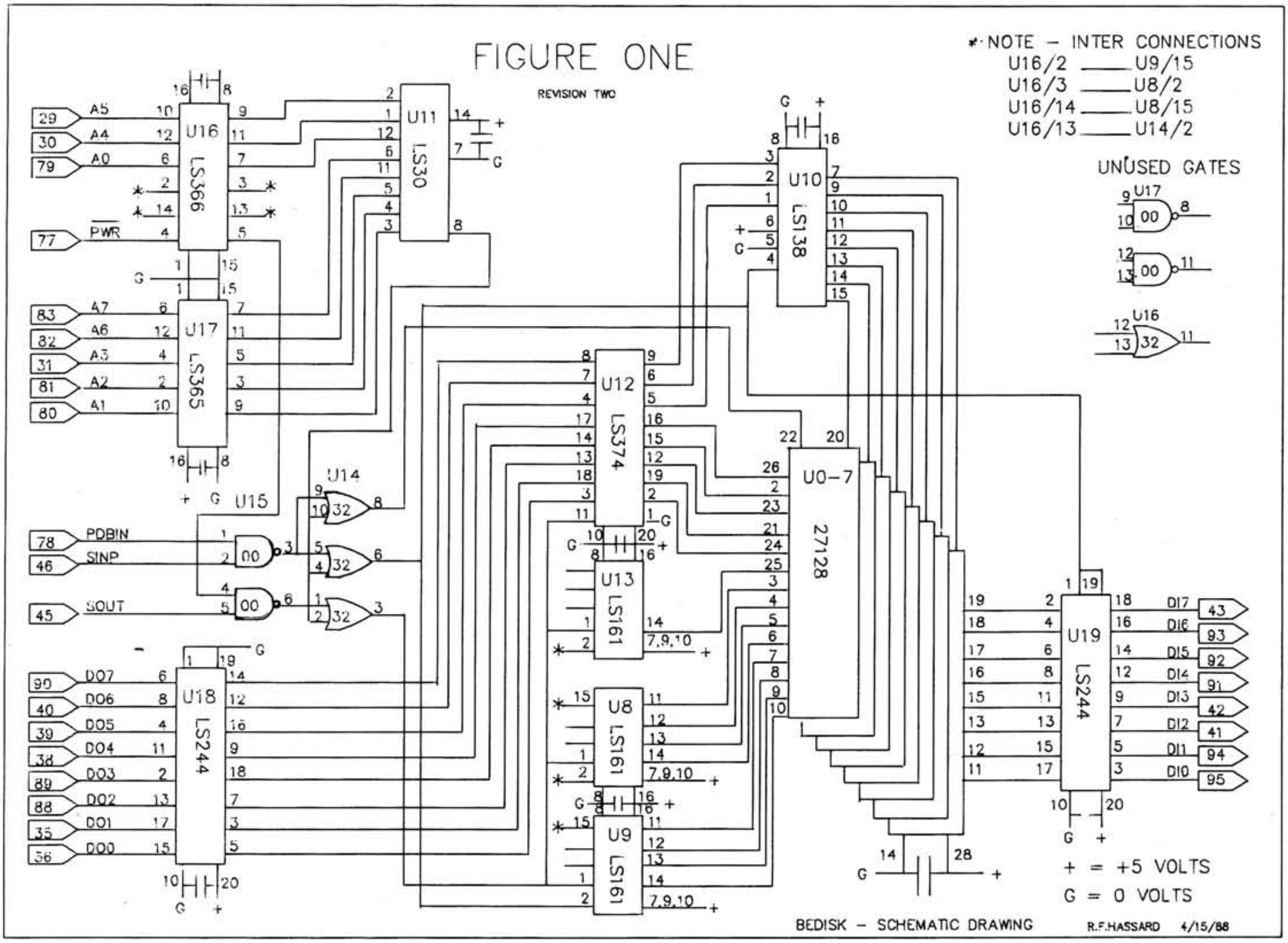


FIGURE ONE

REVISION TWO

*·NOTE – INTER CONNECTIONS
U16/2 ——— U9/15
U16/3 ——— U8/2
U16/14 ——— U8/15
U16/13 ——— U14/2

UNUSED GATES

+ = +5 VOLTS
G = 0 VOLTS

BEDISK – SCHEMATIC DRAWING R.F.HASSARD 4/15/88

The complete power supply circuitry is NOT shown on the schematic because it is standard. Although one +5 volt regulator (7805) should suffice, I used two. One to serve the eight EPROMs and one for the other twelve ICs. Four 10 uf tantulums are required, one at each regulator input and one at each output. In addition there should be an 0.1 uf bypass capacitor at each IC.

Using the board is simple. It is organized into 256 sectors of 512 bytes each for a total of 128K. It has but one address. A write to that address selects the sector to be read. 512 successive reads will read the addressed sector.

In order to reduce power consumption as much as possible, I used the new Advance type chips (e.g., 74ALS00) to the fullest extent available. However, before installing chips it is important to be certain that there are no shorts in the power supply circuits and then to power up and be certain that the chips will receive just 5 volts, no more, no less.

Figure 2 shows the board layout. It is designed to reduce wire lengths to a minimum. The most logical construction technique would be wire wrap. I don't like having to use two slots for one board, so I used a Vector 8801 plugboard, regular low profile sockets and ran wire wrap type wire from pin to pin. I used wire wrap wire because it is thin (30 gauge). I ran the wire on the component side of the board, routing the ends down into the socket holes before inserting the socket. This is a tedious procedure and requires care, because the insulation of wire wrap wire doesn't hold up under heat. But I have a neat thin board that works just fine.

Now about the software. Figure 3 is a program for testing the Board. It is a simple program that writes a test pattern to be recorded on one 128K EPROM. DEBUG is used for this purpose. Call up DEBUG and then enter the program using the A command. Note that I used segment 8000H because it is normally vacant on my computer. You might have to choose a different segment. After the program is entered, use the G command to go do it, but be sure to include a breakpoint as shown so that you will stay in DEBUG. Now use the E command to change the first byte from 00 to EB. (You will find out why when you read Part Two of this series). Next, use the N command to give the test pattern a name. Revise CX to 4000, the size of the pattern. BX must be 0000. Now use the W command to start writing from 8000:0 and you will have the test pattern on a disk file.

As a precaution before writing, you can examine the file starting at 8000:0, using the D command. You will see a simple count starting at 00 thru FF repeated twice. Then a count from 01 thru 00 repeated twice. And so on to the 32nd sector which will be a count from 1F thru 1E repeated twice. This file is then transferred to an EPROM using a Prom Burner. This TEST EPROM is inserted in the U0 position. When it is in the U0 position, your computer will boot a soft disk normally. When it is in any other socket you will get a No Response error message.

Figure 3
Test Program for BEDISK Using DEBUG

```
          Call up DEBUG
          -A8000:4000
          8000:4000 PUSH  ES
          8000:4001 MOV   AX,8000
          8000:4004 MOV   ES,AX
          8000:4006 MOV   DI,0
          8000:4009 MOV   AL,0
          8000:400B MOV   CX,20
          8000:400E CLD
          8000:400F PUSH  CX
          8000:4010 MOV   CX,200
          8000:4013 STOSB
          8000:4014 INC   AL
          8000:4016 LOOP  4013
          8000:4018 INC   AL
          8000:401A POP   CX
          8000:401B LOOP  400F
          8000:401D POP   ES
          8000:401E JMP   401E
          8000:4020 (RETURN)
          -G=8000:4000 401E

          (Display of Registers)
          8000:401E EBFE          JMP     401E
          -E8000:0 00-EB (Return)
          -NBEBRDTST.COM
          -RCX
          CX 0000
          :4000
          -R
AX=0000 BX=0000 CX=4000 SP=FFEE BP=0000 SI=0000 DI=4000
DS=10B6 ES=10B6 SS=10B6 CS=8000 IP=401E NV UP EI PL NZ AC PO NC
(above lines have been shifted left to make room for all flags)
          8000:401E EBFE          JMP     401E
          -W8000:0
          Writing 4000 bytes
          -Q

          A>
```

You must then boot manually by pressing B, F1, and RETURN.

Later on (in Part Two) when we set up the board, it will be organized into 256 sectors of 512 bytes each. So the Test Pattern is organized into 512 byte sectors. There will be 32 sectors on each EPROM.

The first test is performed by powering-up while holding the DELETE key down so as to get the HAND prompt. Now, Out CE,00 followed by In CE should deliver a 00. The next In CE will give a 01. The next a 02, and so on. An Out 01 followed by an In will deliver a 01, and so on. My board didn't do this at first. I had several shorts due to the vulnerable insulation of the wire I used. Although I had checked thoroughly (I thought) with an ohmmeter, I had missed them. However, the strange readings that I got helped me deduce which lines were at fault and lo and behold, each had a short.

You can now examine the board using DEBUG, but that is clumsy. So Figure 4 is a simple dump program which will enable you to bring the test pattern onto the screen one half sector at a time. You can then follow the pattern all the way through. Later on, this program will be useful for examining the contents of the EPROMs to detect errors.

Finally, Figure 5 is a parts list. All of the parts are available from JAMECO Electronics. I haven't priced them because this was intended to be a fun project.

Now I can hear the question: what good is it? Well, for me, I just had to build it. I enjoy constructing circuitry and getting it to work. But, there is also a practical use. Like a memory disk, the EPROM disk is much faster than a floppy. Also, it stands alone with your pre-chosen programs always available.

These programs, including the system overhead, do not have to be on your floppies, so each will have more available capacity. Furthermore, it BOOTS much faster. But I am getting ahead of myself. How all this is accomplished will be covered in the next two parts of this article.

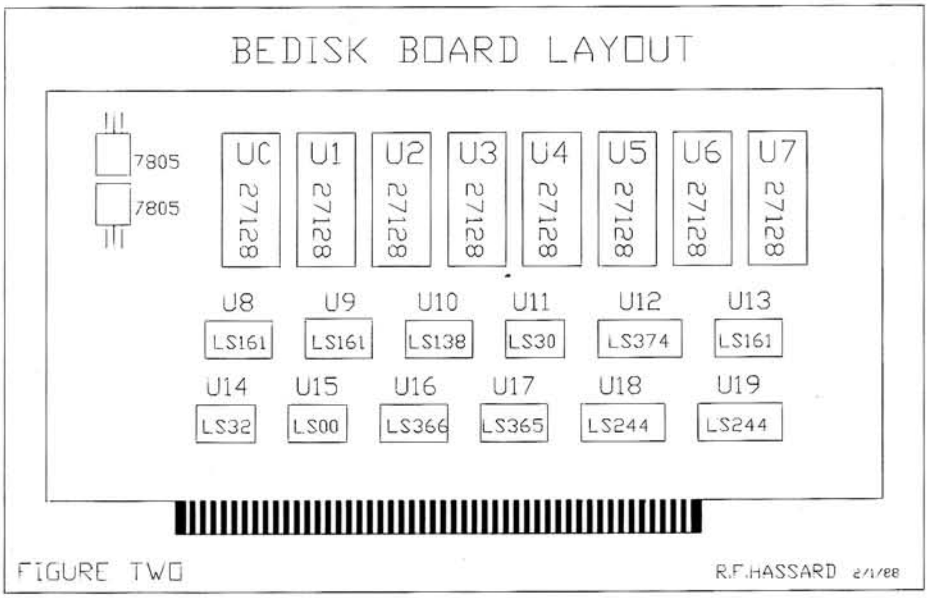


FIGURE TWO

R.F.HASSARD 2/1/88

Figure 4

```
        PAGE    60,92
        TITLE   BEDMP.ASM - BE board DuMP utility
        subttl  R. F. Hassard - January 1988

; This program is intended for use in debugging the
; BEDISK Board following its construction. It may
; be used with a single EPROM created from
; BEBRDTST.COM and installed in U0.
;
; The BEDISK Board is divided into 256 (decimal) Sectors.
; Each Sector contains 512 bytes.
; This utility will display one Sector at a time.
;
; To call, type BEDMP xx - where
; xx is an hexidecimal number without an H and it
; represents the address of the Sector to be displayed.
; Sector addresses run from 00 thru FF. When only
; one EPROM is installed, the top address is 1F.

        .xlist
        INCLUDE DEFASCII.ASM
        INCLUDE DEFMS.ASM
        .list

PORT    EQU     0CEH    ; Must be same as used on Board

PDSEG   SEGMENT
        ASSUME  CS:PDSEG,SS:PDSEG,CS:PDSEG,ES:NOTHING

        ORG     0100H   ; Will be a COM type program

START:  JMP     BEGIN

; Data Area begins here

CRLF    DB      CC_CR,CC_LF,CC_LF,'$'
SP_STR2 DB      '  $'
ASCPRN  DB      '    '
ASCSTR  DB      '                ',CC_CR,CC_LF,'    $'
DSHSTR  DB      ' - $'
PROMPT: DB      CC_CR,CC_LF,CC_LF,'Use [N] for next Sector, '
        DB      'or [P] for previous Sector, ',CC_CR,CC_LF
        DB      '       or Hit RETURN to Abort, or enter '
        DB      'a HEX number (xx) for a new Sector.$'
CON_MSG DB      CC_CR,CC_LF,CC_LF,'   Press any KEY to '
        DB      'continue.',CC_CR,CC_LF,CC_LF,CC_LF,'$'

; Procedures follow from here

; Get address parameter entered by operator
GET_ADR PROC    NEAR
        XOR     DX,DX           ; Default address
        MOV     SI,005DH        ; First FCB parameter
GET_A1: LODSB
        CMP     AL,'0'
        JB      GET_A3          ; Finished
        CMP     AL,'9'
        JBE     GET_A2
        CMP     AL,'G'
        JAE     GET_A3
        SUB     AL,7            ; Convert to HEX
GET_A2: AND     AL,00001111B    ; Convert to Binary
        CBW
        MOV     CL,4
        SHL     DX,CL
        ADD     DX,AX
        JMP     GET_A1

GET_A3: RET
GET_ADR ENDP

; Print address in DX on the screen
PR_ADR  PROC    NEAR
        PUSH    DX              ; Save for other use
        MOV     AL,DL
        AND     AL,11110000B
        MOV     CL,4
        SHR     AL,CL
        CALL    PR_HEX
        MOV     AL,DL
        AND     AL,00001111B
        CALL    PR_HEX
        POP     DX
        RET
PR_ADR  ENDP

; Print a HEX nibble on the screen
PR_HEX  PROC    NEAR
        PUSH    DX
        ADD     AL,'0'
        CMP     AL,'9'
        JBE     PR_H1
        ADD     AL,7            ; Convert to ASCII
PR_H1:  MOV     DL,AL
        SCALL   CONOUT
        POP     DX
        RET
PR_HEX  ENDP

; Print one Byte out of Reg AL
PR_BYT  PROC    NEAR
        PUSH    DX
        PUSH    CX
        PUSH    AX
        AND     AL,11110000B
        MOV     CL,4
        SHR     AL,CL
        CALL    PR_HEX
        POP     AX
        AND     AL,00001111B
        CALL    PR_HEX
        POP     CX
        POP     DX
        RET
PR_BYT  ENDP

; Main Program starts here
BEGIN:  CALL    GET_ADR
BGN1:   PUSH    DX
        MOV     DX,OFFSET CRLF
        SCALL   OUTSTR
        POP     DX
        MOV     CX,32           ; Do 32 lines
        JMP     SHORT BGN12
BGN11:  PUSH    DX
        DEC     CX
        MOV     DX,OFFSET CON_MSG
        SCALL   OUTSTR
        SCALL   DRCINE
        MOV     DX,OFFSET CRLF
        SCALL   OUTSTR
        POP     DX
BGN12:  PUSH    CX
        CALL    PR_ADR          ; Print address once
        PUSH    DX
        MOV     DX,OFFSET SP_STR2
        SCALL   OUTSTR
        POP     DX
        MOV     AL,DL           ; Tell Board Sector #
        OUT     PORT,AL
        POP     CX
BGN2:   PUSH    CX
        MOV     AX,CX
        MOV     CL,4
        SHL     AX,CL
        NEG     AX
        ADD     AX,200H         ; Sector Line address
        PUSH    AX
        MOV     AL,AH
        CALL    PR_BYT
        POP     AX
        CALL    PR_BYT
        PUSH    DX
        MOV     DX,OFFSET DSHSTR
        SCALL   OUTSTR
        POP     DX
        MOV     DI,OFFSET ASCSTR
        MOV     CX,16               ; 16 Bytes per line
BGN4:   IN      AL,PORT
        PUSH    AX
        CMP     AL,'!'              ; Permit only ASCII
        JB      BGN41
        CMP     AL,'$'              ; Discard $
        JE      BGN41
        CMP     AL,'~'
        JBE     BGN42
BFN41:  MOV     AL,'.'              ; All else is a .
BGN42:  STOSB                       ; Save for later
        POP     AX
        PUSH    DX
        CALL    PR_BYT
        MOV     DL,' '
        SCALL   CONOUT
        POP     DX
        LOOP    BGN4
        PUSH    DX
        MOV     DX,OFFSET ASCPRN
        SCALL   OUTSTR              ; Now print ASCII
        POP     DX
        POP     CX
        CMP     CX,17
        JE      BGN11
        LOOP    BGN2
        PUSH    DX
        MOV     DX,OFFSET PROMPT
        SCALL   OUTSTR
        POP     DX
        SCALL   CONIN
        CMP     AL,'N'              ; Next Sector
        JE      BGN43
        CMP     AL,'n'
        JNE     BGN5
BGN43:  INC     DX
        JMP     BGN3

BGN5:   CMP     AL,'P'              ; Previous Sector
        JE      BGN53
        CMP     AL,'p'
        JNE     BGN6
BGN53:  DEC     DX
        JMP     BGN3

BGN6:   MOV     DX,Ø                ; Default address
        MOV     CX,2                ; Get 4 digits
        JMP     SHORT BGN8

BGN7:   SCALL   CONIN
BGN8:   PUSH    CX
        CMP     AL,'Ø'
        JB      FINI                ; Exit if less than Ø
        CMP     AL,'9'
        JBE     BGN9
        CMP     AL,'A'
        JB      FINI
        CMP     AL,'F'
        JA      FINI
        SUB     AL,7                ; Convert to Hex
BGN9:   AND     AL,ØØØØ1111B        ; Convert to Binary
        CBW
        MOV     CL,4
        SHL     DX,CL
        ADD     DX,AX
        POP     CX
        LOOP    BGN7
BGN3:   CMP     DX,Ø1ØØH            ; Ceiling is 256
        JB      BGN31
        MOV     DX,Ø
BGN31:  PUSH    DX
        MOV     DX,OFFSET CRLF
        SCALL   OUTSTR
        MOV     DX,OFFSET CRLF
```

```
        SCALL   OUTSTR
        POP     DX
        JMP     BGN1

FINI:   XOR     AL,AL
        SCALL   EXIT

PDSEG   ENDS
        END     START
```

Figure 5
PARTS LIST Bootable EPROM BOARD

1	Vector 8801 S-100 Plugboard
3	14 pin sockets
6	16 pin sockets
3	20 pin sockets
8	28 pin sockets
2	7805 5 volt 1 Amp regulators
1	74ALS00
1	74ALS30
1	74ALS32
1	74ALS138
3	74ALS161
2	74ALS244
1	74LS365 (or 74LS367)
1	74LS366 (or 74LS368)
1	74ALS374
10	27128-20 128K EPROM 200ns (21V)(1 27128-20) for H-100 Monitor and 1 spare for making changes)
4	10uf Tantulum Capacitors
20	0.1uf Bypass Capacitors

HUG NEW PRODUCTS

10 – Very Good
9 – Good
8 – Average

TABLE C
Product Rating

Rating values 8–10 are based on the ease of use, the programming technique used, and the efficiency of the product.

- **7 – Hardware limitations (memory, disk storage, etc.)**
- **6 – Requires special programming technique**
- **5 – Requires additional or special hardware**
- **4 – Requires a printer**
- **3 – Uses the Special Function Keys (f1,f2,f3,etc.)**
- **2 – Program runs in *Real Time****
- **1 – Single-keystroke input**
- **0 – Uses the H19 (H/Z-89) escape codes (graphics, reverse video)**

Real Time — A program that does not require interactivity with the user. This term usually refers to games that continue to execute with or without the input of the player (e.g., 885-1103 or 885-1211[-37] SEA BATTLE.

ORDERING INFORMATION

For VISA and MasterCard phone; telephone Heath/Zenith Users' Group directly at (616) 982-3838. Have the part number(s), description, and quantity ready for quick processing. VISA and MasterCard require minimum $10.00 order. By mail, send your order, plus 10% postage/handling ($1.00 minimum, $5.00 maximum) to: Heath/Zenith Users' Group, P.O. Box 217, Benton Harbor, MI 49022-0217. Orders may be placed, by mail only, using your Heath Revolving Charge account. Purchase orders are also accepted by phone or mail. No C.O.D.s accepted.

Questions or problems regarding HUG software or REMark magazine should be directed to HUG at (616) 982-3463.

NOTES

When ordering any version of MSDOS software, you must specify what type of media you want the software supplied on. If you want 5-1/4" floppies, add a "-37" to the 7-digit part number. If you want 3-1/2" micro-floppies, add a "-80" to the 7-digit part number.

All special update offers announced in REMark (i.e., ZPC II update) must be paid by check or money order, payable to the Heath Users' Group. **NO CREDIT CARDS ACCEPTED.**

P/N 885-3052
PS's PC and Z-100 Utilities $20.00

Here is another great set of useful programs from HUG. Included on this disk are a utility for testing your disks (hard or floppy), a program to test the operating speed of your computer, a utility that makes fast computers ('286 or '386 machines) run slow (so you can run those old time dependent games), a program that swaps the Caps Lock and Left Ctrl keys on the new 101-key keyboards, and a set of utilities to support batch menu systems.

Requirements: Most of these utilities require MS-DOS version 2 or above and use less than 64k of free memory. DTEST requires 128k of free memory.

Note: Most of the programs on this disk will run on either PC-compatible computers or Z-100 (not PC) computers. A few are for PC-compatible computers only, and are indicated in the individual descriptions below. Of the programs that run on a Z-100, only the ones that do not specify MS-DOS version 2 or above will run under Z-DOS.

Author: Patrick Swayne, HUG Software Engineer. The Z-100 part of IS.COM is by Paul F. Herman, and used with permission.

The PS's PC and Z-100 Utilities disk contains these files:

README	.DOC	CAPCON	.ASM
INSTRUCT	.DOC	KEYS	.COM
DTEST	.COM	KEYS	.ASM
DTEST	.ASM	RUNPROG	.COM
SPEED	.COM	RUNPROG	.ASM
SPEED	.ASM	DT	.COM
KEYCODE	.COM	DT	.ASM
KEYCODE1	.ASM	F	.COM
KEYCODE	.ASM	F	.ASM
IS	.COM	OPTION	.COM
IS	.ASM	OPTION	.ASM
LOOKARG	.COM	LOCATE	.COM
LOOKARG	.ASM	LOCATE	.ASM
RSL	.COM	COLOR	.COM
RSL	.ASM	COLOR	.ASM
CAPCON	.COM	MENU	.BAT

Here is a description of the programs:

INSTRUCT.DOC — Instructions for the programs on the disk.

** Testing utilities.

DTEST.COM — This program performs a non-destructive media test on any disk (floppy, hard, or whatever) supported by MS-DOS. If it locates any bad sectors on the disk that are not used by files, it will give you the option of marking the sectors bad in the File Allocation Table, so that MS-DOS will not attempt to use the bad sectors. DTEST is better than DETECT for frequent use, because you do not have to reformat the disk after testing in order for MS-DOS to recognize the bad sectors. Requires MS-DOS version 2 or above.

SPEED.COM — This program computes the speed of your computer in comparison to the original IBM PC. It times a prime number calculation to compute the speed, and so it will give you a good idea of the performance of your computer at a CPU-intensive task.

KEYCODE.COM — This program shows you what the actual codes produced by your keyboard are at the hardware inter-

Continued on Page 44

HUG Price List

The following HUG Price List contains a list of all products in the HUG Software Catalog and Software Catalog Update #1. For a detailed abstract of these products, refer to the HUG Software Catalog, Software Catalog Update #1, or previous issues of REMark.

PRODUCT NAME	PART NUMBER	OPERATING SYSTEM	DESCRIPTION	PRICE
	H8 — H/Z-89/90			
ACCOUNTING SYSTEM	885-8047-37	CPM	BUSINESS	20.00
ACTION GAMES	885-1220-[37]	CPM	GAME	20.00
ADVENTURE	885-1010	HDOS	GAME	10.00
ASCIRITY	885-1238-[37]	CPM	AMATEUR RADIO	20.00
AUTOFILE (Z80 ONLY)	885-1110	HDOS	DBMS	30.00
BHBASIC SUPPORT PACKAGE	885-1119-[37]	HDOS	UTILITY	20.00
CASTLE	885-8032-[37]	HDOS	ENTERTAINMENT	20.00
CHEAPCALC	885-1131-[37]	HDOS	SPREADSHEET	20.00
CHECKOFF	885-8010	HDOS	CHECKBOOK SOFTWARE	25.00
DEVICE DRIVERS	885-1105	HDOS	UTILITY	20.00
DISK UTILITIES	885-1213-[37]	CPM	UTILITY	20.00
DUNGEONS & DRAGONS	885-1093-[37]	HDOS	GAME	20.00
FLOATING POINT PACKAGE	885-1063	HDOS	UTILITY	18.00
GALACTIC WARRIORS	885-8009-[37]	HDOS	GAME	20.00
GALACTIC WARRIORS	885-8009-[37]	CPM	GAME	20.00
GAMES 1	885-1029-[37]	HDOS	GAMES	18.00
HARD SECTOR SUPPORT PACKAGE	885-1121	HDOS	UTILITY	30.00
HDOS PROGRAMMERS HELPER	885-8017	HDOS	UTILITY	16.00
HOME FINANCE	885-1070	HDOS	BUSINESS	18.00
HUG DISK DUPLICATION UTILITIES	885-1217-[37]	CPM	UTILITY	20.00
HUG SOFTWARE CATALOG	885-4500	VARIOUS	PRODUCTS THRU 1982	9.75
HUGMAN & MOVIE ANIMATION	885-1124	HDOS	ENTERTAINMENT	20.00
INFO. SYSTEM AND TEL. & MAIL SYSTEM	885-1108-[37]	HDOS	DBMS	30.00
LOGBOOK	885-1107-[37]	HDOS	AMATEUR RADIO	30.00
MAGBASE	885-1249-[37]	CPM	MAGAZINE DATABASE	25.00
MAPLE	885-8005	HDOS	COMMUNICATION	35.00
MAPLE	885-8012-[37]	CPM	COMMUNICATION	35.00
MICRONET CONNECTION	885-1122-[37]	HDOS	COMMUNICATION	20.00
MISCELLANEOUS UTILITIES	885-1089-[37]	HDOS	UTILITY	20.00
MORSE CODE TRANSCEIVER	885-8016	HDOS	AMATEUR RADIO	20.00
MORSE CODE TRANSCEIVER	885-8031-[37]	CPM	AMATEUR RADIO	20.00
PAGE EDITOR	885-1079-[37]	HDOS	UTILITY	25.00
PROGRAMS FOR PRINTERS	885-1082	HDOS	UTILITY	20.00
REMARK VOL 1 ISSUES 1-13	885-4001	N/A	1978 TO DECEMBER 1980	20.00
RUNOFF	885-1025	HDOS	TEXT PROCESSOR	35.00
SCICALC	885-8027	HDOS	UTILITY	20.00
SMALL BUSINESS PACKAGE	885-1071-[37]	HDOS	BUSINESS	75.00
SMALL-C COMPILER	885-1134	HDOS	LANGUAGE	30.00
SOFT SECTOR SUPPORT PACKAGE	885-1127-[37]	HDOS	UTILITY	20.00
STUDENT'S STATISTICS PACKAGE	885-8021	HDOS	EDUCATION	20.00
SUBMIT (Z80 ONLY)	885-8006	HDOS	UTILITY	20.00
TERM & HTOC	885-1207-[37]	CPM	COMMUNICATION & UTILITY	20.00
TINY BASIC COMPILER	885-1132-[37]	HDOS	LANGUAGE	25.00
TINY PASCAL	885-1086-[37]	HDOS	LANGUAGE	20.00
UDUMP	885-8004	HDOS	UTILITY	35.00
UTILITIES	885-1212-[37]	CPM	UTILITY	20.00
UTILITIES BY PS	885-1126	HDOS	UTILITY	20.00
VARIETY PACKAGE	885-1135-[37]	HDOS	UTILITY & GAMES	20.00
WATZMAN ROM SOURCE & DOC	885-1221-[37]	CPM	H19 FIRMWARE	30.00
WATZMAN ROM	885-4600	N/A	H19 FIRMWARE	45.00
WHEW UTILITIES	885-1120-[37]	HDOS	UTILITY	20.00
XMET ROBOT X-ASSEMBLER	885-1229-[37]	CPM	UTILITY	20.00
Z80 ASSEMBLER	885-1078-[37]	HDOS	UTILITY	25.00
Z80 DEBUGGING TOOL (ALDT)	885-1116	HDOS	UTILITY	20.00
	H8 — H/Z-89/90 — H/Z-100 (Not PC)			
ADVENTURE	885-1222-[37]	CPM	GAME	10.00
BASIC-E	885-1215-[37]	CPM	LANGUAGE	20.00
CASSINO GAMES	885-1227-[37]	CPM	GAME	20.00
CHEAPCALC	885-1233-[37]	CPM	SPREADSHEET	20.00
CHECKOFF	885-8011-[37]	CPM	CHECKBOOK SOFTWARE	25.00
COPYDOS	885-1235-37	CPM	UTILITY	20.00
DISK DUMP & EDIT UTILITY	885-1225-[37]	CPM	UTILITY	30.00
DUNGEONS & DRAGONS	885-1209-[37]	CPM	GAMES	20.00
FAST ACTION GAMES	885-1228-[37]	CPM	GAME	20.00
FUN DISK I	885-1236-[37]	CPM	GAMES	20.00
FUN DISK II	885-1248-[37]	CPM	GAMES	35.00
GAMES DISK	885-1206-[37]	CPM	GAMES	20.00
GRADE	885-8036-[37]	CPM	GRADE BOOK	20.00
HRUN	885-1223-[37]	CPM	HDOS EMULATOR	40.00
HUG FILE MANAGER & UTILITIES	885-1246-[37]	CPM	UTILITY	20.00
HUG SOFTWARE CATALOG UPDATE #1	885-4501	VARIOUS	PRODUCTS 1983 THRU 1985	9.75
KEYMAP CPM-80	885-1230-[37]	CPM	UTILITY	20.00
MBASIC PAYROLL	885-1218-[37]	CPM	BUSINESS	60.00
MICRONET CONNECTION	885-1224-[37]	CPM	COMMUNICATION	16.00
NAVPROGSEVEN	885-1219-[37]	CPM	FLIGHT UTILITY	20.00
REMARK VOL 3 ISSUES 24-35	885-4003	N/A	1982	20.00
REMARK VOL 4 ISSUES 36-47	885-4004	N/A	1983	20.00
REMARK VOL 5 ISSUES 48-59	885-4005	N/A	1984	25.00
REMARK VOL 6 ISSUES 60-71	885-4006	N/A	1985	25.00
REMARK VOL 7 ISSUES 72-83	885-4007	N/A	1986	25.00

Make the no-hassle connection with your modem today! ***HUGMCP*** doesn't give you long menus to sift through like some modem packages do. With *HUGMCP, YOU'RE* always in control, not the software. Order ***HUG P/N 885-3033-37*** today, and see if it isn't the easiest-to-use modem software available. Joe Katz says it was so easy to use, he didn't even need to look at the manual. "It's the only modem software that I use, and I'm in charge of both HUG bulletin boards!" says Jim Buszkiewicz. *HUGMCP* runs on ANY Heath/Zenith computer that's capable of running MS-DOS!

HEPCAT is here! *HEPCAT* is here! *HEPCAT* is here! So what is *HEPCAT*, you may ask? Why it's just another Pat Swayne SUPER-UTILITY. *HEPCAT* is an acronym for *HUG Engineer's and Programmer's CAlculation Tool*. Just what we don't need, another memory resident calculator, right? Wrong! With *HEPCAT*, you can throw away the rest and use the best. *HEPCAT* only uses two partial lines on your screen, and best of all, does NOT cause existing programs to stop executing! That means, while your computer is grinding numbers internally, you can be grinding them externally. Order ***HUG P/N 885-3045-37***.

Can't remember how to use the MS-DOS 'COPY' command? Forget the exact command line format for 'ASGNPART'. Too far to go for the MS-DOS manuals on the shelf on the other side of the room? Why not just type 'HELP' on the keyboard? You say it comes back with "Bad command or file name"? It wouldn't if you had HUG's ***HELP*** program. With *HELP* installed on your hard disk, all you need to do is type 'HELP' for a complete list of MS-DOS commands and transients along with a brief explanation of how each command works, as well as the format for its use. *HELP*, ***HUG P/N 885-8040-37***, works on *ALL* Heath/Zenith computers that run MS-DOS!

PRODUCT NAME	PART NUMBER	OPERATING SYSTEM	DESCRIPTION	PRICE
SEA BATTLE	885-1211-[37]	CPM	GAME	20.00
UTILITIES BY PS	885-1226-[37]	CPM	UTILITY	20.00
UTILITIES	885-1237-[37]	CPM	UTILITY	20.00
X-REFERENCE UTILITIES FOR MBASIC	885-1231-[37]	CPM	UTILITY	20.00
ZTERM	885-3003-[37]	CPM	COMMUNICATION	20.00

H/Z-100 (Not PC) Only

ACCOUNTING SYSTEM	885-8048-37	MSDOS	BUSINESS	20.00
CALC	885-8043-37	MSDOS	UTILITY	20.00
CARDCAT	885-3021-37	MSDOS	BUSINESS	20.00
CHEAPCALC	885-3006-37	MSDOS	SPREADSHEET	20.00
CHECKBOOK MANAGER	885-3013-37	MSDOS	BUSINESS	20.00
CP/EMULATOR	885-3007-37	MSDOS	CPM EMULATOR	20.00
DBZ	885-8034-37	MSDOS	DBMS	25.00
ETCHDUMP	885-3005-37	MSDOS	UTILITY	20.00
EZPLOT II	885-3049-37	MSODS	PRINTER PLOTTING UTILITY	25.00
GAMES CONTEST PACKAGE	885-3017-37	MSDOS	GAMES	25.00
GAMES PACKAGE II	885-3044-37	MSDOS	GAMES	25.00
GRAPHICS	885-3031-37	MSDOS	ENTERTAINMENT	20.00
HELPSCREEN	885-3039-37	MSDOS	UTILITY	20.00
HUG BACKGROUND PRINT SPOOLER	885-1247-37	CPM	UTILITY	20.00
KEYMAC	885-3046-37	MSDOS	UTILITY	20.00
KEYMAP	885-3010-37	MSDOS	UTILITY	20.00
KEYMAP CPM-85	885-1245-37	CPM	UTILITY	20.00
MAPLE	885-8023-37	CPM	COMMUNICATION	35.00
MATHFLASH	885-8030-37	MSDOS	EDUCATION	20.00
ORBITS	885-8041-37	MSDOS	EDUCATION	25.00
POKER PARTY	885-8042-37	MSDOS	ENTERTAINMENT	20.00
SCICALC	885-8028-37	MSDOS	UTILITY	20.00
SKYVIEWS	885-3015-37	MSDOS	ASTRONOMY UTILITY	20.00
SMALL-C COMPILER	885-3026-37	MSDOS	LANGUAGE	30.00
SPELL5	885-3035-37	MSDOS	SPELLING CHECKER	20.00
SPREADSHEET CONTEST PACKAGE	885-3017-37	MSDOS	VARIOUS SPREADSHEETS	25.00
TREE-ID	885-3036-37	MSDOS	TREE IDENTIFIER	20.00
USEFUL PROGRAMS I	885-3022-37	MSDOS	UTILITIES	30.00
UTILITIES	885-3008-37	MSDOS	UTILITY	20.00
Z100 WORDSTAR CONNECTION	885-3047-37	MSDOS	UTILITY	20.00
ZBASIC DUNGEONS & DRAGONS	885-3009-37	MSDOS	GAME	20.00
ZBASIC GRAPHIC GAMES	885-3004-37	MSDOS	GAMES	20.00
ZBASIC GAMES	885-3011-37	MSDOS	GAMES	20.00
ZPC II	885-3037-37	MSDOS	PC EMULATOR	60.00
ZPC UPGRADE DISK	885-3042-37	MSDOS	UTILITY	20.00

H/Z-100 And PC Compatibles

ADVENTURE	885-3016	MSDOS	GAME	10.00
ASSEMBLY LANGUAGE UTILITIES	885-8046	MSDOS	UTILITY	20.00
BOTH SIDES PRINTER UTILITY	885-3048	MSDOS	UTILITY	20.00
CXREF	885-3051	MSDOS	UTILITY	17.00
DEBUG SUPPORT UTILITIES	885-3038	MSDOS	UTILITY	20.00
DPATH	885-8039	MSDOS	UTILITY	20.00
HADES	885-3040	MSDOS	UTILITY	40.00
HELP	885-8040	MSDOS	CAI	20.00
HEPCAT	885-3045	MSDOS	UTILITY	35.00
HUG BACKGROUND PRINT SPOOLER	885-3029	MSDOS	UTILITY	20.00
HUG EDITOR	885-3012	MSDOS	TEXT PROCESSOR	20.00
HUG MENU SYSTEM	885-3020	MSDOS	UTILITY	20.00
HUG SOFTWARE CATALOG UPDATE #1	885-4501	VARIOUS	PROD 1983 THRU 1985	9.75
HUGMCP	885-3033	MSDOS	COMMUNICATION	40.00
HUGPBBS SOURCE LISTING	885-3028	MSDOS	COMMUNICATION	60.00
HUGPBBS	885-3027	MSDOS	COMMUNICATION	40.00
ICT 8080 TO 8088 TRANSLATOR	885-3024	MSDOS	UTILITY	20.00
MAGBASE	885-3050	VARIOUS	MAGAZINE DATABASE	25.00
MATT	885-8045	MSDOS	MATRIX UTILITY	20.00
MISCELLANEOUS UTILITIES	885-3025	MSDOS	UTILITIES	20.00
REMARK VOL 5 ISSUES 48-59	885-4005	N/A	1984	25.00
REMARK VOL 6 ISSUES 60-71	885-4006	N/A	1985	25.00
REMARK VOL 7 ISSUES 72-83	885-4007	N/A	1986	25.00
REMARK VOL 8 ISSUES 84-95	885-4008	N/A	1987	25.00
SCREEN DUMP	885-3043	MSDOS	UTILITY	30.00
UTILITIES II	885-3014	MSDOS	UTILITY	20.00

PC Compatibles

ACCOUNTING SYSTEM	885-8049	MSDOS	BUSINESS	20.00
CARDCAT	885-6006	MSDOS	CATALOGING SYSTEM	20.00
CHEAPCALC	885-6004	MSDOS	SPREADSHEET	20.00
CP/EMULATOR II & ZEMULATOR	885-6002	MSDOS	CPM & Z100 EMULATORS	20.00
DUNGEONS & DRAGONS	885-6007	MSDOS	GAME	20.00
EZPLOT II	885-6013	MSDOS	PRINTER PLOTTING UTILITY	25.00
GRADE	885-8037	MSDOS	GRADE BOOK	20.00
HAM HELP	885-6010	MSDOS	AMATEUR RADIO	20.00
KEYMAP	885-6001	MSDOS	UTILITY	20.00
LASERWRITER CONNECTION	885-8050	MSDOS	PRINTER UTILITY	40.00
PS's PC UTILITIES	885-6011	MSDOS	UTILITIES	20.00
SCREEN SAVER PLUS	885-6009	MSDOS	UTILITIES	20.00
SKYVIEWS	885-6005	MSDOS	ASTRONOMY UTILITY	20.00
TCSPELL	885-8044	MSDOS	SPELLING CHECKER	20.00
ULTRA RTTY	885-6012	MSDOS	AMATEUR RADIO	20.00

Magazines everywhere, and no way to reference the wealth of information they hold? Not anymore! Now there's ***MAGBASE***; a database designed specifically for referencing magazine articles. Don't let those one-hundred-and-some back issues of REMark, or C Users Journal, or Veternary Medicine, (or any magazine) gather dust, use ***MAGBASE***, and *find* that article you read two years ago! ***MAGBASE*** is available for ***MSDOS HUG P/N 885-3050*** or ***CP/M (P/N 885-1249-[27]***.

Why didn't somebody think of this sooner! Books and magazines are printed on both sides of the page, why not listings from your computer's printer? With HUG's ***"Both Sides"*** printer utility, now you can send your printer's output to both sides of the page; properly positioned, and page numbered! Cut your paper usage, and binder space usage in half! Order HUG's ***"Both Sides"*** printer utility today. ***Both Sides*** is available for MSDOS and is ***HUG P/N 885-3048***.

LAPTOP OWNERS . . . don't feel left out! All of HUG's MSDOS software is available on 3-1/2" micro-floppies too! When ordering, just add a "-80" to the 7-digit HUG part number. For the standard 5-1/4" floppy, just add a "-37".

ORDERING INFORMATION

For VISA and MasterCard phone orders, telephone the Heath Users' Group directly at (616) 982-3838. Have the part number(s), descriptions, and quantity ready for quick processing. By mail, send your order, plus 10% postage and handling ($1.00 minimum charge, up to a maximum of $5.00) to: Heath Users' Group, P.O. Box 217, Benton Harbor, MI 49022-0217. VISA and MasterCard require minimum $10.00 order. **No C.O.D.s accepted.**

Questions regarding your subscription? Call Margaret Bacon at (616)982-3463.

Continued from Page 41

rupt level and at the BIOS level. It is very useful in determining the actual key codes produced by all of the extra keys on the new 101-key keyboards. This program is for PC-compatible computers only.

KEYCODE1.COM — This version of KEYCODE displays only the hardware interrupt key codes. Because it does not access the BIOS, you can examine the codes produced by any key sequence, such as Ctrl-Alt-Del, or Ctrl-Break without causing the computer to reset or the program to exit. This program is for PC-compatible computers only.

IS.COM — This utility performs a quick check to determine if a floppy disk in a specified drive is formatted (readable) or not. If you have ever put an unformatted disk in a drive by accident and then tried to do a DIRectory on the disk, you know how long MS-DOS takes to figure out that the disk can't be read. This program makes the determination much more quickly. Requires MS- DOS version 2 or above.

LOOKARG.COM — With this program, you can see exactly how MS-DOS interprets an argument you type on the command line. It shows you what MS-DOS puts in both File Control Blocks (FCB's), and what it puts in the command line buffer. If you write programs that use command line arguments, now you can see exactly what your programs will "see" when you give them a particular command line argument.

** Programs that modify the operation of your computer

RSL.COM — This program runs any other program you specify on the command line as a "child" at a much slower speed than it would normally run. The main purpose of RSL is to let you run your old time dependent game programs on a new super-fast system. Unlike other slow-down programs that use time slicing to slow the system, RSL uses the trap interrupt. The result is that the slow-down is constant, and is not dependent on the timer-counter chip, which some game programs command to operate in non-standard modes. Since the trap interrupt is disabled during other interrupts, disk access and background interrupt-driven programs (such as the HUG screen clock) run at full speed during the slow-down. The degradation imposed by RSL is greater on 32-bit machines (80386) than on 16-bit machines (80286), and a 16 MHz 80286 or a 25 MHz 80386 will run approximately as fast as the original IBM PC while RSL is active. If your machine is slower, the degradation will, of course, be greater, which can give you an advantage on some of those old games you've had trouble with. Requires MS-DOS version 2 or above.

CAPCON.COM — This program is a memory-resident utility that swaps the operation of the Caps Lock and left Ctrl keys on the new 101-key keyboards. Now you can have your Ctrl key in the "right" place without having to make a hardware modification to your keyboard or install new chips. And you can swap the keys back to their original use any time you need to. If you have already modified your hardware, you can use this program to make the keys work the old way if you need to. This program is for PC-compatible computers only.

KEYS.COM — With this utility, you can set the state of the Caps Lock, Num Lock, and Scroll Lock keys from the MS-DOS command line or a batch file. You can put a line in your AUTOEXEC.BAT file that will set the state of any or all of these keys the way you want them at boot-up. This program is for PC-compatible computers only.

RUNPROG.COM — This utility runs a program as a child with the top 64k of system memory reserved. It was designed primarily for running certain programs under ZPC on a Z-100 that otherwise would not run correctly, but it can be used in any other situation where you need to reserve the top 64k. Requires MS-DOS version 2 or above.

** Directory utilities

DT.COM — This program displays all of the directories on the default drive or a specified drive in "tree" form, with graphic lines connecting the directory names. It is similar to various public domain or "shareware" programs, except that it will run on Z-100's, as well as PC-compatible computers. Requires MS-DOS version 2 or above.

F.COM — This program can search all of the directories on the default or specified drive for a specified file or group of files (if wild card characters are used in the file description). It displays the complete path to the located file(s), along with the date, time, and file size. Requires MS-DOS version 2 or above.

* Batch menu system utilities.

OPTION.COM —This program accepts a single character input within a range specified on the command line, and returns the ASCII value of the input character as its exit code. It is used for selecting items in batch menu systems. Requires MS-DOS version 2 or above.

LOCATE.COM — This program positions the cursor to a spot on the screen that you specify on the command line. It is used to position the cursor at the end of a prompt in batch menu systems.

COLOR.COM — This program clears the screen and sets the foreground and background colors to values you specify on the command line. If you run the program without specifying any values, it will display a list of available colors and their numerical values.

MENU.BAT — This is a sample menu using the above three programs.

Requires MS-DOS version 2 or above. It is recommended that you use MS-DOS version 3 or above for batch menu systems, since batch processing in MS-DOS version 2 is too slow for such menus to be practical.

** Source code

***.ASM** — The assembly source code for each program is included.

Multi-Edit in Review

Thomas Lisanti III
P.O. Box 1432
Melville, NY 11747

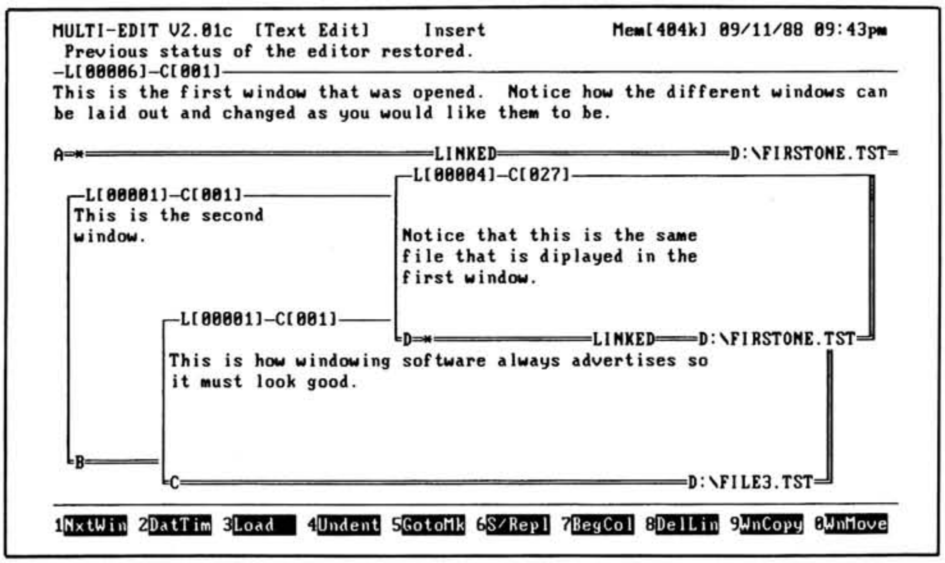
I have recently been searching for the best text editor to use when I program. Since the Z-100 could run CP/M or MS-DOS there was never any problem with getting programming languages. The problem to me always seemed to be the support for these languages. The biggest concern of which was the editor. Borland recognized this way back when they made Turbo Pascal a programming environment, however, Microsoft has been very slow in seeing the light. Only recently with the introduction of the Microsoft "Quick" languages has Microsoft put the user in that warm comfortable feeling of a programming environment. However, since most of my programming has been in either assembler or Pascal, and Microsoft feels QUICK-C and QUICK BASIC are the way to go, Microsoft and I just don't quite see eye to eye. I realize an environment for assembler might be a little too much, but what happened to Pascal? Did Microsoft give up? Was it left in the dust behind Turbo? The environments do give you a lot but they also take away that individuality that some us need. In particular, using the editor that they have incorporated into the system instead of the one we (the users) like best or are most comfortable with using. Yes, Turbo does allow some key assignment changes to make you feel more at home, but trying to get their editor to exactly match BSE is a challenge in itself. For those of you who are less fortunate and have never had the pleasure of being introduced to BSE (Basic Symbolic Editor) I will give a quick overview. BSE comes with the programmers utility package from Heath/Zenith. It is one of the best, yet least known text editor on the market. I have been using it ever since I received the package and every other editor I have ever used or seen has been compared to it. It has everything a programmer needs in a small concise package. The documentation is short and succinct, the editor boasts multiple file edits, virtual memory management, direct full screen editing (move the cursor to where you want to edit, unlike EDLIN), simple keystroke commands and a host of other features I do not have the room to list. Some programmers I know write small programs, and therefore, stick to using their word processors for writing programs. I have converted a few of these people, because until they try an editor specifically designed for programs, they don't know what they are missing. There are major differences between text editors and word processors, each has its own field of expertise.

The one thing that started me on the search for the perfect programmers text editor was my new color EGA monitor. Colors are not needed for the programmer to do his job, but they sure do look nice. Along the way in my search for the perfect programmer's editor I came across some shareware editors that showed some very nice features. One had very nice color selection and windows that could be resized on the screen. This allows two (or more) files to be compared and copied between very easily. However, there was no virtual memory. Another editor added an interesting feature that allowed a compile while using the editor, thus creating a pseudo environment for any programming language, alas no windows, which had now become a must have after seeing the last editor. And so my search continued looking for a "BSE" that had colors and windows and could be used to create some sort of programming environment for myself.

One day I got lucky. A friend who ties

onto bulletin boards around the country found a demo of Multi-Edit. I was at this point very skeptical, but hopeful. Amazingly enough Multi-Edit does everything I have mentioned, along with a host of other features I never thought an editor could do. The program Multi-Edit is by American Cybernetics. After playing with the demo for about half an hour I ordered it without delay. Like BSE, I have not stopped using it since I opened the box. I am so pleased with the functionality, ease of use and all around improvements in my programming efficiency that I feel every programmer should know about Multi-Edit. Although I found out about Multi-Edit through a bulletin board, it is not shareware. At a cost of less than $100 it is the best investment I have made in some time. It is fully supported by a very competent team of programmers that are both friendly and helpful. The people at American Cybernetics are interested in user feedback. They offer a free upgrade for those who bother to mail back the registration and comment form and write down any important feedback after using Multi-Edit for some period of time. Some of the features of Multi-Edit are:

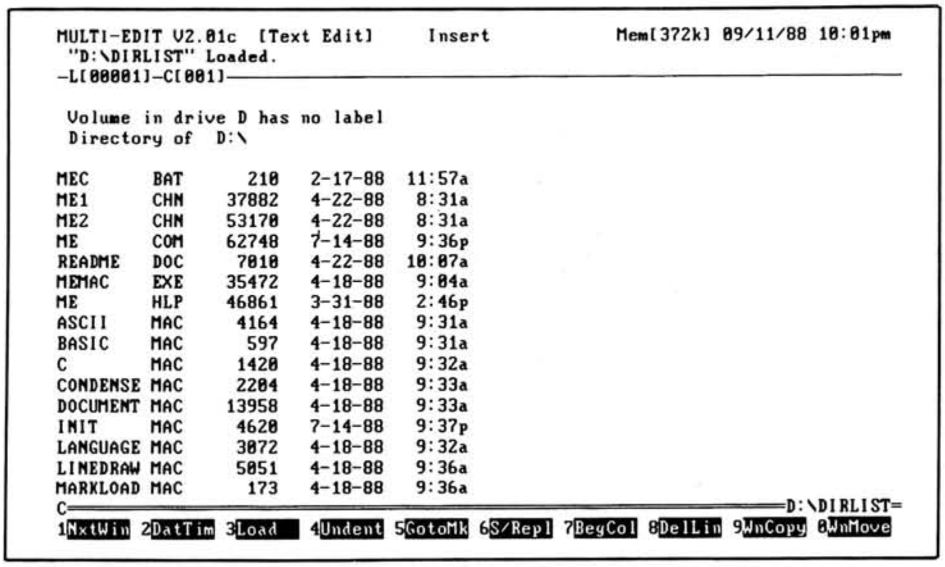
- Uses a C/Pascal type Macro programming language for programming the editor itself!
- Keyboard macros for fast repeatable operations
- More than 25 lines of display when using an EGA or VGA
- Works with DesqView and TopView type programs



```
MULTI-EDIT V2.01c  [Text Edit]     Insert             Mem[404k] 09/11/88 09:43pm
 Previous status of the editor restored.
-L[00006]-C[001]─────────────────────────────────────────────────────────────
This is the first window that was opened.  Notice how the different windows can
be laid out and changed as you would like them to be.

A=*════════════════════════════════LINKED══════════════════D:\FIRSTONE.TST=
                                  ┌L[00004]-C[027]──────────────────────────┐
 ┌L[00001]-C[001]────────────────│                                          │
 │This is the second             │                                          │
 │window.                        │Notice that this is the same              │
 │                               │file that is diplayed in the              │
 │                               │first window.                             │
 │                               │                                          │
 │       ┌L[00001]-C[001]────────│                                          │
 │       │                       └D=*═══════════LINKED═══D:\FIRSTONE.TST═══┘
 │       │This is how windowing software always advertises so              │
 │       │it must look good.                                               │
 │       │                                                                 │
 │       │                                                                 │
 └B══════│                                                                 │
         └C═══════════════════════════════════════════════D:\FILE3.TST═════┘
─────────────────────────────────────────────────────────────────────────────
1NxtWin 2DatTim 3Load   4Undent 5GotoMk 6S/Repl 7BegCol 8DelLin 9WnCopy 0WnMove
```

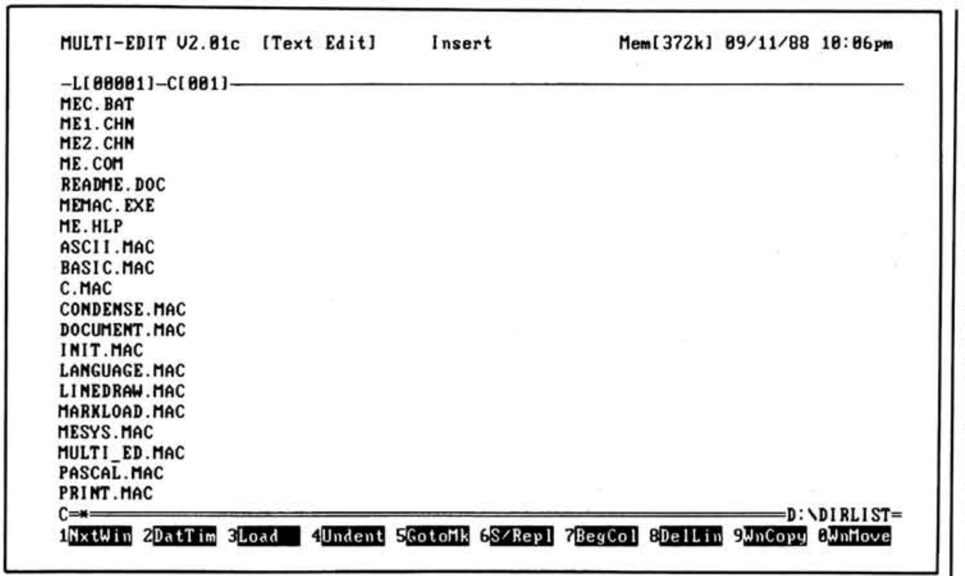
- Up to 100 windows open at any time with a view of the same or different file
- Uses pull down menus or function keys for all functions
- On-line HELP for all operations
- Full reverse editing (Undo) of last 100 operations
- Allows complete redefinition of the keyboard for all functions
- Allows modification of colors by user
- Edits files containing over 2 million lines of code
- Supports a mouse
- Contains printer definition files for enhanced printed files
- DOS directory and Shell functions for file handling functions while using the editor
- Template editing (One keystroke sets up IF/THEN, DO/WHILE, BEGIN/END constructs)

Most of the features listed above show that the one strong feature of Multi-Edit is that it is fully re-definable. The function key assignments can not only be changed to a different function, but with some the function itself can be altered. Colors, keyboard layout and printers, for which everyone has their own belief as to which are better to use than others, can define their own set to be used for Multi-Edit. The editors full screen display, which I usually put in EGA mode for 43 lines of text, really allows a lot of room for windows. Scrolling within each window is allowed in any direction, and is very fast. A mnemonic for each function key 1 through 10 is displayed on the bottom line of the screen for fast reference. When you hold down the ALT, CTRL or SHIFT keys on the keyboard, Multi-Edit changes the mnemonics on the screen correspondingly so that those otherwise hidden key functions are now easily visible. If you want to use that line for displaying more text on the screen, you can turn off the function and use that line for file display space. If you need more detailed help than the quick key reference, then the on-line help for Multi-Edit is ready. The help for Multi-Edit is context sensitive. This means that if you are in the middle of doing a search function and you hit the help key, you get a help screen for the search function, not some general index to browse through.



```
MULTI-EDIT V2.01c  [Text Edit]     Insert            Mem[372k] 09/11/88 10:01pm
 "D:\DIRLIST" Loaded.
-L[00001]-C[001]─────────────────────────────────────────────────────────────

 Volume in drive D has no label
 Directory of  D:\

MEC       BAT     210   2-17-88  11:57a
ME1       CHN   37882   4-22-88   8:31a
ME2       CHN   53170   4-22-88   8:31a
ME        COM   62740   7-14-88   9:36p
README    DOC    7010   4-22-88  10:07a
MEMAC     EXE   35472   4-18-88   9:04a
ME        HLP   46861   3-31-88   2:46p
ASCII     MAC    4164   4-18-88   9:31a
BASIC     MAC     597   4-18-88   9:31a
C         MAC    1420   4-18-88   9:32a
CONDENSE  MAC    2204   4-18-88   9:33a
DOCUMENT  MAC   13958   4-18-88   9:33a
INIT      MAC    4620   7-14-88   9:37p
LANGUAGE  MAC    3072   4-18-88   9:32a
LINEDRAW  MAC    5051   4-18-88   9:36a
MARKLOAD  MAC     173   4-18-88   9:36a
C═══════════════════════════════════════════════════════════════D:\DIRLIST=
1NxtWin 2DatTim 3Load   4Undent 5GotoMk 6S/Repl 7BegCol 8DelLin 9WnCopy 0WnMove
```

I feel the most powerful aspect of Multi-Edit is its macro language. Multi-Edit has keyboard macros, but I am talking about the macro language used by Multi-Edit. The macro language looks almost like Pascal or Modula-2 with a little bit of C mixed in for good measure (a C programmer might see the reverse). The macro language allows programs to be written that modify the text and environment of Multi-Edit. Multi-Edit comes with a compiler for the macros so that they can be run efficiently by Multi-Edit. The macro language has string, integer, character and floating point type variables, along with definable local and global variables. As an example, I have written a small example program



```
MULTI-EDIT V2.01c  [Text Edit]      Insert           Mem[372k] 09/11/88 10:06pm

-L[00001]-C[001]────────────────────────────────────────────────────────────────
MEC.BAT
ME1.CHN
ME2.CHN
ME.COM
README.DOC
MEMAC.EXE
ME.HLP
ASCII.MAC
BASIC.MAC
C.MAC
CONDENSE.MAC
DOCUMENT.MAC
INIT.MAC
LANGUAGE.MAC
LINEDRAW.MAC
MARKLOAD.MAC
MESYS.MAC
MULTI_ED.MAC
PASCAL.MAC
PRINT.MAC
C=*═════════════════════════════════════════════════════════════════D:\DIRLIST=
```

1NxtWin 2DatTim 3Load 4Undent 5GotoMk 6S/Repl 7BegCol 8DelLin 9WnCopy 0WnMove

that cleans up a directory listing produced by DOS into just the file names. It is shown in Figure 1. Notice that the macro has defined variables, logical control loops and simple program flow. The defined functions are shown as all caps.

You should be able to see that the macro language is a very powerful tool for any application. Multi-Edit has many predefined functions for complex string operations, windows (pull down or bar type!), block, and search operations. Multi-Edit comes with a host of macros which you receive the source to when you purchase. Some of the macros are a pop-up calculator, ascii table, line drawing (for boxing in headings, etc.), condensed display, "smart" indenting, and my favorite, the "language macro".

The "smart" indenting allows the programmer to obtain a nicely laid out indented high-level language program without worry. If the editor sees a specific keyword (for that language) as the first word in a line, it will indent the next line slightly further to offset the new function.

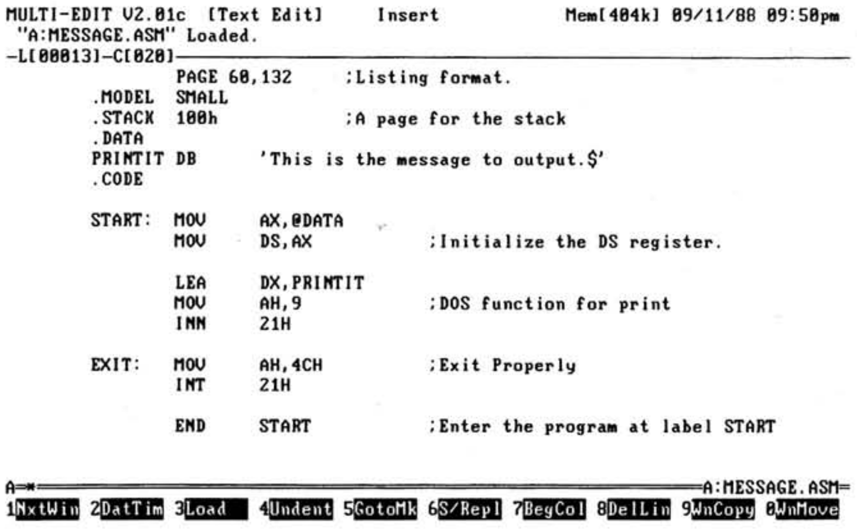
The macro for the condensed display function allows only lines that start on or at a designated column. This in effect displays only the first line and end line of higher level functions, thus easing the movement around the program from different procedures and functions. Any of these functions can be used or turned off, it is totally the decision of the user. It is obvious from the last few functions that Multi-Edit is really tuned for high performance in a programming environment. An interesting note to Multi-Edit's macro language is that the whole initialization procedure for Multi-Edit is written in the macro language. After the user sets the system the way he likes, Multi-Edit compiles INIT.SRC, the source for the initialization procedure using LANGUAGE.MAC (the programming environment macro). It all works together very elegantly.

The language macro (LANGUAGE.MAC) is what allows Multi-Edit to compile your source code and, while still in the editor, show you your errors for fast and easy correction. To show you an example of what Multi-Edit looks like when using it to develop an assembly language program, I wrote a small program that simply outputs a string to the console. I then purposely misspelled the INT instruction so that the compiler would error. Hitting CTRL-F8 (execute LANGUAGE macro) on my keyboard let Multi-Edit do its thing, and voila Figure 2. I can now quickly correct the syntax error and continue. I say quickly because Multi-Edit does not only display the error, but has put my cursor on the line that contains the error. If I were using a compiler that also gives a column number for the error (such as Microsoft Pascal 4.0), the cursor would also be at the column, as well as on the line that is flagged as being an error.

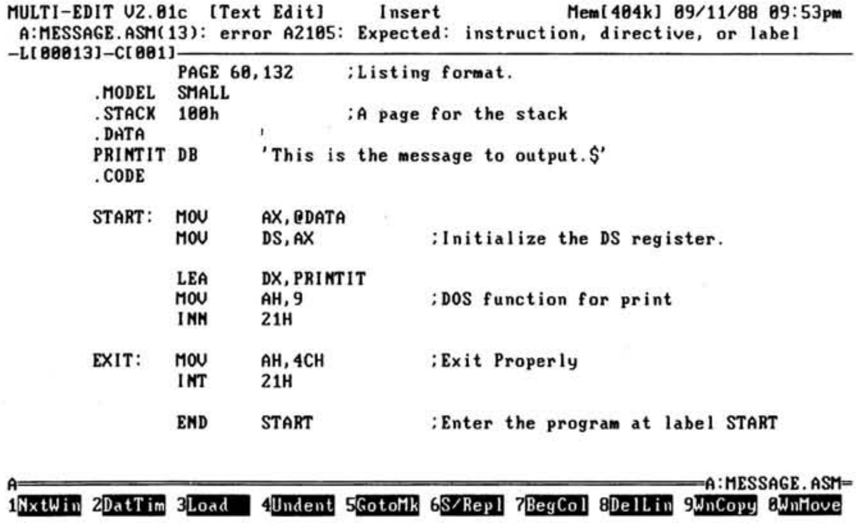
Depressing CTRL-F2 (find next error) steps me to the next error (if there was one). In my example, there are no more errors and Multi-Edit displays 'No Errors' in the status line. Depressing CTRL-F8 again compiles the corrected program without any hassles. How nice it feels to program assembler in my own cozy programming environment. Multi-Edit also works very well with mixed language program systems because each window is handled separately depending on the ex-

Figure 1

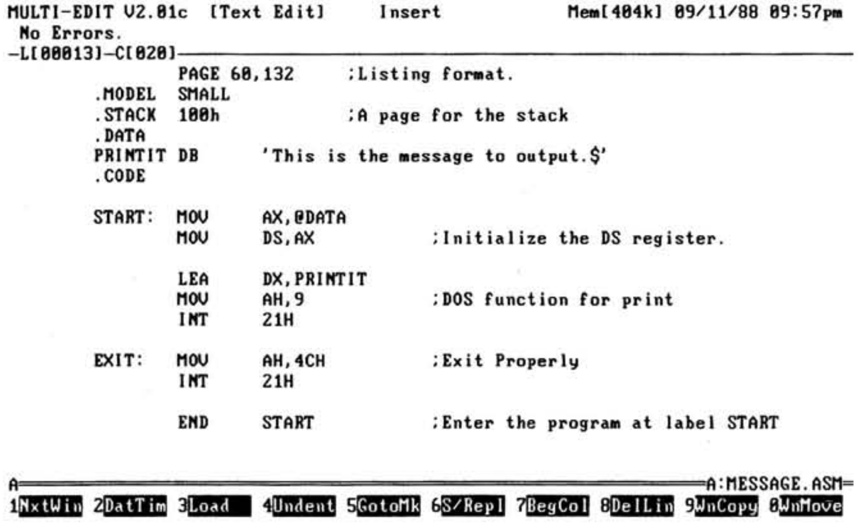
```
$MACRO DIRCLEAN;  {Name the MACRO}

  DEF_STR(The_Work_Line);    {Only one local variable to declare}
  REFRESH := False;         {Do not bother updating the display when run}
  MESSAGES := False;         {Do not tell us what you re doing either}

  TOF;
  WORKING;    {Display flashing message WORKING on status line for user}
  WHILE NOT (XPOS('-',Get_Line,Ø)) DO    {Delete all the lines up to the first}
    DEL_LINE;                             {line a file name is on.           }
  END;  {while}

  WHILE NOT (AT_EOF) DO

   The_Work_Line := GET_LINE;                    {Assign line to a work variable}
      IF (COPY(The_Work_Line,1,1) <> '.') THEN {If it is not . or .. then
                                                                       process}
     IF (COPY(The_Work_Line,1Ø,1) <> ' ' ) THEN           { If there is a file    }
       The_Work_Line := STR_INS('.', The_Work_Line,1Ø); {   extension put a dot }
     END: {If then}                                        {   in front of it.     }

     {Get the filename and extension along with removing the spaces in the name}
     The_Work_Line := REMOVE_SPACE(COPY(The_Work_Line,1,13));
     The_Work_Line := STR_DEL(The_Work_Line,XPOS(' ',The_Work_Line,1),1);

           {Replace the current line by the parsed filename}
           PUT_LINE(The_Work_Line);
     DOWN;
      ELSE
           DEL_LINE;          {If it is a DIR then delete the line}
      END;    {If Then}
  END;    {While}
```



```
MULTI-EDIT V2.01c  [Text Edit]     Insert             Mem[404k] 09/11/88 09:50pm
 "A:MESSAGE.ASM" Loaded.
-L[00013]-C[020]-----------------------------------------------------------------
               PAGE 60,132       ;Listing format.
        .MODEL  SMALL
        .STACK  100h             ;A page for the stack
        .DATA
        PRINTIT DB      'This is the message to output.$'
        .CODE

        START:  MOV     AX,@DATA
                MOV     DS,AX             ;Initialize the DS register.

                LEA     DX,PRINTIT
                MOV     AH,9              ;DOS function for print
                INN     21H

        EXIT:   MOV     AH,4CH            ;Exit Properly
                INT     21H

                END     START             ;Enter the program at label START

A=*=========================================================A:MESSAGE.ASM=
1NxtWin 2DatTim 3Load   4Undent 5GotoMk 6S/Repl 7BegCol 8DelLin 9WnCopy 0WnMove
```



```
MULTI-EDIT V2.01c  [Text Edit]     Insert             Mem[404k] 09/11/88 09:53pm
 A:MESSAGE.ASM(13): error A2105: Expected: instruction, directive, or label
-L[00013]-C[001]-----------------------------------------------------------------
               PAGE 60,132       ;Listing format.
        .MODEL  SMALL
        .STACK  100h             ;A page for the stack
        .DATA
        PRINTIT DB      'This is the message to output.$'
        .CODE

        START:  MOV     AX,@DATA
                MOV     DS,AX             ;Initialize the DS register.

                LEA     DX,PRINTIT
                MOV     AH,9              ;DOS function for print
                INN     21H

        EXIT:   MOV     AH,4CH            ;Exit Properly
                INT     21H

                END     START             ;Enter the program at label START

A===========================================================A:MESSAGE.ASM=
1NxtWin 2DatTim 3Load   4Undent 5GotoMk 6S/Repl 7BegCol 8DelLin 9WnCopy 0WnMove
```

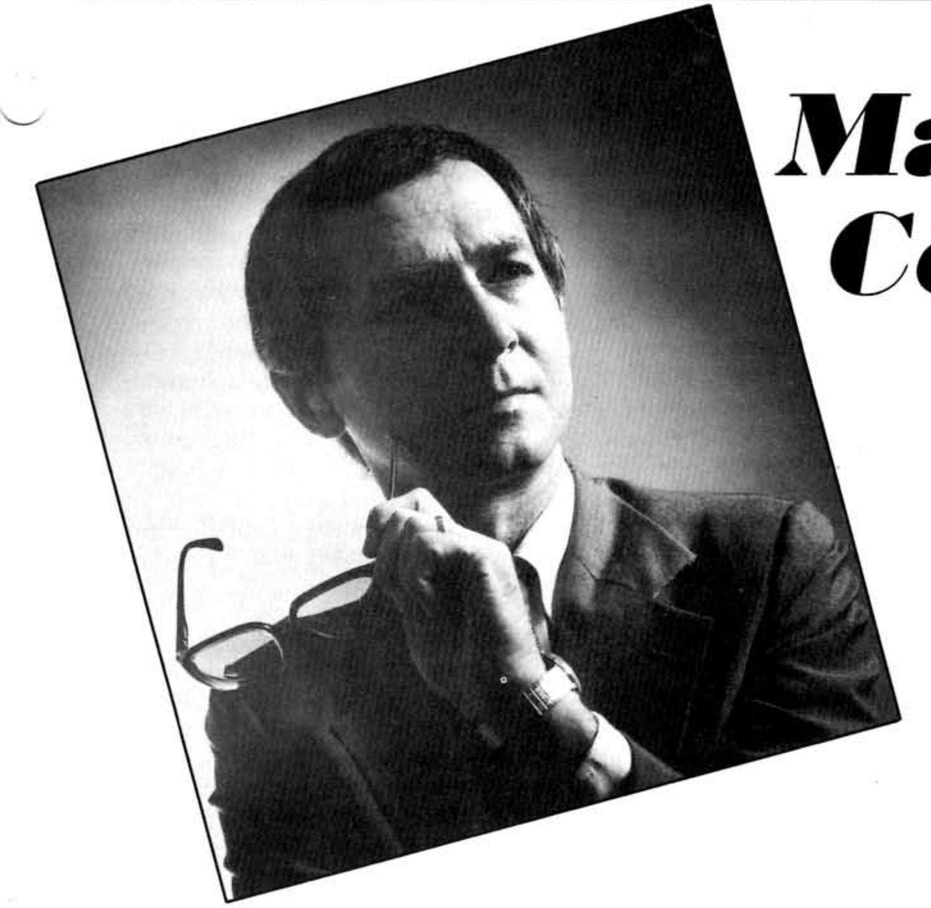


```
MULTI-EDIT V2.01c  [Text Edit]     Insert             Mem[404k] 09/11/88 09:57pm
 No Errors.
-L[00013]-C[020]-----------------------------------------------------------------
               PAGE 60,132       ;Listing format.
        .MODEL  SMALL
        .STACK  100h             ;A page for the stack
        .DATA
        PRINTIT DB      'This is the message to output.$'
        .CODE

        START:  MOV     AX,@DATA
                MOV     DS,AX             ;Initialize the DS register.

                LEA     DX,PRINTIT
                MOV     AH,9              ;DOS function for print
                INT     21H

        EXIT:   MOV     AH,4CH            ;Exit Properly
                INT     21H

                END     START             ;Enter the program at label START

A===========================================================A:MESSAGE.ASM=
1NxtWin 2DatTim 3Load   4Undent 5GotoMk 6S/Repl 7BegCol 8DelLin 9WnCopy 0WnMove
```

tension of the file. This means that if you had one window open for the file HIGHLVL.PAS and another for LOWLVL.ASM, using the 'compile' command with the .PAS file would execute your chosen Pascal compiler, not the assembler.

If your .PAS called the .ASM as a procedure, Multi-Edit allows you to save all files that have changed when the 'compile' command is executed. This insures that the .ASM and .PAS would be up to date so that the "MAKE" command (for your particular compiler) would compile the latest of any of your changes. All of this means that even the most complicated of program systems are easily manipulated and handled by Multi-Edit. The time saved by staying in an editing environment, while developing, are enormous, especially when first generating a piece of software that can have typos and syntax errors. Since the LANGUAGE macro source comes with Multi-Edit, if you have an oddball compiler or a newer version of a popular one and the error does not display as you would like it to, simply go into the language macro and modify it to fit your needs. Lastly, just because Borland's Pascal does have an environment does not mean that you must use it. Choosing TURBO Pascal as the compiler type for Pascal programs in Multi-Edit allows you to use Multi-Edit and Turbo's command line compiler to their fullest.

In conclusion, if you do any programming from simple to complex in any language, Multi-Edit can help you do it more effectively. I have uploaded a copy of the demo to the HUG bulletin board. Give Multi-Edit a try and I am sure it will help. *



Mainstream Computing

Joseph Katz
103 South Edisto Avenue
Columbia, SC 29205

Nope, I wasn't joking last month about their "8080" telephone numbers linking both Intel Corporation and Microsoft Corporation to the first microprocessor for commercial microcomputers. I don't want to beat this subject into the ground, but I am jealous of my reputation for veracity and, besides, you might chuckle over a few other and cagier examples of magic phone numbers. Take two. Traveling Software, which carries a variety of products for laptop computers in addition to its own LapLink and other software, has the two telephone numbers that hint at the scope of its market. One phone number is toll free for orders, the other is not free for tech support and information. The ordering number is another 8080: 800/343-8080. That's the same link as in the Intel and Microsoft numbers. Where Traveling Software is cagier than most is that its main number is an 8088: 206/483-8088. That's a link to IBM XT compatibles, such as the Zenith Z-158, which use the Intel 8088 microprocessor. But my prize so far for the most direct, no-nonsense telephone number in the microcomputer industry goes to a prominent member of the Heath/Zenith community. First Capitol Computer's toll-free order line is 800/TO-BUY-IT. Unambiguous, isn't it?

I'm still delighted with First Capitol's Z-248-to-Z-386 upgrade and I'm enjoying my explorations of the new Zenith Z-386 that it produced. For $1,995 it's still the best upgrade route I've seen. I understand that First Capitol has upgrades for Zenith computers in addition to the Z-248, but I haven't seen them. You might want to call First Capitol's magic phone number to see what it will do to take you from any of the Heath or Zenith machines you own to any of those you want.

A 1.44MB Floppy Drive in the Z-386

A few days after my Z-386 arrived, I wondered if I'd missed an opportunity. Thanks to version 3.21 of Zenith's MS-DOS (versions earlier than 3.20 don't support 3.5 inch drives) and Micro-Solutions' CompatiCard (which I discussed in my August 1988 article, "How to Add Drives to the Zenith Z-386") the computer could have two more internal floppy diskette drives, additional to the two supported by a Z-386 without CompatiCard. I'd already installed one each of the following for a total of three in my Z- 386: 5.25 inch 360KB; 5.25 inch 1.2MB; 3.5 inch 720KB. Up until now I considered the 5.25 inch 1.2MB drives pretty useless, mostly because there are real problems with any diskettes formatted to 360KB in them. (I know there are tricks and programs that promise to do it, but in my humble experience they tend to be unreliable at the most embarrassing times.) For various reasons, most software vendors distribute their wares on 5.25 inch 360 KB diskettes of the kind used by floppy drives in XT compatible computers, and I'd never encountered one that used the 1.2MB diskettes instead. Then I got PageMaker 3.0, which won the minor distinction of being the first commercial software I've seen distributed on 5.25 inch 1.2MB floppy diskettes. Suddenly I

Address all correspondence to me at 103 South Edisto Avenue, Columbia, SC 29205. I'll try to answer letters accompanied by a self-addressed stamped envelope, but my volume of mail is too heavy for me to promise. Unless it specifically says otherwise, I'll assume the right to publish your letter (edited, if I think that appropriate).

needed, for the first and only time yet, the higher density drive. PageMaker 3.0 is still the only program I know that is distributed in this form factor. When I started thinking about reasons why Aldus chose it, one reason became clear. It seems, in a way, to enforce the required minimum computer for PageMaker 3.0: an AT compatible, which is supplied with the higher density drives. You also can get PageMaker 3.0 on 3.5 inch 720KB diskettes, by the way. And, by the way, the drives to read those diskettes are standard on even faster computers, such as the 80386 machines. So the alternative distribution medium also enforces the minimum requirement.

The opportunity I decided I had missed, though, was the fourth drive CompatiCard allowed me to put in my Z-386. What I wanted was a high density 3.5 inch drive that would store 1.44MB on a floppy diskette. So I called First Capitol, asked for its recommendation, and they shipped me the right kind of Mitsubishi. It was easy as pie to install with the CompatiCard. The trouble was that I hadn't thought to check the price of the right kind of diskettes. When I went to pick up a box of 1.44MB floppy diskettes, I couldn't afford them. Don't let me ruin the surprise for you. Check around and see how much those things cost. Until they come down in price, I can use the 1.44MB drive to read from, write to, and format 720KB diskettes. That's what I've been doing. The only problem I've encountered so far is that the drive won't read a few 720KB distribution diskettes from one software vendor. I don't know why. But I've had no other problems.

I regularly use the high density drive with 720KB diskettes for and from my Z-183. If you want to format a diskette to use that way, remember you can't simply issue a plain, unvarnished Format command if you've used Dsksetup to configure the BIOS ("Basic Input/Output System") so it recognizes the drive for what it is. You'll get a failure and an error message if you try. Use this command instead: FORMAT B:/N:9. The "/N" switch specifies a non-default number of sectors per track. A 1.44MB diskette defaults to 18 sectors per track, so the switch is required to specify the non-default 9 sectors per track used for formatting 720KB diskettes in the drive. Another way to do the same thing is to use Dsksetup to configure the BIOS for a 720KB drive instead of a 1.44MB drive. Then you can use the Format command without the "/N" switch. If you do that, however, you won't be able to read, write, or format 1.44MB diskettes until you reconfigure the BIOS with Dsksetup. You pay your money and take your choice.

First Capitol's RamTop

I've said it before: the first thing to do with an AT-compatible computer that has only 512KB of RAM is to fill out that base RAM to 640KB. The Z-241 and Z-248, for example, both need it. One way to backfill the base RAM is with an extended or expanded memory board that can be configured for the purpose. That's the expensive way, especially if all you want to do is add the 128KB RAM your computer is missing. In that case, you're paying for a board with more capability than you're using. With the high cost of RAM chips right now, moreover, you might not use that capability for some time. Even if you do use that capability now or in the future, it's still the expensive way. One way or another, you'll be losing some of the capacity for which you've paid. Some boards practically require 256KB chips, which means you'll be using only half the capacity of those 9 in the bank of chips you assign to backfilling base RAM. Other boards allow you to "split" between 64KB chips and 256KB chips. If you use 64KB chips for backfilling with those boards, you'll consume twice as many sockets to do the job. You pay your money and take your choice.

A far better choice — because with it you don't lose anything at all — is First Capitol Computer's new RamTop board. It's a memory topper for AT-compatible computers, such as the Z-241 and Z-248. On it are two banks of 64KB RAM chips — exactly the 128KB of RAM you need to fill out the base RAM on such machines. Moreover the board is supplied with 120ns chips and is designed to run as is in a machine running up to 12MHz with no wait states. Nor does it require wait states. They handicap Zenith computers, which are designed to run full speed with no wait states. Zero wait state operation is one strong point about Zenith computers, so it makes little sense to slow them down with add-in memory that requires wait states. The Z-241 runs at 6MHz with no wait states, and the Z-248 runs at 8MHz with no wait states. RamTop clearly is what you want for them.

Installation is simple. Insert the board into any free slot except the 8-bit slot, the slot furthest from the power supply and drives. Then use the computer's Setup routine to recognize 640KB of RAM instead of the 512KB with which it was delivered. That's it. In fact, installation is so simple that the RamTop installation manual is complete and comprehensive in only six half-size pages of generously-spaced 12 pt. type. Incidentally, if you've already backfilled the computer with part of a capacious memory board, consider reclaiming its wasted RAM or capacity by using RamTop for backfilling. Then reconfigure your present memory board for extended or expanded memory instead. In other words, make your system more efficient and more economical by taking better advantage of the memory you have.

Don't let the low price of RamTop — $145 including the RAM — fool you. It's good.

Jay Gold's Home Finance System

Making money is only half the trick. Keeping it is the trick's other, and harder, half. If you do the first half, Jay Gold's Home Finance System III will help you along the second half by making it easy to get your household accounts in order and maintain them that way.

Don't confuse HFS III with those awful checkbook balancing programs, of which the world has far too many and so do I. This software is that rare thing: a thoroughly professional accounts system intended specifically for home use. It lets you keep your household accounts by tracking money through up to 100 asset accounts (checking, regular asset, parent, and reserve) and up to 100 credit accounts (charge cards, stores, loans, utility companies, and the other targets at which we throw money). It's a system designed to fit the way these accounts really work in a family, instead of forcing the accounts into some theoretical model that is alien to reality. Take the concept of a "parent account," for example. It fits nicely the reality of an IRA comprising various investments. Those investments are the individual ."children" of the parent IRA. You make each of those investments a regular asset account, then assign the IRA as their parent account. When you post transactions for the children, they automatically are subsumed by the parent. You know where you stand on both levels at the same time.

HFS III is fast: it's written in assembly language. Because it is, it's also tight, and efficient enough so you could use it on a Z-

183 or other laptop computer with a hard disk. (HFS III is shipped on 5.25 inch disks, however, so you'll have to transfer the files to your laptop.) The system allows password protection, so you should have no worries about reasonable security. It supports a variety of printers, including the Hewlett-Packard LaserJet and Epson dot matrix printers, and even allows switching between two different printers. On those printers, HFS III will produce various reports you can use as a paper trail for tracking your cash flow. It even will print checks. HFS III will run on either the Z-100 with which we are not concerned here, or on the mainstream computers with which we are concerned here. On our mainstream computers, version 3.11 supports color displays.

As I've said, Home Finance System III is thoroughly professional software. If you're making money but not keeping it, or if the job of watching it is wearing you down, or if you're simply not watching it, take a look at Jay Gold's money minder.

Don't let the low price of HFS III — $99 — fool you. It's good.

DOS, MS-DOS, and PS/2 from Que

I'm an unabashed Chris DeVoney fan. He talks well about the two major dialects of the major operating system for mainstream computers: DOS (for IBM's computers) and MS-DOS (for everyone else's, including Zenith's). And he manages to do it on multiple levels without ever talking down. DeVoney's *Using PC DOS* is in its second edition and his *MS-DOS User's Guide* is in its second edition. As their titles indicate, the two books are directed at supplying the user with information about how to use the operating system. Each book will take you from the basics through to the knowledge required for certification as a "power user," that mythical being everyone reveres as possessing all knowledge. These latest editions cover the operating system through IBM's buggy version 3.3 and its equivalent from other manufacturers. Zenith's equivalent is version 3.21 and is not buggy. If you work with IBM machines, you need — really need *Using PC DOS*. For our machines you want the *MS-DOS User's Guide*. It includes appendixes on special features of the operating system as supplied by IBM's competitors. The appendix on Zenith's version is a little superficial. For example, it does note the Mode switch that controls power to the Z-183 internal modem, but it does not point out other switches, only some of which are documented, that control the LCD and hard disk power. The latter two switches are far more important to Z-183 owners. I can't be too hard on DeVoney for that, in part because that's the kind of specialized knowledge that goes into my articles on Zenith's laptop computers and makes those articles useful. The more important reason for my not being troubled by those omissions, however, is that they don't at all weaken the real strength of this book — its explanations of what to do with MS-DOS, and how to do it.

Among its recent slew of other books are three more I think especially useful. One thick volume of more than 800 pages is Terry R. Dettmann's *DOS Programmer's Reference*. The title indicates who it's for and why. It's crammed with the sort of things that systems programmers have to search for in a stack of other books and like to have in one place. I especially appreciate the examples in C and assembly language, and would like even more of those examples. The model for what I like is Que's own *C Standard Library* by Jack Purdum and Timothy Leslie. Every reference book for programmers ought to be like that one. The third special book from Que is Richard Dalton's *IBM PS/2 Handbook*. Nope, Zenith doesn't make a PS/2 equivalent. If you're interested in just what is a PS/2, however, these 360 pages will show and tell you. There's a devotional posture in this book that I find a bit offputting and its elevation of IBM's admixture of technology and marketing into a PS/2 "philosophy" seems a bit much, don't you think? The attitude seems especially surprising from the founding editor of *The Whole Eart Software Catalog*. Get around that theological stuff and stick to the elucidations, however, and you ought to find this book a nice guide for the perplexed.

See you later.

Products Discussed

RamTop $145
First Capitol Computer
16 Algana Drive
St. Peters, MO 63376
(800) TO-BUY-IT

Home Finance System III $99
Jay Gold Software
Box 2024
Des Moines, IA 50310
(515) 279-9821

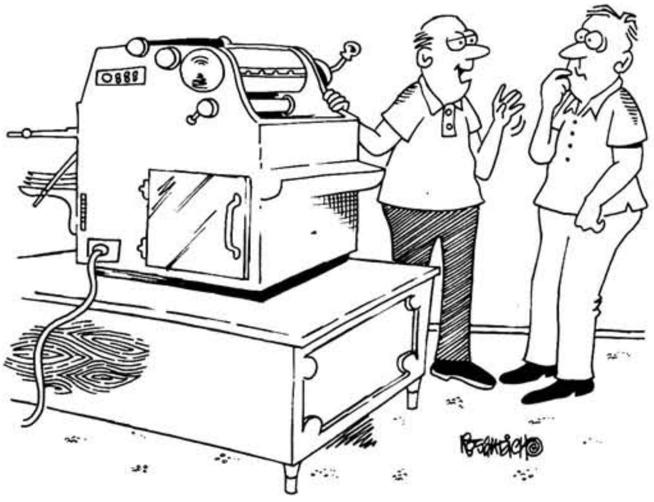
Chris DeVoney. *Using PC DOS.*
2nd Ed. $22.95
ISBN 0-88022-335-9.

Chris DeVoney. *MS-DOS User's Guide.*
3rd Ed. $22.95
ISBN 0-88022-349-9.

Terry R. Dettmann. *DOS Programmer's Reference.* $22.95
ISBN 0-88022-327-8.

Jack Purdum and Timothy Leslie.
C Standard Library. $21.95
ISBN 0-88022-279-4.

Richard Dalton. *IBM PS/2 Handbook.* $19.95
ISBN 0-88022-334-0
Que Corporation
11711 N. College Avenue
Carmel, IN 46032-9903
(317) 573-2500



"I'VE GONE INTO DESKTOP PUBLISHING IN A BIG WAY."

MICROSOFT
WINDOWS $38.00

*** ZENITH SOFTWARE FOR THE *** Z-100 SERIES COMPUTERS

PART NUMBER	DESCRIPTION	LIST PRICE	SALE PRICE
MS-463-1	Z-Basic (16 bit)	$175.00	$12.00
MS-463-7	Multiplan	$195.00	$12.00
MS-253-1	Basic-80 (8-bit)	$175.00	$12.00
CD-463-2	Condor File Manager	$299.00	$12.00
LT-Z100	All 4 Listed Above	$819.00	$40.00

*** SOFTWARE FOR IBM *** COMPATIBLE COMPUTERS

PART NUMBER	DESCRIPTION	LIST PRICE	SALE PRICE
MS-5063-30	Microsoft Windows	$ 99.00	$ 38.00
NU-411	Norton Utilities	$99.00	$ 69.00
NU-413	Norton Utilities Adv.	$150.00	$ 99.00
WP-528	WORDPERFECT 5.0	$495.00	$259.00
SR-3950	dBase III plus	$695.00	$479.00
BO-290	QUATTRO	$195.00	$159.00
LO-311	LOTUS 123	499.00	$359.00
MS-450	MS DOS 3.3 with GW BASIC		$124.00

ZFL-181-93 Laptop Portable with the amazing Supertwist LCD backlit screen. 640K RAM, two 720K 3.5" disk drives, clock, P & S Ports $1649.00

*** VIDEO MONITORS ***

ZCM-1490	ZENITH Color Flat Screen VGA	$718.00
XC-1409C	Mitsubishi Color CGA 640x200	$296.00
XC-1410C	Mitsubishi Color EGA 640x350 0.40mm.	$419.00
XC-1430C	Mitsubishi Color EGA 640x350 0.31mm.	$489.00
AUM-1371	Mitsubishi Color EGA Multi-Scan 920x580	$529.00

*** UPGRADE ACCESSORIES FOR *** ZENITH PC COMPUTERS

Z-171 CMS Lite Drive 10 Meg Hard Disk Internally mounted and with automatic head parking. This hard disk replaces one floppy drive and runs on either AC or battery. List price $995.00 but now available while supplies last at $495.00

AUTOFAX IMAGER Video Tape Backup System Use your VHS or Beta Video Recorder to backup your hard disks. With this interface card and cable set connected to your video tape recorder, you can backup up to 110 Megabyte on a standard video tape. The required software is included. $212.00

SmartWatch from FBE Research Installs in ROM Socket on CPU Board in Zenith computer series Z-100/138/148/150/160. This clock/calander contains a ten year battery and keeps your computer informed of both time and date at each boot-up. Instructions and software included. $38.00

Z-150 Series Hard Disk Drive Kit Includes new generation High Speed (28 MS) Seagate Drive with Auto Park heads. Each kit is complete with controller card, cables, hardware and instructions to mount the Hard Disk under your two floppy drives in the Z-150 series computers. 32 MEG ST-138/150 Kit $383.00

INTERNAL MODEM Fully Hayes compatible (software included)

1200/300 baud	$94.00
2400/1200/300 baud	$159.00

EXTERNAL MODEM Fully Hayes compatible (software included)

1200/300 baud	$137.00
2400/1200/300 baud	$199.00

ZENITH KEYBOARDS These were used in a training class but function like new. For some unknown reason, most have a few light scratches near the Zenith label. Specify XT (Z-140/150/160 Series) or AT (Z-200 Series). A bargain at $24.50

Z-449 VIDEO CARD We made a special bulk purchase on this latest model Zenith video card. This is the standard card for the new ZCM-1490 Zenith VGA Monitor. The card will also drive EGA, CGA Color and MDA TTL Monochrome Monitor. List $499.00 . . . . Payload Price $147.00 While our supplies last.

Z-148 Hard Disk Drive Kit Includes the Hard Disk Drive and controller in the kit above plus the PTZ-148 Expansion card described below. Each kit includes all cables, hardware and instructions required to replace one floppy drive with a high speed low power Hard Disk Drive. 32 MEG ST-138/PTZ-148 Kit $492.00

PTZ-148 Expansion Card for Z-148 Includes 2 expansion slots plus a clock/calander. $118.00

VMM 150 Video Eliminator for Z-150 computers. Allows use of EGA color card $45.00

*** UPGRADE ACCESSORIES FOR *** Z-100 SERIES COMPUTERS

High Density 1.2 Meg Drives. External floppy drive set-up complete with drive, power supply, case and cable. Ready to connect to your 8" floppy controller $224.00 . . . . Dual Drive Unit $369.00

SmartWatch by FBE Research. If you don't have a clock for your Z-100, get this one. More details under Z-150 upgrade listings $38.00

Gemini Emulator Board. Makes the Z-100 compatible with the IBM PC library of programs. $432.00

UCI EASY PC. IBM PC Emulator. Makes your Z-100 IBM Software Compatible. Full 8 MEG operation, color graphics and audio compatible. Retail $699.00 . . . . Payload $477.00

UCI Easy87. Add an 8087 Numeric Coprocessor. $69.00 for the board without an 8087 Chip. With 5 MEG 8087 $188.00 or with 8 MEG 8087 installed . . $234.00

ZMF100A by FBE Research. A modification package which allows 256K chips to be used on the old-style motherboard to reach 768K. Simple assembly with no soldering or trace cutting. Compatible with Easy PC and Gemini Emulator. . . $60.00 Requires 27 of the 256K RAM chips to complete the kit.

UCI Memory Upgrade Pal Chip Set For the Z-100's with the newer motherboard part number 181-4918 or greater. This chip set allows the installation of 256K RAM chips on the motherboard. With the addition of 27 256K RAM chips, a total memory or 768K is obtained. PAL Chip Set $64.00

UCI Memory Upgrade Card We recommend this one highly. The board has sockets for up to 2 MEG of RAM. With no RAM installed $288.00. Add $35.00 for EasyDrive RAM Drive Software if desired. Either 64K or 256K RAM chips may be used to complete this kit.

UCI EASY-I/O S-100 board that provides IBM PC communications port compatibility with your EasyPC. EasyI/O-1, One Serial Port $91.00. EasyI/O-2, Two Serial Ports, One Game Port, Clock- Calendar $127.00

UCI EasyWin Winchester Drive Systems at reasonable prices. Complete Hard Disk Systems for mounting inside your Z-100. Systems complete with Seagate Drives, 21 MEG $598.00, 31 MEG $634. System without Drive $317.00

CDR Z-100 Speed Module Run your Z-100 Computer at 7.5 MHz. Installs easily with no soldering. Externally switchable between Speed and Normal mode. Payload $44.00

*** FLOPPY DISK DRIVES ***

MITSUBISHI MF501	5.25" 48 TPI DS/DD 320K/360K	$ 95.00
MITSUBISHI MF504	5.25" High Density 360K/1.2 MEG	$124.00
MITSUBISHI M-353	3.5" in 5.25" frame 720K	$103.00
MITSUBISHI M-355	3.5" in 5.25" frame 1.44 MEG	$139.00
	M-355 Software Driver	$ 19.00

M-355 runs on AT compatable or special controller only.

** SEAGATE HARD DISK DRIVES ***

ST-225	21 MEG, 65 MS, With Western Digital Controller & Cables	$ 259.00 $ 313.00
ST-138	31 MEG, 28 MS, Auto Park Heads With Western Digital Controller & Cables	$ 329.00 $ 383.00
ST-238	31 MEG, 65 MS, RLL With Western Digital RLL Controller & Cables	$ 278.00 $ 323.00
ST-251	42 MEG, 40 MS, Auto Park, Software With Western Digital Controller & Cables	$ 384.00 $ 438.00
ST-251-1	42 MEG, 28 MS, Auto Park, Software With Western Digital Controller & Cables	$ 465.00 $ 519.00
ST-4096	82 MEG, 28 MS, Auto Park, Software.	$ 697.00

PAYLOAD COMPUTER SERVICES

15718 SYLVAN LAKE
HOUSTON, TEXAS 77062
PHONE (713) 486-0687

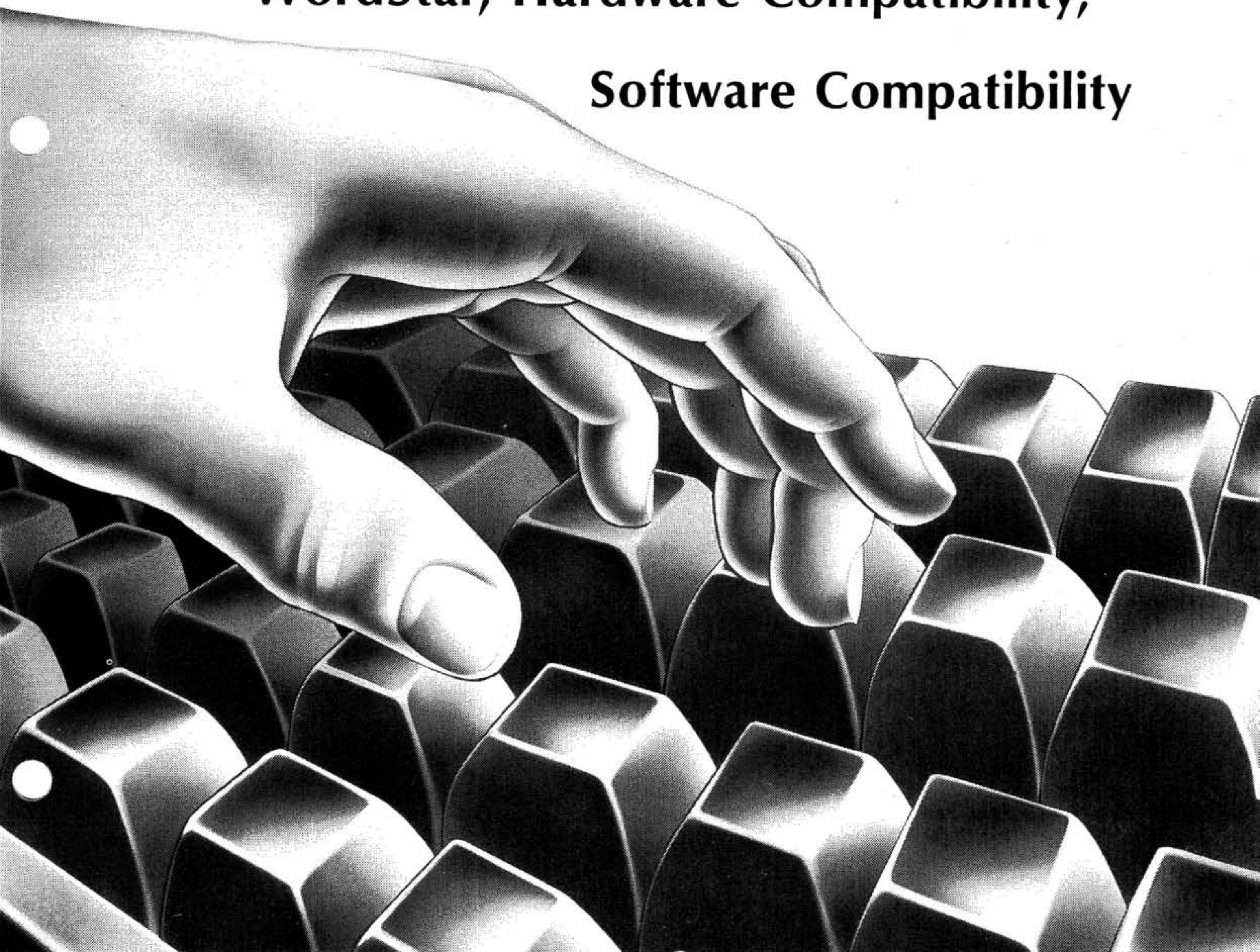
MasterCard VISA

Your satisfaction is guaranteed. All hardware carries a 90 day Payload warranty. VISA and MASTERCARD orders welcome with no surcharges. Add $5.00 to all prepaid orders for handling and shipping in Continental USA, we pay the balance. Actual shipping costs for foreign, overseas and net billing orders to approved accounts. We accept purchase orders from schools, government and approved accounts. Mail or Phone your order for prompt service. Texas residents please add 8.0% state sales tax.

On The Leading Edge

William M. Adney
P.O. Box 531655
Grand Prairie, TX 75053-1655

WordStar, Hardware Compatibility, Software Compatibility



When discussing word processors, you will often notice that somehow WordStar manages to enter the discussion. For one reason or another, many successful professionals still use WordStar even though it is not as powerful as some of the other word processors that are available today. And it is usually pretty easy to provoke an interesting argument with a die-hard statement like: "WordStar is STILL the best word processor around". In today's software market, the distinction of what really constitutes a word processor is getting quite blurred. The differences are also interesting because various manufacturers use terms like "word processing", "word publishing", and "desktop publishing"; sometimes indiscriminately. I don't intend to discuss the merits and demerits of WordStar here because, in the final analysis, the best word processor (or whatever you want to call it) is really the one you like the best. But there is one thing you should know about WordStar, and that is its design philosophy.

Why should you care about the design philosophy of WordStar? No matter what your favorite word processor is, it is still difficult to NOT run into WordStar in some form or another. It is interesting to observe that many people know more than they think about WordStar even though they have never seen it. There is more than a hint of the influence of the WordStar design philosophy in unrelated programs like Borland's SideKick, Ecosoft's CED editor (furnished with the C compiler), and Living VideoText's ThinkTank. To understand why this has happened, let's go back about a century (in microcomputer software development) or so (1979 actually) to see what happened.

In the Old Days

WordStar was initially designed when there was no such thing as compatibility in computers or in disk formats. One could choose from an incredibly wide range of computers — some, like the H-89, were self-contained meaning that the computer and CRT were in one unit. Other configurations included the CompuPro S-100 series where you actually built the computer by buying the "box" (with a motherboard and power supply) and then bought various boards to build the computer: CPU, memory, disk controller, and so on. And you also had to buy some kind of terminal (a CRT and keyboard), such as the Heath H-19 or whatever. The availability of these different terminals, and their hardware differences, made it necessary that any software could work in some kind of common way. Most had function keys, but some did not. There was no such thing as a "standard keyboard". Some keyboards did not even have the cursor keys as we know them today. That presented an interesting problem to software designers. In order to cope with all of these keyboards, MicroPro came up with an interesting solution since virtually all of these keyboards had a CTRL key: Use the CTRL key with various letter combinations to perform a function, such as moving the cursor.

In the old days, the CTRL key was just about the only "modifier" command key commonly available on nearly all 8-bit microcomputers. In the case of this specific key, it also happened to be located in the standard position just above the left shift key on nearly all keyboards which made it a natural choice for designers and touch typists. For that reason, some thoughtful designer decided to use the CTRL key to control all functions of WordStar. That solution was particularly attractive since it meant that the program could be operated in exactly the same way whether you were using a Heath, Kaypro or Osbourne system. If you knew how to run WordStar on a Heath system, you could also run it on a Kaypro or Osbourne. The solution began with the definition of the "WordStar Diamond" which is still in use today in programs that I have mentioned.

What is the WordStar Diamond?

Sometimes you will hear that WordStar commands are cryptic, and many potential new users shy away from the program because they have heard that. Nothing could be further from the truth. There really is a logical method for assigning commands to various keys, and it begins with an understanding of the basic WordStar Diamond.

Cursor control is an obvious fundamental requirement for any word processor, but some of the old-time terminals did not have cursor keys. If you stare at a keyboard for a minute, you can see that the E-S-D-X keys on the left side of the keyboard form a reasonably good imitation of a diamond-shaped pattern. That is the basic key to the WordStar Diamond as shown in Figure 1.

```
  E             (Up)
 S D       (Left  Right)
  X            (Down)
```

Figure 1
The Basic WordStar Diamond

These four keys, used in combination with the CTRL key, were used to move the cursor in the obvious way. That is, CTRL-E (^E) moved the cursor up one line, CTRL-D (^D) moved the cursor right one character, CTRL-S (^S) moved the cursor left one character, and CTRL-X (^X) moved the cursor down one line. The values shown in parentheses indicated a "shorthand" used in the MicroPro documentation where the caret symbol (^) was used to indicate the press and hold for the CTRL key. From a user view, that diamond combination had two clear advantages.

First, it represents a logical, meaningful, intuitive, and STANDARD way to move the cursor on any keyboard even if cursor keys were not available. The second advantage (for touch typists, at least) was that the CTRL key used with these letter combinations was easy to type because the CTRL key was always in the same place.

From a designer perspective, this solution was attractive because it simplified the design of the software considerably. The use of standard key combinations made it easy to define how the software should work. Not being content with that, WordStar's designers carried the logic a little further by expanding the diamond pattern to include two other helpful functions.

```
 E                 (Up)
AS DF     (Word) (Left  Right) (Word)
 X                 (Down)
```

Figure 2
The WordStar Word
Left/Right Diamond

Since ^S moves the cursor one character to the left, the use of the diamond logic says that ^A will move the cursor one WORD to the left. Similarly, ^F moves to the beginning of the next word to the right of the current cursor position. Current versions of WordStar also allow the use of the right/left arrow keys with the CTRL key to move to the previous or next word which is also intuitive. Many of today's other word processors also use this same logic so there is no learning curve involved for this function.

Old keyboards did not have anything like a standard PgUp or PgDn key, so something had to be done about that too. The diamond was again expanded to provide those functions as shown in Figure 3.

```
  ER                 (Up) (Page)
 AS DF    (Word) (Left  Right) (Word)
  XC                (Down) (Page)
```

Figure 3
The WordStar PgUp/PgDn Scroll Diamond

As you can see, the PgUp (^R) and PgDn (^C) commands maintain the same kind of intuitive logic by following the diamond pattern. These two commands are used to SCROLL the screen up (i.e., backward, toward the beginning of the file) or down (i.e., forward, toward the end of the file). Instead of scrolling an entire screen at a time, it is sometimes useful to be able to scroll only a line or two, and WordStar has commands in the diamond pattern for that too as shown in Figure 4.

```
 WER         (Line) (Up) (Page)
AS DF   (Word) (Left  Right) (Word)
 ZXC        (Line) (Down) (Page)
```

Figure 4
The WordStar Line Up/Down Scroll Diamond

Now you have total control of the screen with the line scroll up (^W) and a line scroll down (^Z) commands. These commands are quite useful when you only want to see the end or the beginning of a paragraph by scrolling a line or two instead of the entire screen. One nice thing about these commands is that the cursor stays on the same line, and only the screen "moves".

The point is that the WordStar commands may not be as cryptic as you have been led to believe. There is a definite logic to the commands which is there by design, and with a little practice, these keystrokes are second-nature. And if you know these commands on ANY computer (e.g., an '89, '100 or PC compatible), the same commands will work on any OTHER system with WordStar. That is one of the reasons that I use WordStar so much — I originally began using it on my '89, then on my '100, and now I use it on my '248 and '386 systems. It is difficult to think of any other program that you can use like that on all of those different systems. To continue with this design philosophy, let's look at the features in the different menus.

Other Basic WordStar Commands

All of the commands discussed so far are listed on the Edit Menu that is normally displayed when you are editing a file. The cursor and scroll commands are listed as previously described, and there are a few other important commands that are also listed. When WordStar is started, it normally begins in the Insert Mode. This can be changed to the Overwrite Mode using ^V (or the Insert key on PC compatibles) as a toggle. A toggle is a command that works like a light switch — it turns a function on and off.

There are several basic commands that allow you to delete one or more characters from text. Early versions of WordStar implemented the Backspace key as a nondestructive delete which means that it functioned just like the ^S or left arrow key — that is, it moved the cursor backward but it did not "erase" a character. Current versions of WordStar still allow this function, but most people seem to prefer the destructive backspace.

If you want to delete the character at the cursor position, the ^G does that. In PC compatibles, the Del key performs the same function. To delete a word (or the rest of a word) to the right of the cursor position, you can use the ^T. And a ^Y will delete the entire line of text at the current cursor position. Some people like to remember that a ^Y "Yanks" an entire line of text out of the document.

WordStar versions up to 4.0 do not automatically reformat a paragraph when you insert text, so you must use a ^B to reformat when a line extends beyond the ruler margins. And a ^J can be used to see the help screen if it is not already displayed. If you are using WordStar Version 4 and there is no menu displayed when you start the program, you can easily change that by typing ^J^J followed by a 3 and a RETURN to display all menus. All of these commands, and more, are shown on the basic Edit Menu that is typically displayed during normal editing.

Aside from the Edit Menu, there are four other menus that may be displayed during editing: the Quick Menu activated by ^Q, the Block and Save Menu activated by ^K, the Onscreen Format Menu activated by ^O, and the Print Controls Menu activated by ^P. When you know some of these basics, the rest of WordStar is a piece of cake because most of the commands are shown on menus.

The WordStar Menus

I think that one of WordStar's nicest features is that a menu (you can also call it a help screen) can be displayed on the screen at all times to show you which commands do what. Even nicer is the fact that, once you learn the commands, you can tell WordStar NOT to display these menus which essentially means you can use the entire screen for writing and editing. WordStar essentially contains five menus that can be displayed with appropriate help information: the Edit Menu (default display), the Quick Menu (^Q), the Onscreen Format Menu (^O), the Block Menu (^K), and the Print Controls Menu (^P).

In order to continue with the logical discussion of moving the cursor and the WordStar Diamond, let's look at the Quick Menu that provides an expansion of those functions starting again with the basic E-S-D-X diamond.

```
    ^QE          (Up)        (Top of Screen)
 ^QS ^QD    (Left  Right)    (Line)
    ^QX         (Down)       (Bottom'of'Screen)
```

Figure 5
The Basic WordStar Quick Menu Diamond

When preceded by ^Q as shown in Figure 5, the basic diamond logic is maintained in the same relative way, except that the cursor movement now includes the entire screen display. For example, the ^QE command quickly moves the cursor to column 1 on the first displayed line, and the ^QX command moves the cursor to the last character column on the last displayed line — that is, the top of the screen or the bottom of the screen. Similarly, ^QS and ^QD move the cursor to the left and right ends of the current line.

If you remember that ^R is the PgUp command, then it is logical to assign ^QR to mean "PgUp to the Top of the File". And of course the ^C PgDn command becomes ^QC which takes you quickly to the end of the file.

This has been a short introduction to the basic movement of the cursor using the

WordStar commands. Many of these commands have other equivalents on PC compatibles to make them easier to use, but if you know these basic commands and the logic behind them, they will work on just about any computer system with nearly any version of WordStar that you will find. Next time, we will spend some more time looking at the logic of some of the other WordStar commands, but now let's look at a question about the computer systems that we use.

Zenith Hardware Compatibility

In the last few months, a number of you have asked questions about software and hardware compatibility with Zenith PC compatible computer systems. As usual, there is no single or easy answer to that question, and the compatibility question depends on a number of factors.

For example, Lynda Tom (Oakland, CA) recently wrote a letter to HUG asking about hardware compatibility with the '151. One particular question concerned hardware that can be used to speed up the '151 beyond its 4.77 MHz speed. That is a particularly tricky question because there are more ramifications to speeding up a '151 than you might suspect.

Before we get too far along in that discussion, I think it is important to place this in the proper perspective. First, you need to recognize that ALL hardware designers have a rather narrow view of their particular world. It does not matter whether you are talking about a car ("See Mr. Goodwrench", and "Use genuine GM parts"); a vacuum cleaner ("Replace the paper bag in your cleaner with a QUALITY made bag by Eureka"); or a computer ("Any attempt to alter or modify the design, or to use this device in a manner other than described in the Owner's Manual, will void the Warranty and release the manufacturer from any responsibility for its operation" — from the Z-200 PC Series Computers Owner's Manual). It really doesn't matter what hardware you want to look at because virtually all manufacturers will only guarantee proper operation when you use "factory authorized" parts and accessories. If you don't, all bets are off. In the worst case, you may even void a warranty if you are not careful.

Every once in a while, you may see a comment that a specific Zenith computer is not quite as compatible with an IBM computer as it should be. In the early ROMs for the '151 for example, there were some specific instances where PC software would not run correctly or at all because of a firmware incompatibility in the system ROM. Today, nearly all of those compatibility problems have been fixed although one can occasionally find a situation where the Zenith system ROM still may have a "glitch" because they are DIFFERENT — by necessity and by design — from their IBM counterparts to avoid legal problems.

For example, I recently found that Borland's new Sprint word processor does not function correctly on a '151 with ROM version 2.3B. When we tried to print something within the Sprint program, DOS would display a "Divide overflow" message. I understand from Borland that this same problem also occurs on '148s and '158s with older ROMs. The solution is to get a new ROM set which is under $25 for all Heath/Zenith computers that I have checked. For the '151, the cost is under $15. Sometimes a new ROM will fix that kind of problem, but I would not want to say that it always will. Sometimes those differences cause real problems for one reason or another as illustrated by this example.

As a consultant, I work with a number of different types, brands, and sizes of computers. And in my nearly 22 years of computer-related work, I think the best thing that has happened was when IBM effectively set the standard for microcomputer compatibility. But even then, IBM microcomputers have never been completely compatible with each other. The best example is probably a compatibility comparison (or lack of it) of the IBM PC with the ill-fated PC Jr., not to mention that some software developed for the original (1982 vintage) PC would not run on later models due to a change in the system ROM by IBM.

In short, there is no such thing as total PC compatibility even among the various models of IBM microcomputers and sometimes within the same model. Given that knowledge, it is silly and completely unreasonable to expect that any given brand and model computer will be 100% compatible with any other. I have to admit that I am immediately suspicious of any computer advertisement that says it is 100% IBM compatible because it just can't be from a technical perspective. It would have to use exactly the same circuit boards that are a trace-for-trace (and chip-for-chip) duplicate of IBM which would of course cause legal problems for the manufacturer. And all firmware, such as the system ROM, would have to be a bit-for-bit duplicate of the corresponding IBM ROM which would again cause legal problems.

And so we have the incredible variety of PC compatible computers from which to choose. In the general case, some are more PC compatible (e.g., Zenith and Compaq), and some are less (e.g., the Tandy 1200). During one testing session, I was absolutely astounded to see that the then popular Word Perfect 4.1 caused a completely loaded Tandy 1200 (with 640 K and a hard disk) to go into a hard system freeze (i.e., CTRL-ALT-DEL did not work) when we attempted to use the thesaurus.

By now I think you can see the scope of the problem, but let's get back to Lynda's original question about speeding up the '151. There are several options ranging from Software Wizardry's popular Wildfire kit to "accelerator boards" (e.g., 80286) that you can add to your system. How do you know whether these and other hardware options are compatible with your Heath/Zenith system?

My best recommendation is to buy from vendors who specialize in Heath/Zenith computers, and that is a specific plug for reading the advertisements in this magazine to find what you want. First Capitol Computer (also known as Software Wizardry) is probably the oldest advertiser in REMark. Since I have at least talked to most of the REMark advertisers (and have reviewed some of their products in this column), I think you will find them quite reliable and helpful. Moreover, they are especially interested in helping you solve a specific Heath/Zenith computer problem since that is their business. If you take a look at this year's issues of REMark, you will probably find an advertisement for something that will solve a specific hardware problem or question. If you don't find what you are looking for, take a moment to write or call one or more of these vendors with your questions. My experience is that these people are quite knowledgeable and willing to help you solve problems.

For example, I have found that these vendors know that you may have a problem in trying to add a "standard" PC-type hard disk controller to some of the dual-speed systems, such as the Z-148 and the various Z-150 series computers. Although the standard PC (i.e., non-Zenith) hard disk controller (e.g., Omti and Western

Digital) may work just fine at the usual 4.77 MHz clock speed, there are at least a few that will not work at 8 MHz on these systems. For example, Frank Gaenger (Prescott, AZ) wrote to me last March about a problem on his Z-158 with a Seagate ST-225 and an Omti controller where the FORMAT program gave a '"Format not supported on drive C:" error message. More recently, Dick Bidwell (Huddleston, VA) had a problem in running Digital Research's GEM on his '151 that had a "turbo speed-up" kit and many other hardware upgrades.

I have seen that particular FORMAT error message before, and my experience is that it usually is a result of using a non-Zenith hard disk controller that, for one reason or another, cannot operate properly at 8 MHz. In those cases, the usual cure is to only run the system at 4.77 MHz (to be sure that the controller is good at the usual speed), but one needs to know that the ROM on the Zenith-brand hard disk controllers was specifically designed to operate correctly at the higher speed in Zenith designed hardware. The bad news is that many of the upgrades for the Heath/Zenith PC compatibles cost as much as 50% more than the standard PC hardware. So, many people avoid some of these outrageous prices and try to save money by buying a standard PC compatible whatever through mail order places that do not understand Zenith equipment. And sometimes these "whatevers" do not work correctly or at all because they were not designed to operate in the Zenith hardware environment, such as running at 8 MHz.

As far as I know, there are only two manufacturers that provide hard disk controllers that are guaranteed to work satisfactorily on these dual-speed systems. The first is, of course, Zenith, and a few of the REMark advertisers sell hard disk systems with the ZDS ROM on the controller. The second source is First Capitol Computer, and Tom Jorgenson told me that they have specifically designed a ROM to work in Heath/Zenith hardware. Although you may find another hard disk controller that SEEMS to work right on your system, it may be false economy to take a chance in losing your data at the wrong time because of some kind of controller difference.

Unfortunately, there is more to the compatibility issue than just solving a hardware question or problem. In more than a few cases, the hardware may generally be compatible with your Heath/Zenith system, but some software is not, such as the GEM problem that Dick Bidwell has. How can that be?

Zenith Software Compatibility

When I first got my '241 (and later converted it to my '248), I was initially surprised to find that the Microsoft Flight Simulator program would not run on my system. It appears that the version of Flight Simulator I have uses a form of copy protection that prevents the program from running at faster than the usual PC clock speed of 4.77 MHz. In my opinion, the speed restriction is just a particularly insidious form of copy protection because you would have to pay additional money for an upgrade when you get a new computer. It is also one reason that some of these Heath/Zenith computers have a dual-speed mode. The bottom line is that the program will not run on my '248 system.

Does that mean my '248 is not PC or AT compatible? Of course not — that is going from the ridiculous to the sublime. But look at all of the things you MUST "know" before you can correctly define the problem. First, you need to know that one form of copy protection is sensitive to the clock speed of the CPU. Then, you must know that some programs, especially games, use this form of copy protection. Of course, you must also know the clock speed of your computer. And finally, do you have any unexplained programs (e.g., general program failure or system freeze) with general software, such as word processors or spreadsheets? If you don't have problems with other programs, you can probably correctly deduce that the problem is "something" in that game program, although you may not know specifically whether it is speed-sensitive or not. Unfortunately, it is altogether too easy to blame Zenith for a lack of compatibility in the hardware design (or in MS-DOS) when one does not have any idea what the problem is. In many cases, a particular program causes the problem that has nothing whatever to do with Zenith. By the way, Flight Simulator will not run on any of the IBM ATs that I have tried either, so I have concluded that it must be one of the games that checks the CPU clock speed.

I had a similar experience with a pinball type of game that I bought a few months ago. It was also copy-protected although the outside of the box did not say so. After reading the basic instructions, I tried to run the program, and it absolutely locked up my system — that is, the usual CTRL-ALT-DEL had no effect. Even though I followed the basic start-up instructions, there was a hidden note in the back of the manual that the program would ONLY work with a CGA video card. My system has a Vega EGA card with a NEC MultiSync monitor which explained the problem. Even though I had opened the package, I took it back to the store (a local discount software retailer) and got my money back since the program would not work on my system.

Now you can see the nature of this problem, but what about running GEM on Dick Bidwell's '151? Well, let's take a quick look at his hardware configuration. He has upgraded his '151 with a speed-up kit (not Wildfire), a video card eliminator so that he can use a Quad EGA+ card with a MultiSync monitor, a Logitech bus mouse, a Western Digital hard disk controller (probably non-Zenith but he did not say), and a V-20 CPU in place of the 8088.

What is the specific problem with GEM? In Dick's words: "Everything else has always run well at the higher speed, but not GEM. I have tried everything to see if something else is causing the problem, such as booting up without any resident programs [an excellent first choice in troubleshooting this problem by the way — WMA], but it doesn't help. I have tried booting up slow [4.77 MHz] and switching to fast [7.5 MHz] and at the point I switch, from then on it goes bananas — crazy colors, things scattered all over the screen, etc."

It also helps to know that GEM (like Microsoft Windows) uses bit-mapped graphics as part of the program which, by its nature, requires absolute compatibility with the video display system, especially the video card. This is another example of a program that works just fine at a slower speed, but does not work at a "turbo" speed like the hard disk controller I mentioned earlier. Based on this background, it is reasonable to assume that there is a problem in the video system that is directly related to the CPU clock speed. Pinpointing the problem is more difficult.

The first step is to remove the EGA card and the video card eliminator so that the '151 can be returned to the standard CGA mode with a CGA monitor. If GEM works properly at both normal (4.77 MHz) and turbo (7.5 MHz) speeds, then you can assume that the problem is directly related to the EGA/video card eliminator combination where something in that hardware combination does not "understand" the system is running at 7.5 MHz. Since the Quad EGA+ is known to run fine in 8 MHz Z-200 or AT systems, the problem is most likely related to the video card eliminator and/or the turbo hardware. In other words, the system, in general (and the GEM program, in particular) is not communicating with the EGA card in exactly the same way that it would on a Z-200 system. And so you get strange results similar to the "Wild Interrupt" message on the Z-100, except that GEM makes the display go wacko.

There is one other possibility for a quick fix that may or may not solve this kind of problem: Try the latest ROM in your system. In some cases, that will fix a problem such as occurred with the "Divide overflow" message that I mentioned earlier in this article. If a new ROM set does not fix the problem, what can you do next?

About the best you can do in a situation like this is to find out WHAT is causing the problem, but there is usually little or nothing that you can do to effectively solve it. In this case, it may not be possible to get GEM running in the turbo mode. Does that mean that the '151 is not PC compatible? Not in this case.

The problem is that this particular '151 has a hardware configuration that exceeds, by far, its intended design. One of the peculiarities of the '151 is that it was not designed for a separate video card, and in order to go to EGA, some kind of hardware "replacement" must be performed that amounts to emulation. And since the '151 was only designed for 4.77 MHz, there are all kinds of potential problems lurking out there, not to mention possible problems in speeding up the system AND replacing the video adapter.

Although this section began as a discussion of software compatibility, there is no way to intelligently discuss it without dealing with hardware-related issues as you have seen. These are a few very specific instances that point out some of the possible dangers of modifying the system beyond its intended design. Even worse, there is no way, in many cases, to predict what modification will cause a problem with a specific software program until you actually try it. Perhaps you will get lucky and find that a new ROM set will fix a specific problem. If that does not work, it may be something related to MS-DOS, but that is not quite likely. This is one of those situations that occurs in many brands of computer systems, including IBM, and it poses a real problem for us.

Hardware, Software, Compatibility, and Prices

What is the real answer to this morass of compatibility between hardware and software? Most of you have probably heard about the standard Zenith "party line" answer that non-Zenith products are not supported and are not guaranteed to maintain the PC compatibility level inherent in the system. While that sounds like kind of a cold answer, it is difficult for me to see how Zenith could respond any differently. Unfortunately, there is the added implication that any non-Zenith hardware added to our systems may or may not work with all software. That has the further implication that we must pay outrageous prices for items, such as hard disks and controllers, in order to maintain any kind of assurance that we won't have compatibility problems.

My objection to this state of affairs is not the hardware, but the prices as I have said before. If, for example, you want to buy a Zenith hard disk upgrade for a Z-159 computer, you might (for a very SHORT time) want to consider buying an HWD-20 Winchester Upgrade that includes a 20 MB drive for $449.95 as shown on page 85 of the Fall 1988 Heath Catalog (No. 213). That is extremely difficult to reconcile with the fact that I can get a PC compatible 30 MB hard disk (yes, 30 MB) and controller at a local discount store for a tad over $300. While I would suggest that most of us would not object to a modest premium, say $25-50 or so, for Zenith hardware; a 50% premium of $150 is far too much.

In perusing the ads that appeared in the June REMark, I note that Payload Computer Services (page 16) has a listing for a "Winchester Hard Disk Drive Internal Setup" with a 30 MB drive for the Z-148/150 series computers for $315. To me, that seems like a much more reasonable and competitive price for that hard disk, and since it is shown under the Z-148/Z-150 heading, I assume that it works fine even though I haven't tested it. I have more than one letter that gives Payload high marks for customer service and satisfaction. You may want to check with them if you are considering a hard disk or any other hardware for that matter.

Other Alternatives

Assuming that you have an "old" '151 or similar system, is it worthwhile to try to upgrade it or should you consider getting the "latest and greatest", such as a '286 or '386 system? If you are like me, the biggest single factor is cost, but I think there is at least one other consideration — would you believe software compatibility?

Considering some of the potential problems with upgrades that I have mentioned, I think the most important question is: What do you REALLY want to USE your system for? For example, if you use your system primarily for word processing like I do, you will probably find that a high-resolution display is particularly important to you. If you want to do any programming using a compiler (like C) or work with a large database, a hard disk is nearly indispensable. Or if you work with a large spreadsheet, you may require a big chunk of expanded memory. So far, these are some basic "requirements" based on the kind of work that you might want to use your computer for.

On the other hand, there are some definite "wants" that would also rate high on most lists. The first one that occurs to me is speed of processing which does not necessarily equate to the clock speed of the CPU, but that's a good first approximation. There are other factors that must be considered here, such as the number of wait states for memory, but I sometimes think that using my 8 MHz '248 is probably overkill for word processing even though I particularly like the speed. While my 16 MHz '386 is faster yet, it is definitely overkill for my uses.

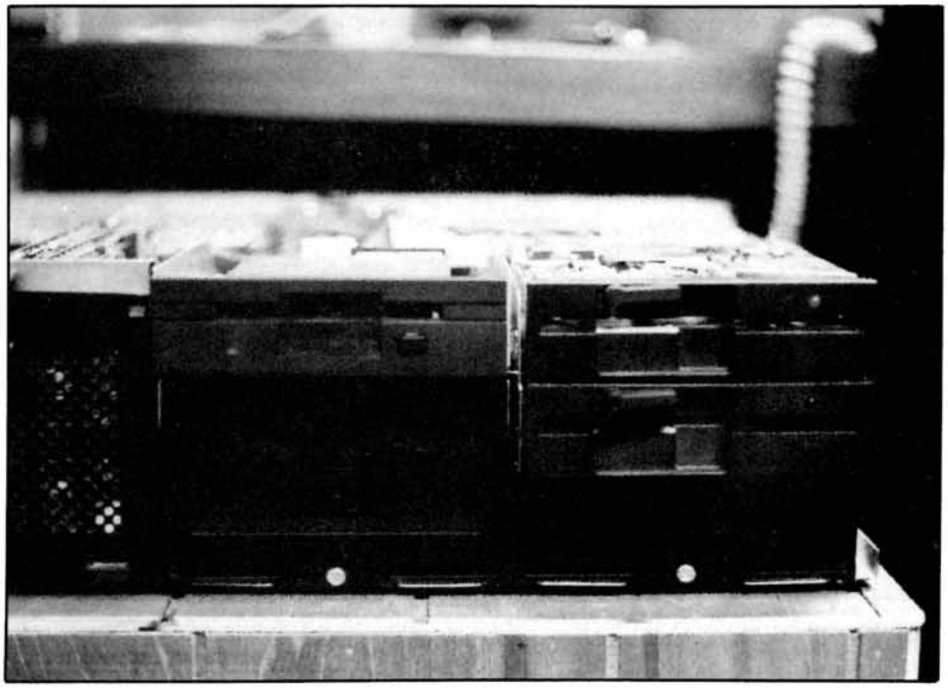
And so, it is easy to be torn between all of these factors with the basic question: "Should I buy a newer and faster computer, or should I upgrade my current one?" That is a really tough question, and there are several ways to approach it. I will go through some of my thought processes on that in the next issue.

Powering Down

Next month marks the fifth anniversary of my writing articles for REMark in this column. For those of you who have been HUG members for over five years, I will be

Continued on Page 61

Three Drives on a Z-248



Mike Raick
1159 Forest Lane
Birmingham, MI 48010

I recently acquired a Zenith Z-181 portable computer. It was acquired officially as a business computer, to allow me to work in various environments, such as motels, airports, and such places, and I felt I could make much better use of this 'time' with the aid of a portable computer. In actuality, the Z-181 has proven to be everything I'd hoped for and more, and I find myself doing a large portion of my work on it. I can enjoy a pleasant day on the patio, and still work (or play) on the computer, as well as utilize it any place I am willing to carry it.

With a Z-151 in my office, and a Z-248 at home, I was faced with the problem of transferring data between these 3 machines. The answer appeared to lie in a transfer problem, to transfer files between the 3 machines, and consequently, I purchased the "Brooklyn Bridge", and found out it did the job quite nicely, as long as the portable was physically located near the machine to which it must talk. This, however, was not the case, and I spent a considerable amount of time carrying the Z-181 around, needlessly.

I thus began to consider the possibility of installing 3.5 inch drives in the 2 desktops. I first acquired a 3.5 inch for the Z-151, and with considerable apprehension, installed it in place of one of the 5.25 inch drives. I purchased it from my friendly Heath store in Farmington Hills, MI, and was shown a Mitsubitshi brand drive. One of it's major attractions was an adaptor board on the back, which converted the header connector to the 5.25 inch type (a circuit board with dual rows of 17 gold 'fingers') and the small 4-pin power connector to the larger type found on 5.25 inch drives. The installation was completed quickly, due to the fact that the drive was mounted in a 5.25 inch frame. I booted the machine, ran DSKSETUP to inform DOS of the presence of a 720K drive, and much to my surprise and pleasure, it ran flawlessly, and continues to do so. This particular brand appears to be excessively noisy, but otherwise functions very well.

I then turned to the Z-248 at home, and planned the same type of installation, but I quickly realized I wanted to have my cake and eat it, too. I did not wish to sacrifice my only 360K drive, as I frequently trade public domain programs with friends. I did not wish to give up my 1.2 meg drive, since it was so awfully convenient for backup disks, as well as quickly running a Winchester backup program. Where would I put the 3.5 inch drive, physically, as well as how could it be installed as a functioning part of my system?

I considered a new type of floppy controller, which will support 4 floppy drives, but was unclear as to whether or not drivers would be required to support the third drive, since Zenith DOS definitely supports only 2 drives in the system at a time. In addition, the machine has a floppy/Winchester controller, and installing a new floppy controller would mean either disabling the floppy portion of the board, which cannot be done in most cases, or purchasing a Winchester only controller to go with the new floppy board — a considerable expense for a project designed for convenience only.

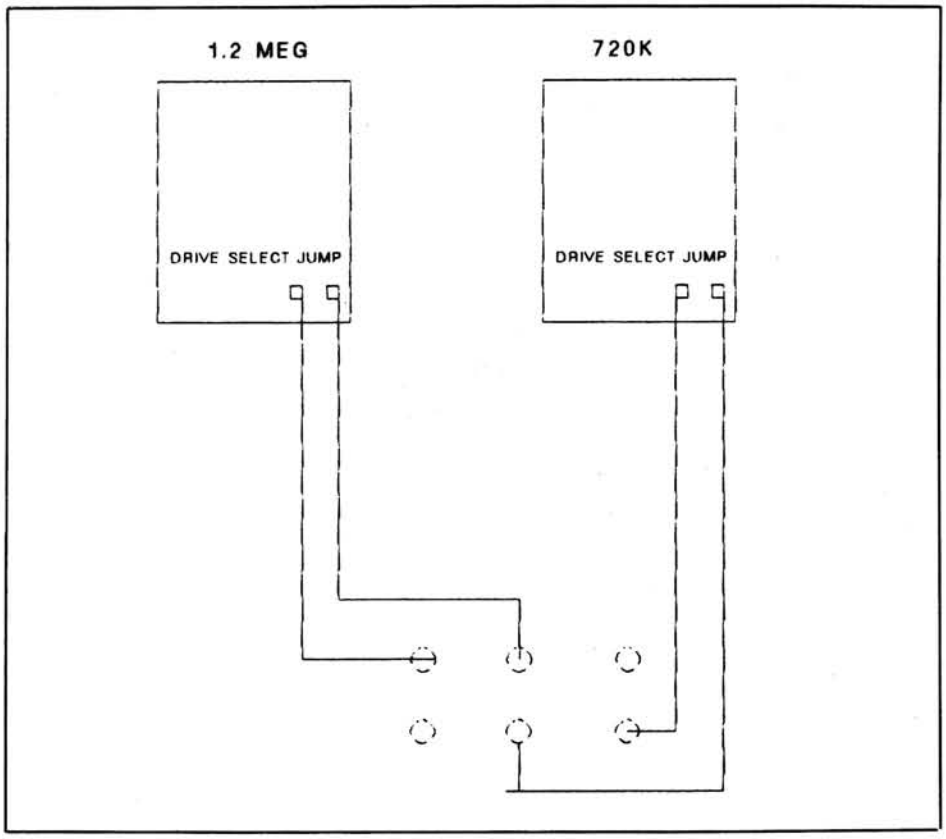
I experimented with many options, only to come to the conclusion that the Z-248 was designed to handle 2 and only two drives, and for me to try and outsmart the designers of this machine was akin to fooling with Mother Nature. I mentioned my predicament to Mark Ruthenbeck — gallant Engineer and friend, and he suggested a way to install and utilize 3 drives in the Z-248, although only 2 may function at any given time, all 2 are available, and very readily accessible.

Mark advised me that a third 34-pin card edge connector could be installed on the floppy cable, and permanently installed on the third drive, without disruption to the system, as the presence or absence of any drive was a function of its drive select jumper. With the jumper removed, that drive is nonexistent to the system, and can remain there indefinitely without distracting DOS.

I now had the ability to have all three types of media available to me in my Z-248. I began by doing the physical installation. I first ran 'ship' to force the heads on the Winchester to move to the park and safe position, since I knew I would be jarring the drives. Do not forget to do this. I then removed one of the drive 'cages' which contained a floppy drive, and at the bottom, a Winchester drive. There is space in this cage for a third drive, inbetween the top and bottom drives, and insertion was a matter of drilling 4 small holes for mounting the drive, and fitting it in. This was easily accomplished.

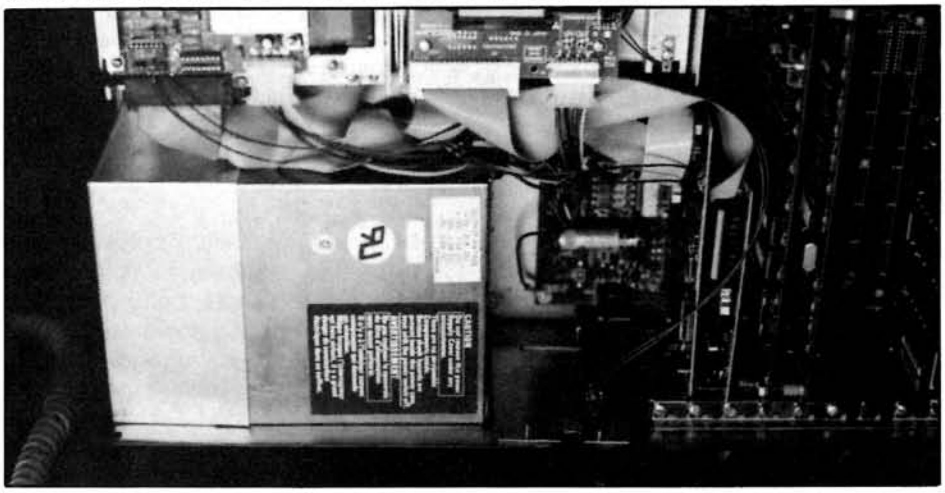
It then was necessary to take a hack saw, and cut the face metal panel to allow the lower drive to protrude. This also was accomplished without any complication, although extreme care must be taken due to vibration and metal shavings being scattered about.

This now leaves a third drive protruding out of the front panel, and a plastic faceplate with provision for only 2 drives. Obviously, something has to give. Bear in mind that you have 2 options in this regard. 1) You may cut the existing faceplate, and make provision for the drive, or 2) since Zenith makes a 4 drive faceplate for their '386 machines, you may opt to acquire that particular faceplate. The latter is the approach I selected.



I then installed the third card edge connector, and a 'Y' connector on the power cable to bring power to the drive. When installing the card edge connector, be very certain your connector is properly positioned on the cable. There are 'tracks' on the connector that allow you to position it properly, and be certain also, to seat the top portion of the connector, so it does not come loose.

With the aforementioned accomplished, all that remained was to configure the jumpers on the drives, and the hardware portion of this adventure would be complete. I decided to install a toggle switch on the rear of the computer. There is a removable plate on the rear panel, next to the power supply, which made the drilling, etc., quite easy. I then removed the jumpers from the 1.2 meg drive, and the 720K drive, which were going to take turns acting as drive 'B', and carefully inserted spring brass female pins taken from a 'D' shell connector. The jumper on the 720K drive is very small, and not terribly accessible, but after some effort, they stayed put. I ran the pair of wires from each drive to the toggle switch, and installed them. Bear in mind that the toggle switch must be a DPDT (double-pole, double-throw), as shown in the drawing, as you cannot mix the wires from different drives in any way.



Now that the hardware portion is accomplished, there remains only the task of telling DOS which drive is installed. Zenith makes this very easy with a file called 'DSKSETUP'. I do not believe this program is available with IBM's version of DOS (another reason we are such devoted Heath/Zenith fans). Call up DSKSETUP and follow the instructions. Once this is done, you will be able to run the selected drive at will. Setting up this procedure on a keyboard "tape recorder", available in the public domain for a nominal cost, can reduce the effort to accomplish this to a single keystroke.

Several cautions must be observed.

1. On a Z-150 series machine (XT), as stated, the 720K drive installs as it comes out of the box.

2. On a Z-200 series computer (AT) pin 34 of the 720K drive must be cut (or taped under the connector). It cannot be in the circuit. You will get an error message if it is. This applies only to the 720K drive. It does not apply to either of the 5.25 inch drives.

3. Drive 'A' in either an 'XT' or 'AT' environment is at the end of the cable. This drive must have a 'twist' in the cable prior to it. This is the way IBM accomplished drive select between drives 'A' and 'B'. There is no getting around it. It is one of those things you must live with.

4. The drive select jumpers on all drives are in the same position. The only thing we are doing relative to the third drive installation is to select one of the 2 drives to be drive 'B'. Drive 'A' remains as originally installed. The selection of drive 'B' types (720K or 1.2 meg) is done by means of the toggle switch only.

Obvioulsy, this technique is limited only by your imagination. No doubt 3.5 inch floppies are here to stay, some feel they will eventually obsolete 5.25 inch, due to their increased capacity and ease of handling and carrying about. Regardless of how you configure your system, now or in the future, additional drives will certainly enhance its usefulness and convenience.

*

Continued from Page 58

talking about some of the ideas that were originally presented in the last two REMark issues (November and December) in 1983. Those ideas will, of course, be updated to reflect the current state-of-the-art, but in looking back through those articles; I note that my December 1983 article was called: "Trade Your H/Z-89 for an H/Z-100?" Yes, I really did start using Heath computers with the old H-89, and that article discusses why I found it appropriate to get a '100.

If you have any questions about anything in this column, or about Heath/Zenith systems, in general, be sure to include a self-addressed, stamped envelope (business size preferred) if you would like a personal reply to your question, suggestion or comment.

Items Discussed

Hardware

HWD-20 (20 MB hard disk w/controller) $449.95
HWD-20-AT (20 MB hard disk w/out controller) $379.95
Heath/Zenith Computer Centers
Heath Company Parts Department
Hilltop Road
St. Joseph, MI 49085
(800) 253-7057
(Heath Catalog orders only)

Software

WordStar Professional $495.00
MicroPro International
Attn: Customer Service
P.O. Box 7079
San Rafael, CA 94901-7079
(800) 227-5609 (Orders only)

*

Revisiting CRTSaver Revisited

William T. Vomocil
1143 Brahma Lane
Yuma, AZ 85364

I was very pleased with Robert S. Brasfield's article, "CRTSAVER Revisited", in the March '88 REMark, because he provided a way to set the inactivity time of the original CRTSAVER utility published in Frank T. Clark's article, "Advanced Assembly Language Programming", in the July '84 issue of REMark.

I also use the CRTSAVER program routinely, from a batch file, actually in my AUTOEXEC.BAT file. My experience has been much the same as Mr. Brasfield's, in that the original ten (plus) minutes inactivity time was excessive for most of my uses. Thus, I was greatly gratified to have a program that allowed me to adjust the inactivity time to suit my personal desires, and a terminate-and-stay-resident (TSR) program that minimized the amount of wasted memory. Thank you, Mr. Brasfield!

I copied Mr. Brasfield's program, assembled it, and it functioned perfectly. It wasn't long, however, before I ran into a situation where I wanted to reset the inactivity time. To do so required that I warm-boot my Z-100 so that the program was removed from memory and could therefore be reinstalled with a new delay time. This seemed singularly cumbersome and time- consuming. So, I attempted to do what Mr. Brasfield suggested at the end of his article - write a TSR program with a special section of code that would evict its own resident code if the program was run a second time with a command line switch that activated the eviction routine. The code I wrote didn't turn out precisely that way, but it's close. What follows is an explanation of my revision of Brasfield's enhancement to Clark's original code.

I began by using Mr. Brasfield's program, making several changes designed to recover and save a few parameters when the program is re-called after already having been installed in memory. These parameters are required to later remove or reset the resident program. Then I wrote two additional sections of code, one to remove the program from memory, and the other to allow the user to reset the inactivity time of the already-installed routine. Finally, I added a menu to indicate the options available to the user, and some error-trapping code to display error messages if the user inadverdently made an improper keyboard input or if other errors occurred.

When installed for the first time, the program runs exactly as it did before. If no inactivity time is specified, it will automatically set the delay time to three minutes. if a number of seconds in the range 20 to 600 is entered as part of the command line, the inactivity time will be set accordingly. Also, as before, only a space or a "/" is allowed as the first character following the filename on keyboard entry.

If the user wishes to change the existing memory-resident program, all that is required is to type the filename "CRTSAVER" from the keyboard, this time without any characters following it. If the CRTSAVER routine is already installed, and the user enters the filename with following characters, the program will ignore them. Of course, the CRTSAVER.COM program must reside on the floppy disk or hard disk partition in use, or be available through an active path when in a different partition. When the filename is entered, the program will respond by first clearing the screen, then displaying the menu of options available to the user.

The menu will look like this:

```
Resident copy of CRTSAVER already exists.

Do you want to:

(1) Return to MS-DOS without changes?
```

```
(2) Remove the memory-resident CRTSAVER
   routines?

(3) Reset the CRTSAVER delay time? (20 to
   600 seconds)

              Enter number desired:
```

If the user selects either (1) or (2), the program will execute immediately. If (3) is selected, the program will respond with the following statement:

```
Enter delay time desired:
```

When a number in the range 20 to 600 is entered at the prompt, the program will reset the inactivity time of the existing memory-resident routines to the number input. The amount of memory used by the TSR program will not be changed. This can be verified by running CHKDSK before and after the change. After execution, the program will display the following statement:

```
CRTSAVER routines reinstalled
```

If an improper input occurs, e.g. "18", "610", or "/", the program will respond with the statement shown below:

```
Command Line Format Error - Aborting!
```

And, it will immediately return control to MS-DOS. If no number is entered before pressing the <RETURN> key, the default setting of three minutes inactivity time will be set.

Next is a general discussion of the changes I made to Mr. Brasfield's code. After that, I will provide some explanation of the additional code that I wrote. The complete source code for the modified program is included at the end of this article.

The first change to Mr. Brasfield's code is the addition of three equate statements for a carriage return, linefeed and an escape at the beginning of the program. These are just a matter of preference to the programmer. I find it makes it a little easier for me to define common constants up front. Next, the line ORG 80H was added to define a buffer for keyboard input, followed by a byte to store the length of the input string. After the NXT_TM definition, I added a doubleword to store the contents of the original DI register, called OLD_DI. It is needed to remove the program from memory. Two other doublewords were added, OLD_CS and OLD_DT, to store the contents of the CS register and the offset to the original delay time, respectively. Only one of these parameters, (OLD_DI) will become part of the memory-resident program. The others do not, and are therefore positioned in the part of the program that is discarded after the routine is installed. Next is a line of code following the line ENTRY:, defining the message CRTSAVER ROUTINE. This does not directly contribute to the program, but it does uniquely identify the memory-resident program and greatly increases the effectiveness of the compare process when searching for a TSR program if a number of memory-resident routines all begin with the same sequence of instructions. The next changes are additional error, clear-screen and menu messages following the line DELAY EQU WORD PTR ON + 5. Next, immediately after the line INSTALL:, the offset of the delay time of the original program is saved in the word OLD_DT. Additionally, in place of the line INT 20H return to DOS, I used a near procedure named EXIT to provide a new line on the screen as well as a graceful exit to the operating system.

No more changes were made until the line following JE INS080. The next line, JMP INS200, directs the program execution to the menu as soon as the resident CRTSAVER routines are found in memory. Following the line INS130:, a new line of code saves the value of the original CS register in the word OLD_CS. Next, following the line OR AL,AL, the JZ INS180 was changed to JZ INS190, and following the line CMP AL,'/', the JNE INS180 was changed to JNE INS190. Likewise, following the line CMP AL,30H, the JB INS180 was changed to JB INS170; following the line CMP AL,39H, the JA INS180 was changed to JA INS170; following the line OR DX,DX, the JNZ INS180 was changed to JNZ INS170; following CMP BX,258H, the JA INS180 was changed to JA INS170; and following CMP BX,14H, the JB INS180 was changed to JB INS170. All these changes were included in order to show an error message if improper keyboard input occurs. The two lines of code following the former INS170: were replaced with the necessary code to display the error message, and these two lines now follow INS180:. The original INS180: was replaced by INS190:. Finally, a new line was added after the first CALL SETVEC. This code, MOV OLD_DI,DI, saves the original value of DI for use later.

The code I added from INS200: to the SET_VEC procedure merely displays the menu, obtains keyboard input from the user, and displays appropriate messages on the screen. The first three lines following INS200: clear the screen. The next three lines display the message that the CRTSAVER routines are already resident in memory. The following three lines display the menu of options available to the user. The next two lines wait for keyboard input, and the following seven lines, beginning with CMP AL,'1', and ending with RET, compare the keyboard input to "1", "2", or "3", and direct execution of the program to the appropriate location in the code based on this input. The seven lines of code following INS210:, beginning with MOV DX,OFFSET CRLF, and ending with JMP QUIT, skip a line on the screen and then print an error message if the keyboard input was not a "1", "2", or "3", and then direct the program execution to a line of code that calls the procedure which returns control to the operating system. The code following the line REMOVE: calls the section of the routine that evicts the memory-resident program, displays the message that the program has been removed, and jumps to the line of code that calls the section of the program that resets the inactivity time of the already resident routines, and displays the message that the program has been reinstalled. The code following the line QUIT: calls the section of the routine that returns control to the operating system.

The code that actually removes the routine from memory is contained in the near procedure called EVICT. The Get Interrupt Vector (Function 35H) is used to obtain the original values of ES:BX. Using ES as the original code segment address, the DI register is loaded with the original value of DI, (OLD_DI), and using this as a pointer, the original vector offset and segment are moved into the DX and CX registers respectively. Then, using the Set Interrupt Vector (Function 25H), the vector is restored to its original value. All three vectors are restored in an identical manner. After this is done, the Free Allocated Memory (Function 49H), is used to free the main memory block. The environment block was previously freed in Mr. Brasfield's original code. The lines of code following MOV AX,ES free the allocated memory or display an error message to the user if the operation is unsuccessful. The first four lines of code free the allocated memory by using the Free Allocated Memory (Function 49H), and continue on with the program if the memory is successfully freed. The remaining lines of code display an error message to the user if the attempt to free the memory was unsuccessful for any reason.

The code contained in the near procedure RESET changes the inactivity time of the memory-resident TSR program. The code from the beginning of this procedure to the line XOR DX,DX, merely obtains the user keyboard input. The code following this line to the line MOV ES,OLD__CS, is very similar to the original code. It limits the acceptable character input immediately following the menu keyboard input request to a space or a slash; it obtains the inactive time number from the buffer containing the input string and allows only ASCII numbers 0 through 9 or else uses the default inactive time; and it requires the time to be at least 20 seconds, but not more than 600 seconds. The seven lines of code following the line INS240: skip a line on the screen and display an error message if an improper keyboard input is made. The interesting part of this section of code is the three lines which change the inactivity time of the existing TSR routine. These lines, i.e. MOV ES,OLD__CS, MOV BX,OLD__DT, and MOV ES:WORD PTR [BX],AX, function as follows: The delay time, in of the original code segment is loaded into the ES register, (MOV ES,OLD__CS). The offset to the original delay time is then loaded into the BX register, (MOV BX,OLD__DT). And finally, the number for delay time contained in the AX register is stuffed into the existing memory-resident routine using ES as the segment address and BX as the offset (MOV ES:WORD PTR [BX],AX). Similarly, the code immediately following the line INS260: inserts the three-minute delay time in the existing program if no inactivity time is specified by the user.

The final few lines of code are contained in the near procedure EXIT. This procedure displays a carriage return and a linefeed (skips a line) on the screen and then exits the CRTSAVER program and returns control to the operating system. Skipping the line is purely for aesthetic reasons. I personally think it makes the screen display look better.

Since I am a neophyte in the area of assembly language programming, I am absolutely certain there are more efficient and effective ways to accomplish what I have done here. I'll leave that for some of you really smart guys and gals out there who are REMark fans like I am. I had a great deal of fun and frustration getting this program to work, but it was an educational experience for me. I hope it is useful to some of you.

```
Title -- CRTSAVER with removal amd timing reset features

cr      equu    ØDh                              ;Carriage return
lf      equ     ØAh                              ;Linefeed
esc     equ     1Bh                              ;Escape
;*******************************************************************
crtsaver        segment
        assume  cs:crtsaver,ds:crtsaver,es:crtsaver,ss:crtsaver

        org     8Øh
cmd_len db      ?                                ;Command line string length
cmd_buf dw     ?                                ;Command line string storage
        org    . 1ØØh

begin:
        jmp     install                          ;Skip parameter storage

crt_off         db      Ø
timer           dw      ?
nxt_crt         dd      ?
nxt_kb          dd      ?
nxt_tm          dd      ?
old_di          dw      ?

entry:
        jmp     int_kb                           ;Skip ''identification
                db      CRTSAVER ROUTINE         ;Subroutine identification

int_kb:
        call    testit
        jmp     dword ptr cs:nxt_kb
int_crt:
        call    testit
        jmp     dword ptr cs:nxt_crt
;------------------------------------------------------------------
testit  proc    near
        test    cs:crt_off,-1
        je      on
        push    ax
        in      al,ØD8h
        and     al,ØF8h
        or      al,Ø8h
        out     ØD8h,al
        mov     cs:crt_off,Ø
        pop     ax

on:
        mov     cs:timer,465Øh                   ;3 minute delay
        ret
testit  endp
;------------------------------------------------------------------
int_tm:
        sub     cs:timer,ax
        jc      off_crt
        jmp     dword ptr cs:nxt_tm
off_crt:
        test    cs:crt_off,-1
        jne     off
        push    ax
        in      al,ØD8h
        and     al,ØF7h
        or      al,Ø7h
        out     ØD8h,al
        mov     cs:crt_off,-1
        pop     ax
off:
        jmp     dword ptr cs:nxt_tm

this_site       label   near                    ;Or, LABEL BYTE, but
span    equ     this_site - begin               ;must have an attribute

if      span mod 16
        org     (this_site + 16) - (span mod 16)
endif

env_seg equ     word ptr begin - ØD4h           ;Set up symbolic names
delay   equ     word ptr on + 5                 ;for these memory items

insmsg  db      cr,lf, CRTSAVER routines installed. ,cr,lf, $
rinsmsg db      cr,lf,lf, CRTSAVER routines reinstalled. $
notmsg  db      cr,lf, CRTSAVER not installed. ,cr,lf, $
dosmsg  db      cr,lf, Wrong version of MS-DOS. ,cr,lf, $
eximsg  db      cr,lf, Resident copy of CRTSAVER already exists. ,cr,lf, $
memmsg  db      cr,lf, Memory allocation is in error. ,cr,lf, $
faimsg  db      cr,lf, This copy of CRTSAVER may be defective. ,cr,lf, $
bad_cmd db      cr,lf, Command line format error - aborting! $
fail    db      cr,lf, Failed to free allocated memory - aborting! ,cr,lf, $
crlf    db      cr,lf, $
new_dly db      cr,lf,lf, Enter delay time desired: $
remmsg  db      cr,lf,lf, CRTSAVER routines removed from memory. $
cls     db      esc, E , $
menu    db      cr,lf, Do you want to: ,cr,lf,lf,lf,
        db              (1)  Return to MS-DOS without changes? ,cr,lf,lf,
        db              (2)  Remove the memory-resident CRTSAVER routines? ,
        db      cr,lf,lf,
        db              (3)  Reset the CRTSAVER delay time? (2Ø to 6ØØ seconds)
        db      cr,lf,lf,lf',
        db                                  Enter number desired: $

mult1   dw      ØAh
mult2   dw      Ø64h

old_cs  dw      ?
old_dt  dw      ?
old_cpy db      Ø
new_cpy db      Ø

install:

        mov     old_dt,offset delay             ;Save address of delay time

;Determine MS-DOS version number. Must be 2.Ø or over.

        mov     ah,3Øh
        int     21h
        cmp     al,Ø2h
        jae     insØ2Ø
```

```
        mov     dx,offset notmsg
        mov     ah,Ø9h
        int     21h
        mov     dx,offset dosmsg                ;If not 2 or over
        mov     ah,Ø9h
        int     21h
        call    exit                            ;For Z-DOS termination
insØ1Ø:
        push    dx
        mov     dx,offset notmsg                ;Show common message
        mov     ah,Ø9h                          ;first,
        int     21h
        pop     dx                              ;then specific message
        mov     ah,Ø9h
        int     21h
        call    exit                            ;Return to MS-DOS

;Set ES to segment of environment block, release it.

insØ2Ø:
        mov     es,env_seg
        mov     ah,49h
        int     21h

;Determine if CRTSAVER is already resident in memory. Do not
;retain this copy in memory if so. Locate any existing copy
;by locating DOS s chained memory control blocks and working
;our way up the chain.

        mov     dx,word ptr ds:[2]              ;Top of memory, from PSP
        xor     bx,bx
insØ3Ø:
        cmp     bx,dx                           ;Are we at the top?
        jb      insØ4Ø
        mov     dx,offset faimsg                ;Should never be at top
        jmp     insØ1Ø                          ;while in this loop
insØ4Ø:
        mov     es,bx
        cmp     byte ptr es:[Ø],4Dh             ;Marks start of a block
        jne     insØ5Ø

;When the first genuine memory control block is found, it will
;be validated by the fact that, in two successive calls to the
;procedure REACH, the value of ES, plus one, plus the word at
;ES:[3] points to another paragraph whose first byte is either
;Ø4Dh or Ø5Ah. A Ø4Dh character may exist as the first in a
;paragraph just as a random event, in executable code. Ø4Dh is
;the instruction to ''DEC BP  , or could be a data byte.

        call    reach
        cmp     cl,4Dh                          ;If not Ø4dh, pass over
        jne     insØ5Ø                          ;paragraph held in BX
        call    reach
        cmp     cl,4Dh                          ;If second test succeeds,
        je      insØ6Ø                          ;break out of this loop
        cmp     cl,5Ah
        je      insØ6Ø
insØ5Ø:
        inc     bx
        jmp     insØ3Ø
insØ6Ø:
        mov     es,bx

;When we arrive at INSØ7Ø, we are presumably into the chain
;of memory control blocks. This continues to be verifiable by
;success in using one element of the chain to locate another.

insØ7Ø:
        call    reach                           ;For next in the chain,
        cmp     ax,dx                           ;AX has ES value
        jb      ins1ØØ

;When at the top of memory, decide what to do based on flags
;we set on the way.

        cmp     old_cpy,Ø                       ;If old copy found,
        je      insØ8Ø
        jmp     ins2ØØ                          ;get user input from menu
insØ8Ø:
        cmp     new_cpy,Ø                       ;If this copy verified,
        jne     ins14Ø                          ;go to next task
insØ9Ø:
        mov     dx,offset memmsg                ;Else show error message
        jmp     insØ1Ø                          ;and bail out

;If CL has Ø4Dh or Ø5Ah, we should perform a signature check
;except that if the block is our code segment, it is not
;needed. If not one of those, we have an error situation.

ins1ØØ:
        cmp      cl,4Dh
        je      ins11Ø
        cmp     cl,5Ah
        jne     insØ9Ø
ins11Ø:
        mov     ax,word ptr es:[1]
        mov     cx,cs
        cmp     ax,cx
        jne     ins12Ø
        inc     new_cpy
        jmp     insØ7Ø
ins12Ø:
        push    es
        mov     es,ax
        mov     si,offset int_kb                ;Compare this part of
        mov     di,si                           ;code area as signature
        mov     cx,2Øh                          ;Compare 32 bytes
        cld
        repe    cmpsb
        pop     es
        jcxz    ins13Ø                          ;If both areas same
        jmp     insØ7Ø                          ;else continue search
ins13Ø:
        mov     old_cs,ax                       ;AX has old CS, save for
                                                ;reset subroutine

        inc     old_cpy                         ;Set ''in place   flag,
        jmp     insØ7Ø                          ;and continue search
```

```
;Get user s input from command line for inactive time.

ins14Ø:
        xor     dx,dx
        mov     cx,dx
        mov     bx,dx
        mov     si,8Øh
        lodsb
        or      al,al                           ;If nothing there, skip
        jz      ins19Ø                          ;this number input
        lodsb                                   ;Get delimiter

;Permit only space or slash as first character after the filename.

        cmp     al,2Øh
        je      ins15Ø
        cmp     al, /
        jne     ins17Ø

;The INS15Ø loop acquires the inactive time number from the
;command line. Only ASCII characters Ø through 9 are allowed,
;else the default inactive time will be used.

ins15Ø:
        lodsb
        cmp     al,cr                           ;Ødh = all done
        je      ins16Ø
        cmp     al,3Øh
        jb      ins17Ø
        cmp     al,39h
        ja      ins17Ø
        and     al,ØØØØ1111b                    ;Strip off ASCII part
        mov     cl,a                           ;Hold till BX multiplied
        mov     ax,bx                           ;BX has prior result
        mul     mult1                           ;Multiply it by 1Ø
        add     ax,cx                           ;Add this new digit
        mov     bx,ax                           ;Put new result in BX
        jmp     ins15Ø                          ;Do until carriage return
                                                ;found

;Require that the time be at least 2Ø seconds, but not more than 6ØØ.
;Otherwise use the existing Ø465Øh value for delay.

ins16Ø:
        or      dx,dx                           ;DX must remain zero,
        jnz     ins17Ø                          ;else delay is too large
        cmp     bx,258h
        ja      ins17Ø
        cmp     bx,14h
        jb      ins17Ø
        jmp     ins18Ø
ins17Ø:
        mov     dx,offset bad_cmd
        mov     ah,Ø9h
        int     21h
        call    exit
ins18Ø:
        mov     ax,bx
        mul     mult2                           ;Make .Ø1 second units
        mov     delay,ax                        ;Move to executable code
ins19Ø:
        mov     al,55h                          ;Install vectors
        mov     dx,offset int_crt               ;Our interrupt handler
        mov     di,offset nxt_crt               ;Where exit address goes
        call    set_vec
        mov     old_di,di                       ;Save old DI for evict
                                                ;subroutine
        mov     al,5Øh
        mov     dx,offset  int_kb
        mov     di,offset nxt_kb
        call    set_vec
        mov     al,51h
        mov     dx,offset int_tm
        mov     di,offset nxt_tm
        call    set_vec
        mov     dx,offset insmsg
        mov     ah,Ø9h
        int     21h

;DX has size of the stay-resident area. Convert to paragraphs.

        mov     dx,offset insmsg
        mov     cl,Ø4h
        shr     dx,cl
        mov     ah,31h                          ;Keep process function
        int     21h                             ;End of this phase

;Display the options menu to the user

ins2ØØ:
        mov     dx,offset cls                   ;Clear screen
        mov     ah,Ø9h
        int     21h
        mov     dx,offset eximsg                ;Display CRTSAVER exists
        mov     ah,Ø9h                          ;message
        int     21h
        mov     dx,offset menu                  ;Display menu
        mov     ah,Ø9h
        int     21h
        mov     ah,Ø1h                          ;Get keyboard input
        int     21h

;Display error messages if improper input

        cmp     al, 1
        jb      ins21Ø                          ;Show error message if < 1
        jz      quit                            ;Return to DOS
        cmp     al, 2
        jz      remove                           ;Go to remove subroutine
        cmp     al, 3
        jz      rset                            ;Go to reset subroutine

ins21Ø:
        mov     dx,offset crlf                  ;Skip a line
        mov     ah,Ø9h
        int     21h
        mov     dx,offset bad_cmd               ;Display error message
        mov     ah,Ø9h                          ;if > 3
```

```
        int     21h
        jmp     quit                            ;Return to DOS

;Jump to the subroutines selected by the user

remove:
        call    evict                           ;Get subroutine
        mov     dx,offset remmsg                ;Display removed message
        mov     ah,Ø9h
        int     21h
        jmp     quit                            ;Return to DOS
rset:
        call    reset                           ;Get subroutine
        mov     dx,offset rinsmsg               ;Display reset message
        mov     ah,Ø9h
        int     21h
quit:
        call    exit                            ;Get out of here
;-------------------------------------------------------------------
set_vec proc    near
        push    ax                              ;Save interrupt number

;Get exit address in place before we direct the interrupt here.

        mov     ah,35h                          ;Function to get prior
        int     21h                             ;interrupt vector
        mov     [di],bx                         ;Store its offset
        mov     [di+2],es                       ;and its segment
        pop     ax                              ;Recover AL
        mov     ah,25h                          ;Function places vector
        int     21h                             ;to our code
        ret
set_vec endp
;-------------------------------------------------------------------
reach   proc    near
        mov     ax,es
        add     ax,word ptr es:[3]
        inc     ax
        mov     es,ax
        mov     cl,byte ptr es:[Ø]
        ret
reach   endp
;-------------------------------------------------------------------

;Reset the vectors to their original values

evict   proc    near
        mov     al,55h                          ;CRT interrupt number
        mov     ah,35h                         ;Get interrupt vector
        int     21h
        mov     di,es:old_di                    ;Get old DI
        mov     dx,es:[di]                      ;Get old vector IP
        mov     cx,es:[di+2]                    ;Get old vector CS
        push     ds                              ;Save current DS
        mov     ds,cx                           ;Set vector destination
        mov     ah,25h                          ;Set vector address
        int     21h
        mov     al,5Øh                          ;Keyboard interrupt number
        mov     ah,35h                          ;Get interrupt vector
        int     21h
        mov     ah,25h                          ;Set vector address
        int     21h
        mov     al,51h                          ;Timer interrupt number
        mov     ah,35h                          ;Get interrupt vector
        int     21h
        mov     ah,25h                          ;Set address
        int     21h
        pop     ds                              ;Restore DS

;Remove main memory resident routine memory block

        mov     ax,es                           ;Address of main block
        mov     ah,49h                          ;Free allocated memory

        int     21h
        jnc     free_ok

;Tell the user that there was a problem

        push    ax
        mov     dx,offset fail                  ;Display fail message
        mov     ah,09h
        int     21h
        pop     ax
        call    exit
free_ok:
        ret
evict   endp
;-------------------------------------------------------------------

;Ask the user for delay desired, and show error message if improper input

reset   proc    near
        mov     dx,offset new_dly               ;Ask user for new delay time
        mov     ah,09h
        int     21h
        mov     [cmd_buf],0
        mov     byte ptr [cmd_len],80
        mov     dx,offset cmd_len
        mov     ah,0Ah                          ;Get keyboard input
        int     21h
        xor     dx,dx
        mov     cx,dx
        mov     bx,dx
        mov     si,81h                          ;Skip the string length
        lodsb
        or      al,al                           ;If nothing there, skip
        jz      ins260                          ;this number input
        cmp     al,20h
        je      ins220
        cmp     al, /
        je      ins240
ins220:
        lodsb
        cmp     al,cr                           ;Carriage return = all done
        je      ins230
        cmp     al,20h                          ;Ignore space
        je      ins220
```

```
        cmp     al,2Fh                          ;Ignore slash
        je      ins220
        cmp     al,30h                          ;Can t be less than zero
        jb      ins240
        cmp     al,39h                          ;Can t be greater than nine
        ja      ins240
        and     al,00001111b
        mov     cl,al
        mov     ax,bx
        mul     mult1
        add     ax,cx
        mov     bx,ax
        jmp     ins220
ins230:
        or      dx,dx
        jnz     ins240
        cmp     bx,0258h
        ja      ins240
        cmp     bx,14h
        jb      ins240
        jmp     ins250
ins240:
        mov     dx,offset crlf                  ;Skip a line
        mov     ah,09h
        int     21h
        mov     dx,offset bad_cmd               ;Display error message
        mov     ah,09h
        int     21h
        call    exit

;Plug the new time delay into the original memory-resident code

ins250:
        mov     ax,bx
        mul     mult2                           ;Delay in .01 second units
        mov     es,old_cs                       ;Get old CS
        mov     bx,old_dt                       ;Get old offset
        mov     es:word ptr[bx],ax              ;Move to executable;code
        ret

;Make the delay time 04650h (3 minutes) if no user input given

ins260:
        mov     ax,4650h                        ;Delay time (3 minutes)
        mov     es,old_cs                       ;Get old CS
        mov     bx,old_dt                       ;Get old offset
        mov     es:word ptr[bx],ax              ;Move to executable code
        ret
reset   endp
;-------------------------------------------------------------------
exit    proc    near
        mov     dx,offset crlf                  ;Skip a line
        mov     ah,09h
        int     21h
        mov     ah,4Ch                          ;Terminate process
        int     21h
exit    endp
;-------------------------------------------------------------------
crtsaver        ends
;*******************************************************************
        end     begin
```

Gregory D. Elder
Qtrs 4301-A
USAF Academy, CO 80840

GETTING STARTED WITH....

BBS's

Anybody with an interest in computers has probably seen the acronym BBS, which stands for Bulletin Board System. Many computer users, however, do not know what a BBS is or what a BBS can be used for.

In this article, I will provide some basic information about BBSs. I will describe the purpose of a BBS, how to access a BBS, and what benefits you can derive from using a BBS. This article will not be a detailed guide to every type of BBS. On the other hand, it should give you enough information to get started with a BBS.

What is a BBS?

As mentioned above, a BBS is a computer bulletin board system. Specifically, it is a computer running a program which allows other computers to connect to the BBS to exchange information. Think of a BBS as an information bank — a place where you can deposit and withdraw information.

Almost every BBS has a message area. This is where BBS users can post messages for viewing by other users. This is similar to the old cork bulletin board found on many office walls. People in the office tack paper notes to the board for everyone else to read. (You can see how the term computer bulletin board system originated.)

Why Use a BBS?

The best reason for using a BBS is to have access to the wealth of information that is available. This information takes two forms — messages posted by users (comments, thoughts, ideas, etc.), and software which you can download to your computer.

Most BBSs are organized for specific interests. For example, certain BBSs are concerned only with IBM-PCs, others may be for Z-100 users, still other BBSs may be for people interested in the stock market, movies, science fiction, or arms control. No matter what your interest, there is probably a BBS for you. (HUG even provides a BBS for Heath/Zenith users. See advertisements in current issues of REMark.)

What Equipment is Needed?

Besides your computer, two other items are required before you can access a BBS: a modem and a communications program. A modem is a device which converts the digital signals of your computer to an analog signal which can be transmitted across a telephone

line. (By the way, modem is short for MOdulator/DEModulator. The sending modem modulates the computer signal while the receiving modem demodulates the signal.)

A cable from the modem connects to a serial communications port on your computer. In addition, a telephone line from the modem plugs into a modular phone outlet. By the way, older modems may have accoustic couplers instead of a modular phone jack. An accoustic coupler is a pair of round, rubber connectors in which the handset of the telephone plugs into.

One important consideration when selecting a modem will be the transmission speed--how fast the modem can send and receive data. Most BBS's communicate at 300 or 1200 bits per second (bps). Many even operate at 2400 bps. Of course, the faster the modem, the more expensive the cost.

You will need communications software to make your computer "talk" to the modem and BBS. While numerous commercial comm packages exist (ZSTEM, Cross-Talk, HyperAccess, etc.), a good number of public domain and shareware programs are available which do just as good a job. For example, I use KERMIT or BESTERM on my Z100. For PC-type machines, you can use KERMIT, PROCOMM, or PC-TALK, just to name a few. *(or HUGMCP - Ed)*

If you plan to download software from BBS's, ensure your comm program has a file transfer capability. In addition, I would recommend software which supports the Xmodem file transfer protocol. (Xmodem provides for error checking during file transfers to ensure your files arrive correctly.) There are a number of other error-checking protocols (KERMIT, Ymodem, SEALink), but not all BBS's support them. Almost all BBS's which allow file transfers do support Xmodem, however.

What BBS to Call?

You have your modem and comm program, and are now ready to call a BBS. Where do you find a telephone number for a BBS? If you belong to a computer users group, other members of the group may be familiar with some BBS's. Many computer clubs even operate BBS's for their members. Another place to look is in some computer publications. For example, Computer Shopper has a section listing BBS numbers. Also, the January issue of REMark lists current HUG clubs and indicates BBS numbers for some of those clubs. Finally, once you access a BBS, you may find a file on the BBS listing other BBS phone numbers.

Logging In

You finally have everything you need to access a BBS. After dialing the phone number, what happens? Well, once the BBS answers your computer, you will probably see some kind of welcome notice, followed by a prompt for your name. Figure 1 shows what you see after connecting to the HUG BBS. After you type in your name, the BBS will search its users data base to see if you are a registered user. If you are a registered user (have used the BBS before), the BBS will then ask for your password.

If you are a new user, you will probably have to enter your city, state, and telephone number. Some unscrupulous people like to use false names on BBS's and leave obscene messages. By leaving your phone number, the SYSOP (system operator of the BBS), can call to verify who you are. In fact, some BBS's won't allow you to use the system until the SYSOP has called to validate your log-in information. (More on unscrupulous people later.)

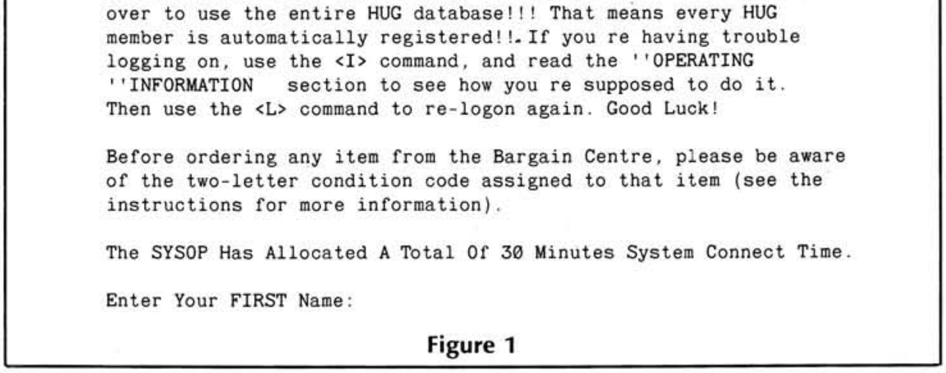
The last piece of log-in information that may be required is a password of your choosing. This will prevent others from logging into the BBS with your name (unless they know your password.) Normally, the password will not echo on the screen. This is a safety measure in case someone happened to be looking over your shoulder at the computer screen. In all likelihood, you will be asked to enter the password a second time to verify its correct spelling. (Remember your password! You won't be able to log-in to the BBS again without it.)

After you have successfully logged-in, you may see a message which says you are limited to a certain amount of time on the BBS (usually around 30-60 minutes). Most BBS's only have one phone line which means only one person at a time can use the BBS. SYSOPs set time limits so one person can't "hog" the system for himself.

Once in, What Do I Do?

Most bulletin boards provide a menu of commands. For example, the HUG BBS uses the menu shown in Figure 2. As a first time user, you may want to take a look at each command to see how they work. You normally just have to type the first one or two characters of a word from the menu to execute that command.

As I mentioned before, probably all BBS's have a message section where users can post messages. People may leave notes about hardware/software for sale or that they want to buy; people may ask questions concerning a problem they are having with their computer; or people may leave quick reviews of new computer products. Maybe you don't know how to clear the screen of your computer from within a C program. Leave a message on the BBS. Someone else may be able to help you.



```
          HUGPBBS - Heath Users' Group Personal Bulletin Board System
                              Version 2.10.M
      Copyright 1985, 1986, 1987, 1988 (c) Heath  Users' Group

  **  This System Is Available To National HUG Members ONLY. **
  **    Control-C (^C) Aborts, Control-S (^S) Pause Toggle.   **

Now you can order ANY, yes ANY HUG product listed in the REMark
product list for 20% OFF!!!!! MODEM orders ONLY. NO exceptions!

ATTENTION: Efective immediately, we have switched this system
over to use the entire HUG database!!! That means every HUG
member is automatically registered!!.If you re having trouble
logging on, use the <I> command, and read the ''OPERATING
''INFORMATION   section to see how you re supposed to do it.
Then use the <L> command to re-logon again. Good Luck!

Before ordering any item from the Bargain Centre, please be aware
of the two-letter condition code assigned to that item (see the
instructions for more information).

The SYSOP Has Allocated A Total Of 30 Minutes System Connect Time.

Enter Your FIRST Name:
```

Figure 1

```
SS --> Scan Subject Headers           C --> Database Catalog
SR --> Scan And Retrieve              U --> Upload A File
SQ --> Scan Quick                     D --> Download A File
SM --> Scan And Match                 T --> Talk To Sysop
RI --> Retrieve Individual            L --> Log-In Retry
RC --> Retrieve Continuous            G --> Goodbye (Disconnect)
E --> Enter A Message                 I --> First Time User Info
K --> Kill A Message                  B --> Retype All Bulletins
M --> Minutes Connect Time Left       H --> Print This List

OI --> How To Order Instructions      OL --> View HUG Bargain List
OO --> Place An Order
```

Some BBS's will have the message section divided into interest areas. There may be an area for For Sale messages, another for MS-DOS information, another area concerning Turbo Pascal programming, and so on. Also, some BBS's will allow you to leave messages for specific individuals. In this situation, users can only read messages addressed to them or to everyone.

A files section is another area present on many BBS's. This will be a collection of public domain and shareware software available for downloading to your computer. Before I continue, let me briefly explain the difference between public domain and shareware programs. (I have met many people who think anything found on a BBS is free to use.) Public domain software is exactly that. The author of the program has made it freely available to anyone who wants to use it. On the other hand, the author of a shareware program does expect you to pay for the software if you intend to use it. Normally, you can try out a shareware program for free to see if you like it. If after testing the program you decide to keep it and use it on a regular basis, you should send the requested payment in to the author. Simply read the documentation which comes with any program you download to determine if it is public domain or shareware.

The files section may also be divided into various program areas. For example, there may be an area for MS-DOS utilities, an area for Pascal programs, an area for games, etc. There usually will be a command to allow you to list the files available for downloading. Normally, the file names will be listed with a brief one line description of each file.

Many of the files on the BBS will probably be compressed to reduce the amount of time needed to download them. A popular file compression program used on many BBS's is ARC. ARC places all the files needed for a particular program into one archive file. It also compresses the size of each file. You can tell which files have been ARCed because they will have the extension ".ARC". You will need the ARC program (or an equivalent program) to dearchive such files. If a BBS does use ARC, it will have the ARC program itself available for download. This should be the first file you download, if you do not already have it. (By the way, ARC is a shareware program.)

Once you see a file you would like to have, select the appropriate download command from the BBS menu. You may be prompted for the type of file transfer protocol to use (Xmodem, KERMIT, etc.) After initiating the download, sit back and wait for the file. This may only take a couple of minutes, or as long as an hour, depending on the size of the file and speed of your modem.

A WORD OF CAUTION!

Remember those unscrupulous people I mentioned earlier. Some of them are even down-right mean. Perhaps you have heard of trojan- horse or virus software. This is harmful software which someone has uploaded to a BBS with the name of a useful program. Such software can do such things as destroy all the files on your hard disk or damage the COMMAND.COM file in some inconspicuous manner. Some SYSOPs will place all newly uploaded programs into an area unavailable to the BBS users. The SYSOP can then check out the program and make it available to everyone once he has tested it. Not every SYSOP does this. Therefore, be cautious of new software. (There may be a separate file area on the BBS for new or recently uploaded software.) See Pat Swayne's article, "Keeping Your System Healthy," in the June issue of REMark for some simple procedures you can use to protect your system from harmful programs.

If you discover any useful public domain or shareware programs not on the BBS you use, consider uploading the software so other users may take advantage of it. That's the greatest thing about BBS's. All users benefit from each other's contributions.

Chatting

One final feature I will mention is a chat mode. Not all BBS's have this capability. Some BBS's may allow more than one person at a time to use the system (via multiple dial-in phone lines). A chat mode allows two users logged into the BBS to carry on a conversation by typing directly to one another's computer screen. Of course, this isn't as efficient as just talking to someone by voice on the telephone. By the way, some BBS's which do not have multiple dial-in lines may provide for a chat mode between one user and the SYSOP. (The SYSOP has direct access to the computer running the BBS.)

Goodbye

When you are through using the BBS, terminate your session with a command like Goodbye, Quit, Exit, or something similiar (again, refer to the BBS' menu). Most BBS's I have seen ask if you want to leave a private message to the SYSOP before logging off. You could leave a note here about a problem you encountered while on the BBS, or a note explaining about a new piece of software you have uploaded. You may also just want to leave a note of appreciation to the SYSOP about the service he is providing.

In this article I have attempted to get you started with BBS's. I have explained what they are and why they are useful. I have also described how you can use most of them. I hope I have provided enough information to make you interested in trying a BBS. Who knows, after trying one you may like it so much you will want to become a SYSOP.

Upgrade Kits Available For The Z-100

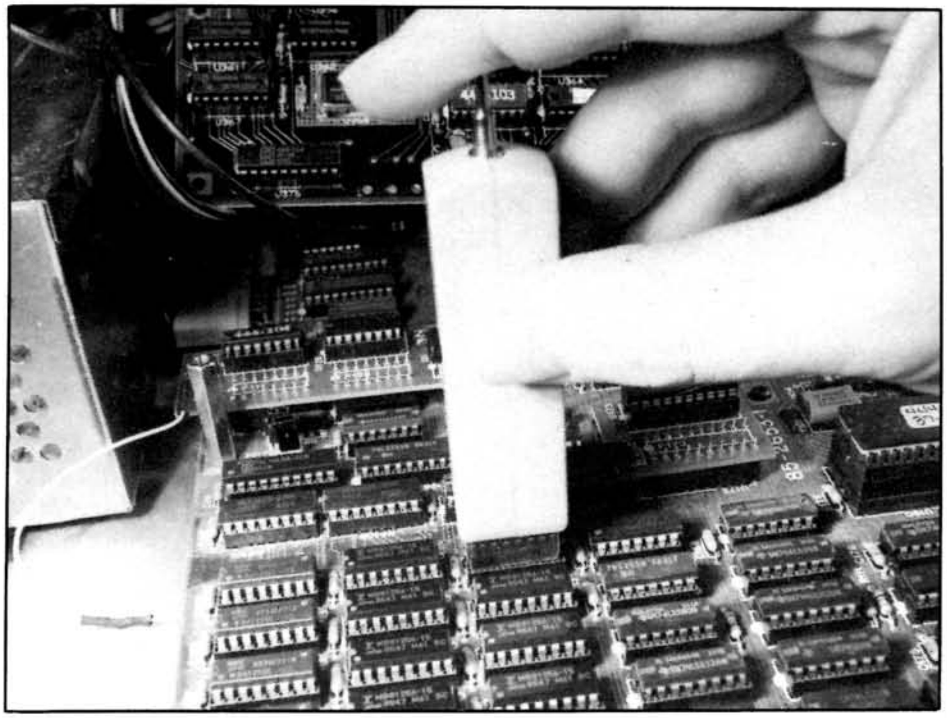
George Elwood *1670 N. Laddie Court* *Beavercreek, OH 45432*

The Z-100 computer is one of the most advanced computer systems around and was the last of the Heath designed systems before Zenith. It is better than the IBM PCs in many ways, including video and RAM memory. Can you get 400 by 640 screen resolution in color on your PC without adding another board with only software? I feel that the Z-100 keyboard is the best available on any computer. I have used Z-150s, Z-248s, IBM XTs, Sperry's, ITT 6300 to name a few and always look forward to my Z-100. Unfortunately, it is not IBM compatible and has been dropped from the rolls of Zenith computer systems. There are two systems available to degrade the computer to IBM compatibility, the Gemini cards and the UCI EASYPC cards. Of the emulators, the EASYPC seems to be the one that operates the best and is the fastest. The hardware modifications tend to slow the operation of the computer.

I have found Z-100 versions of most types of programs will run faster than the hardware emulators. Using Pat Swayne's ZPC, I have been able to run the few IBM programs I need. My son has modified many IBM programs to run under ZPC including some very powerful programs. He enjoys this and it keeps him happy. This article will cover upgrades to the Z-100 that will provide increased memory, speed, and storage capability. I will not cover the IBM emulators.

The Z-100, unlike the IBM compatibles which can address a maximum of 704k, has the capability to address 768K of RAM. The first Z-100s (the old motherboards), were capable of supporting 192k RAM on the motherboard. The later system was built to support 768k on the motherboard using 256k chips. This shows that the Heath engineers were thinking ahead. When I bought my Z-100 the old mother board was standard. The cost of chips was such that increasing memory was not feasible. I did upgrade my video memory to 192k for a mere $124. The 64k chips were very expensive then. I remember that 256k chips cost $78 each at that time (1982), about like 1Meg chips now.

As the price of chips dropped, upgrading the main memory to 768k became possible. The first memory upgrades kits were made in a garage and entailed replacing three PAL chips and two other chips on the motherboard and the 64k RAM chips with 256k RAM chips. The 64k chips are then moved to video memory. Zenith had placed 32k chips in the video memory as a cost control method. The video control board has jumpers which select 32k high, 32k low or 64k chips. As an additional comment, on the video board by changing one jumper, you can output video signals that are IBM compatible. Just change J304 on the video logic board to the "+" setting and it should work. I have done this and used the Z-100 on an ElectroHome projection device. Again, Heath engineers were thinking. This easy memory upgrade took about 30 minutes to install in the all-in-one (Z-120) comput-



Picture 1
Radio Shack Chip Inserter

ers. This included the disassembly and reassembly of the computer. I have made over 50 of these upgrades to Air Force computer systems and was able to do it in about 20 minutes each. If you have more than one computer to upgrade, buy the chip inserter from Radio Shack to protect your fingers (see picture 1). This device cost $4.98 and comes with the chip puller and is well worth the money. The memory upgrade cost is about $150 and is available from W.S Electronics.

Since I had an "old" motherboard, I could not use the above upgrade. I purchased the original FBE upgrade kit that required removing the motherboard and soldering a wire between all of the pin ones on the memory sockets. This procedure sounded hard but turned out to be quite easy. The Z-100 is the last of the Heath designed computers and the construction is well conceived. When was the last time you saw all of the chips in sockets in a computer? The entire computer comes apart easily and the motherboard was out in about 10 minutes. You mark the pin locations and start to solder. This procedure takes about 15 minutes. You reconstruct the computer and install the 256k chips in the memory sockets and add a small PC board on the motherboard to complete the installation.

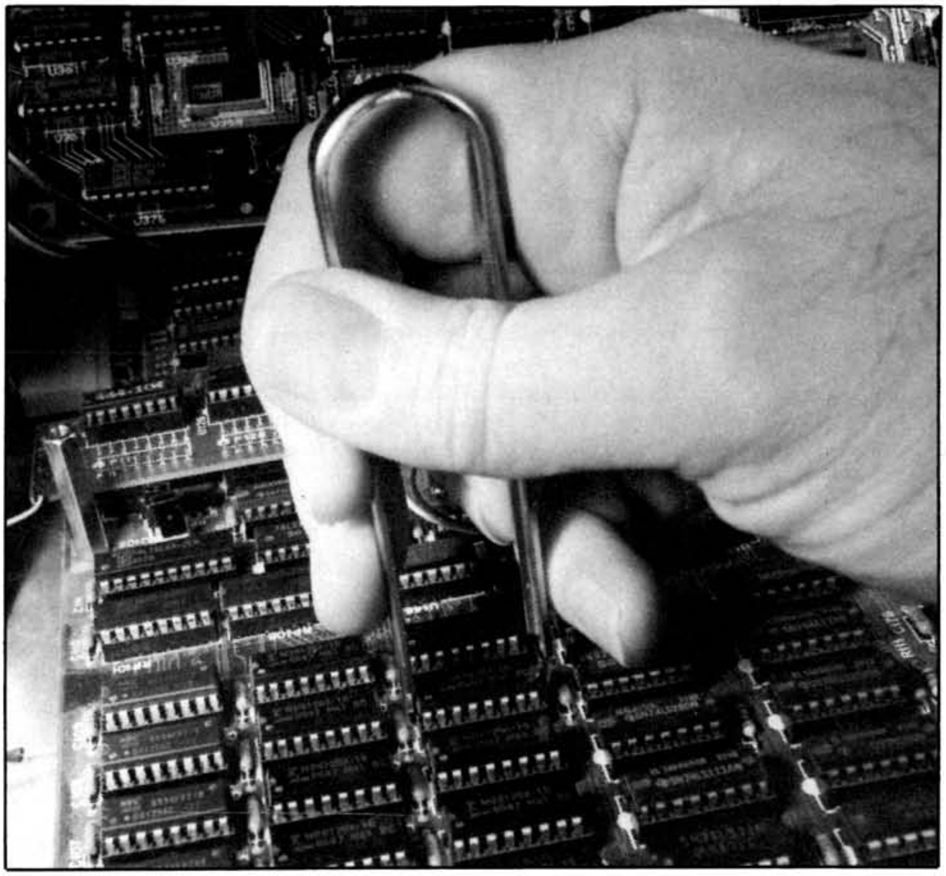
FBE has since come out with an upgrade for the "old" motherboard Z-100s. This consists of a wire bar that fits in the pin one slot on the memory sockets. You remove the 64k memory chips from the motherboard and insert the connector bar into the sockets using a pencil eraser to seat them. I have also done this upgrade and it is not hard. It is considerably easier and faster than the old way that required soldering.

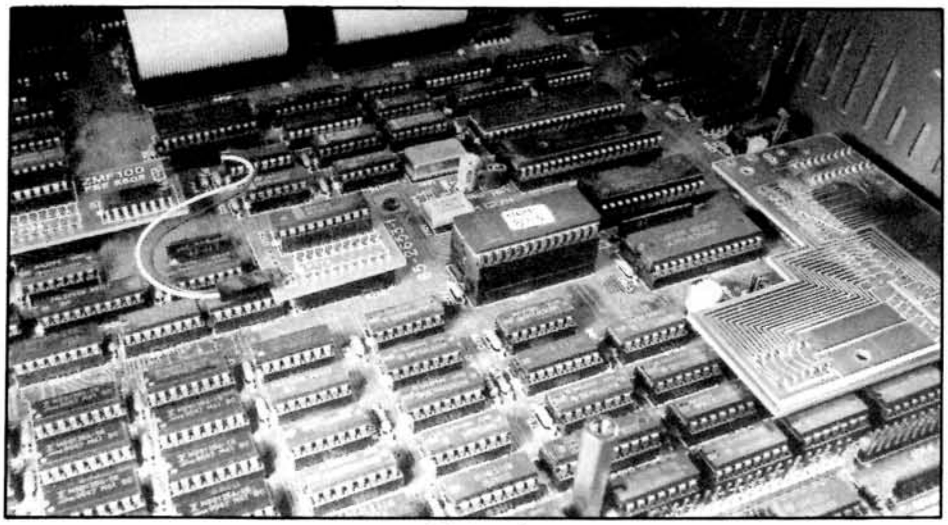
Zenith also offered a motherboard that had 256k chips installed and operated at 8MHz. The Z-118 low profile computer was configured in this manner.

The memory upgrades to the Z-100s are easy and are required if you plan on continued operation of you computer. The memory upgrade, including the video memory upgrade, is necessary if you plan on using the ZPC emulator software package.

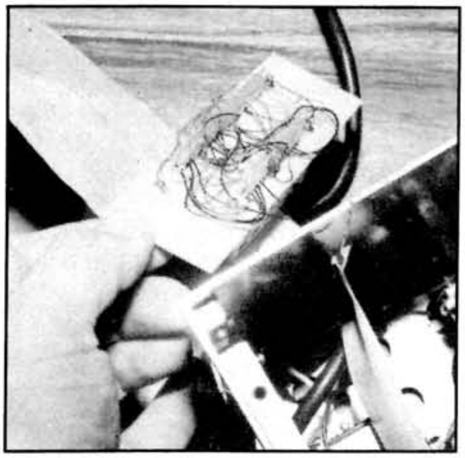
While you have your computer opened up and have the chip puller in hand, I highly recommend adding a speedup kit. These kits offer a 50 to 60 percent increase in speed over the standard unit. There are two kits available, a 7.5MHz kit from CDR and UCI is again offering the 8MHz kit. The UCI kit is more extensive in that it provides chips that may be slow on the motherboard while the CDR kit is just a small board with a new crystal and a

Picture 2
Integrated Chip Puller





Picture 3
FBE Memory Upgrade, UCI Speedup Kit, and SmartWatch, Installed in Z-100



Picture 4
Adapter plug to connect Z-100 8" drive output and 5-1/4" high capacity drive.

switch. In the 11 CDR kits I have installed, I have had about a 75 percent success rate on the first try. The other computers required additional chip replacements.

I installed the UCI speedup kit in my computer several years ago and it worked right off the bat. UCI now offers two speedup kits, the ZM8100 which has 25 chips in the kit and two small boards and the ZM9100 which has 29 chips and one board. This kit also offers the memory upgrade PALs without the 256k memory chips. I replaced my 8086 with an 8MHz NEC V-20 chip which is supposed to offer a 5 to 40 percent increase in processing speed. The installation is very straightforward and the instructions are very clear. If you installed the ZMAX memory upgrade kit, do not replace these new chips as they are already high speed or use the ZM9100 if you plan to do the entire upgrade.

Using the UCI speedup kit and the NEC V-20 chip, my Z-100, running the ZPC emulator package, runs at 2.6 using the Norton Utility SI program. The results of the PC Magazine benchmark testing routines are shown in figures 1-4 below.

PC Magazine Laboratory Benchmark Series
BENCH28 Version 1.01: Processor Speed Instruction Mix Test

This test measures processor speed by executing a mix of Assembly Language instructions. A higher "Speed Index" means a faster execution. You may see variations in hundreths of seconds if you repeat the test. Tests using the 80286 and 80386 instruction sets have not yet been implemented.

	Time in Seconds	Speed Index Relative to 4.77 MHz PC	Speed Index Relative to 8.00 MHz AT
8086/8088 Instruction Set:	16.88	1.9	0.5

Figure 2

PC Magazine Laboratory Benchmark Series
BENCH21 — Processor Speed Benchmark Test — Version 1.31

The following Tests 1 to 4 are written in Assembly Language and take about 10 seconds each on a standard IBM PC. Tests 5 and 6 are written in Microsoft C 4.00. A higher "Speed Index" means a faster execution. You may see variations in hundredths of seconds if you repeat the test.

	Time in Seconds	Speed Index Relative to 4.77 MHz PC	Speed Index Relative to 8.00 MHz AT
1. 128K NOP Loop:	6.72	1.5	0.6
2. Do-Nothing Loop:	6.12	1.6	0.6
3. Integer Add Loop:	6.02	1.7	0.4
4. Integer Multiply Loop:	3.72	2.7	0.3
5. String Sort and Move:	5.80	1.8	0.5
6. Prime Number Sieve:	9.73	1.6	0.4

Figure 1

One last modification for the Z-100 before you close it up would be the SmartWatch clock chip. This small device, which looks like a chip holder, will provide the date and time for your Z-100 for the next 10 years. This unit has a built-in battery and IC which provides the system a date and time so you can by-pass this prompt on boot up. A small program is included with the chip and it is added to the AUTOEXEC.BAT file to load these upon booting. The unit is placed under the PROM located at U190. With the Z-100, you have to buy the extension kit to raise the video board about a 1/8 inch to clear the added height of the PROM.

If you have been following my series of articles on ENABLE in REMark, you will understand the need for a high capacity storage system. I use ENABLE on my system by placing part of the program on two disks. The Z-100 version of ENABLE comes on six disks. It is possible to oper-

```
                 PC Magazine Laboratory Benchmark Series
     BENCH29 Version 1.01: Floating Point and Math Coprocessor Test

The microprocessor is the NEC V20. An 8087 math coprocessor is NOT installed.
An Equipment Check call says a coprocessor is NOT installed.

The following floating point tests were written in Microsoft C 4.00. The first test
uses the math coprocessor emulation library and takes about 2.5 minutes on a
standard 4.77 MHz PC. The second test uses the math coprocessor and will be
performed only if one if present.

                                          Speed Index     Speed Index
                             Time in      Relative to     Relative to
                             Seconds      4.77 MHz PC     8.00 MHz AT
Floating point (no 8087):     91.53           1.7             0.4
```

Figure 3

```
                 PC Magazine Laboratory Benchmark Series
             BENCH20 — Memory Access Speed Benchmark Test
                              Version 1.12

This benchmark allocates 256K bytes of conventional, Lotus/Intel/Microsoft ex-
panded, and AT-type extended memory and treats it as a series of 64-byte rec-
ords. 16,384 random records are then read into and written from local memory.
Speed indices for expanded and extended memory are based on the Intel Above
Boards.

Expanded Memory (Lotus/Intel EMS):                    — None —
Extended Memory (Protected Mode 80286                 — None —

                                          Speed Index     Speed Index
                             Time in      Relative to     Relative to
                             Seconds      4.77 MHz PC     8.00 MHz AT
Conventional Read:             2.38           2.5             0.6
Conventional Write:            2.37           2.5             0.6
```

Figure 4

```
dir /w

 Volume in drive C in ENABLE D
 Directory of  C:\

PØ11     COM    PROFILE  $PR    TPSETUP  $TP    TUTOR    BAT    ENABLEZ  EXE
UTL2ØZ   TSG    CHK2ØZ   TSG    HLP1M2ØZ TSG    HLP2M2ØZ TSG    SPLLM2ØZ TSG
SPLLDICT TSG    ENABLE   BAT    TOL2ØZ   TSG    DBMSDEF  $RF   LETTER   TSG
LABEL    TSG    MAIL     $BF    MAIL     DBF    MAILIN   $IF    ITALIC1  FNT
BLOCK2   FNT    SCRIPT1  FNT    ITALIC2  FNT    ROMAN2   FNT    BLOCK1   FNT
ROMAN1   FNT    SCRIPT2  FNT    USERDICT BAD    DEF2ØZ   TSG    PRT2ØZ   TSG
MACOM    BAT    MATTS    BAT    PØØ8     COM    HLP3M2ØZ TSG    PØØ1     COM
EP1      TSG    USERDICT TSG
       37 File(s)     37888 bytes free

C>dir

 Volume in drive D is IGSAFE2
 Direcotyr of  D:\

SYS2ØZ   TSG   357756   2-12-87  1Ø:Ø4a
OPR2ØZ   TSG   35626Ø   2-13-87  11:23a
        2 File(s)    535552 bytes free
```

Figure 5

ate on a dual floppy system, but this is painful after awhile. Most operations take one or two disk swaps to complete and if you plan on extensive work, you will not want to continue. Although I completed the 1.2 Meg floppy system before I got ENABLE, I do not think I would have moved from WordStar, LOTUS, and dBase II if it had not been for this system.

In 1984 I wanted to upgrade the storage capability of my Z-100. Winchester disk drives were expensive and my better half would not authorize the Purchase Request. The Z-100 has an eight inch floppy drive connector in addition to the 5 1/4 inch connector. A dual sided eight inch drive would provide 1.2 Meg of storage. I read about the Teac FD-55FGV-17 1.2 Meg floppy disk drives and I thought that I could adapt this drive to look like an eight inch drive on the Z-100. By studying the documentation that came with the Z-100, remember all of those books, I was able to come up with the jumpers required. The first systems I built used a wire wrap 50 pin and 34 pin connector. I mounted these connector on a small piece on perf board and ran the wires. This made it easy to make changes as necessary. I thought I killed my computer once but a power off and back on restored it. Kids banging on the keyboard do not hurt a computer but I sure try.

I run a 50 wire ribbon cable from the computer to the adapter and a 34 wire ribbon cable to the floppy drives. I have built six or so of these cables for members of the Dayton Heath/Zenith Users Group (DAYHUG). In the end I built the cables as one piece with a 50 pin connector on one end and a 34 pin header on the other. This mounted in the rear of the Z-100 making movement easier. All that is needed is a 34 pin header at one end and an edge card connector on the other end of a cable that is long enough to reach from the back of the Z-100 to the disk drives. These cables are available from various sources now.

All odd numbered pins should be grounded. The drive should be a TEAC FD-55FGV-17. There are many drives TEAC drives but the -17 appears to be the best. The following jumpers on the drive should be set on: FG HL HG RY II and the drive select DS0 or DS1.

Initially I ran this setup laying out on the table using a Radio Shack switching power supply. A friend built me a case which permitted me to mount all of the compo-

50 Pin connector at the controller	34 Pin connector at the drive
10	Ground
14	32
18	4
20	8
22	34
26	10
28	12
30	14
32	6
34	18
36	20
38	22
40	24
42	26
44	28
46	30
	Ground

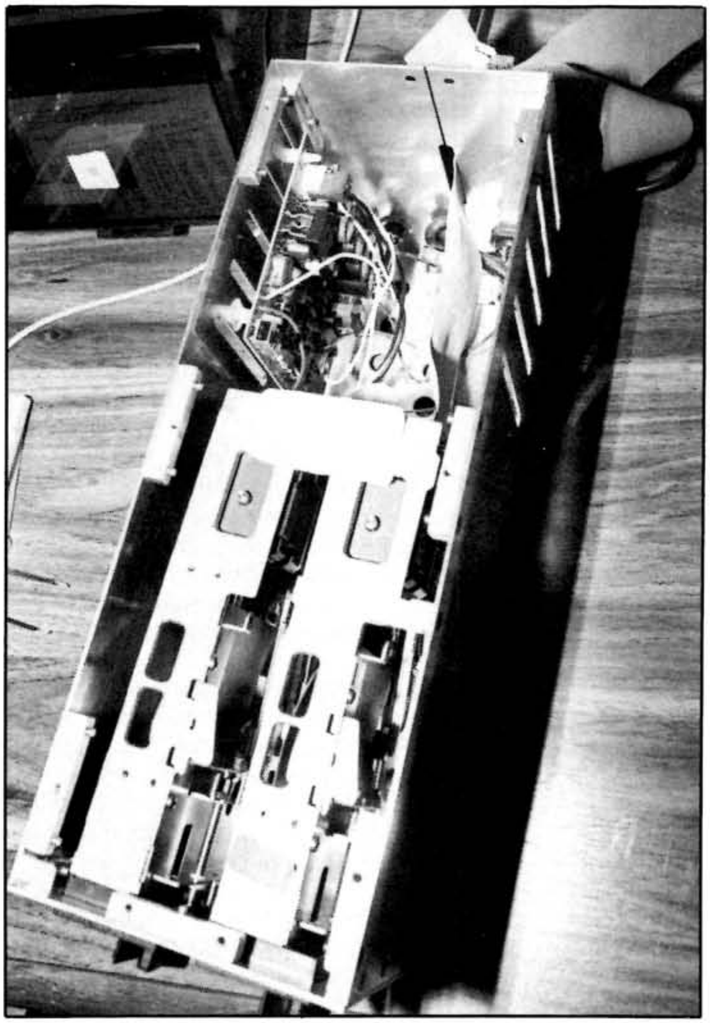
Figure 6

nents inside. I have since added another disk drive for a total of 2.4 Meg memory. The Radio Shack power supply handles both drive without problems. When you mount the power supply, use nylon standoffs. If you use metal fasteners, you can ground the power supply and burn up the transformer. I built several of these systems for other members of DAYHUG over the past few years and we have not had any problems. You can save yourself the time and trouble by buying a case and power supply for about $60.

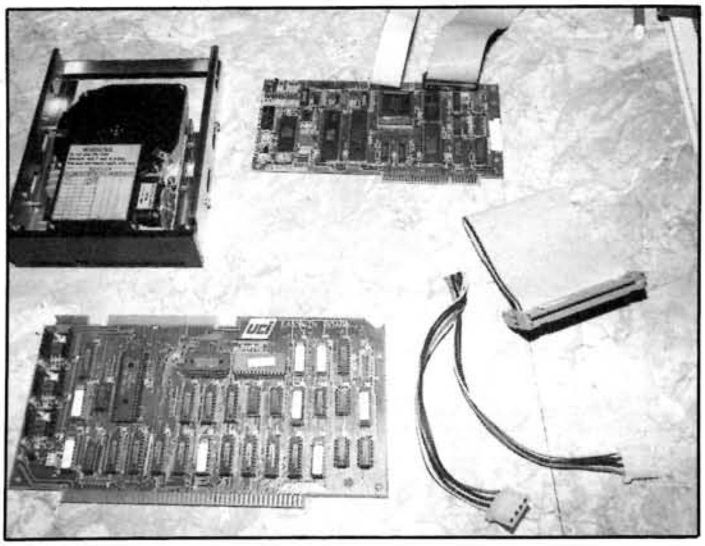
John Pierce, the DAYHUG software guru, modified format and other MS-DOS programs to provide 1.3 Meg per drive by placing additional tracks on the disk. These modifications are shown in listing 1-4 in John's article.

Recently both UCI and CDR have offered hard disk upgrades for the Z-100. These kits install in the Z-100 and offer 20 or 30 Meg of disk space. The CDR system uses the SCSI interface and will use the Zenith winchester software. I have not installed one of these systems so I can not provide any details on installation or operations.

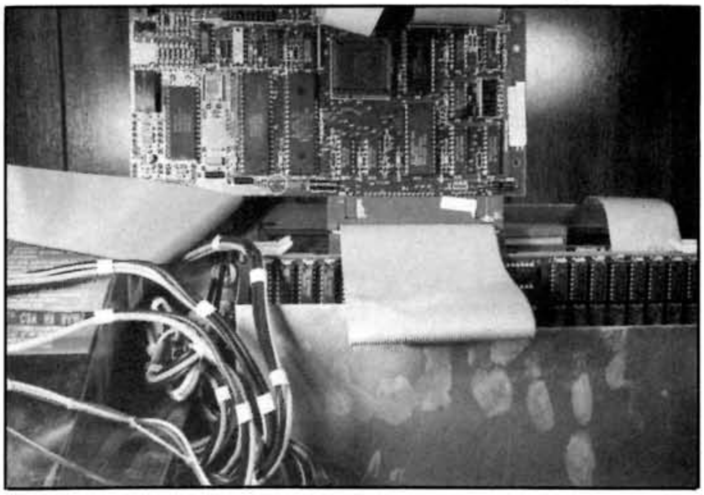
The UCI system supports either a 20 or 30 Meg Winchester hard disk system. The difference is in the Western Digtal controller. The Seagate ST-225 provides 20 Meg using the MFM format while the ST-238 provides 30 Meg using the RLL format which is provided by the controller. The UCI card contains an 8088 controller chip and provides an IBM type buss on top of the card. From this buss a ribbon cable runs to the hard disk controller card. If purchased from UCI or one of the other suppliers, the Western Digital controller and Seagate ST-225 are provided. W.S Electronic, Xenia, OH provides Western Digital controller and Fuji FK309-26 hard



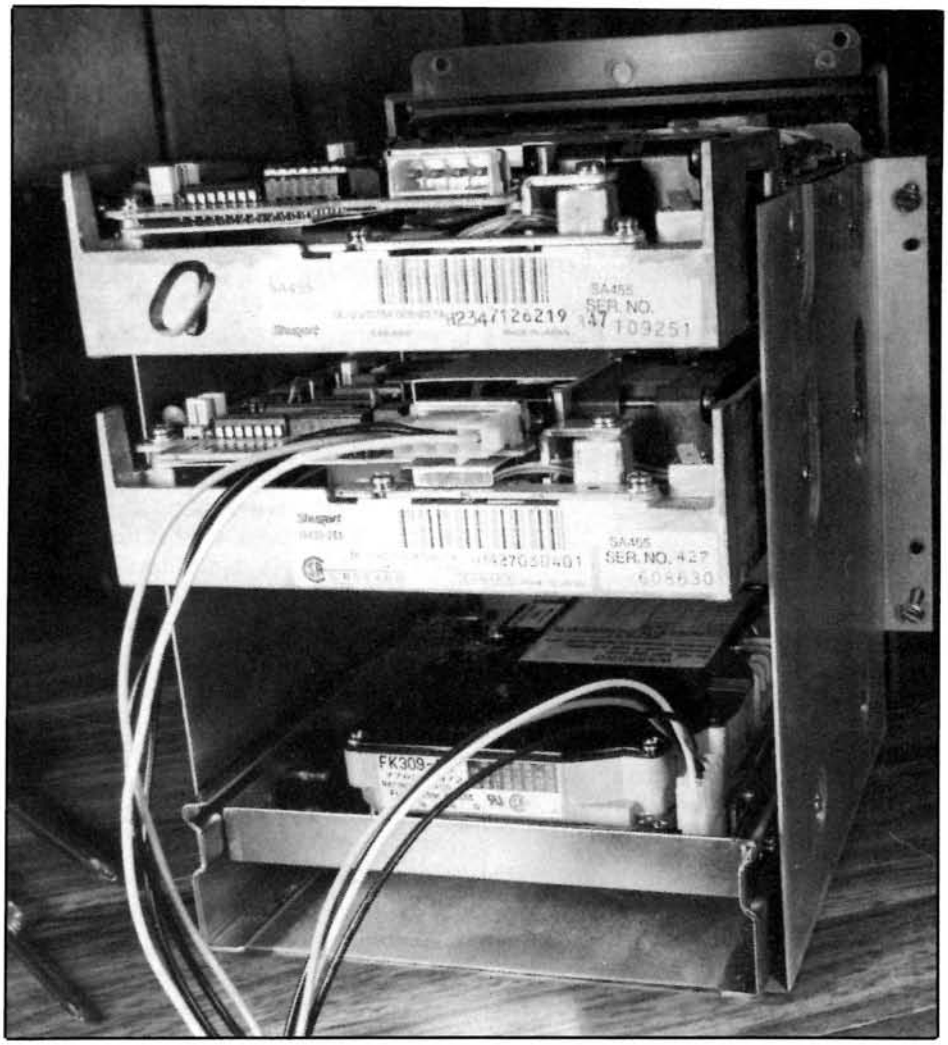
Picture 5
Inside of External 1.2 Meg Floppy Disk Drive Cabinet



Picture 6
UCI Card, Controller, Hard Disk and Cables



Picture 7
UCI Card and Controller Mounted in Computer

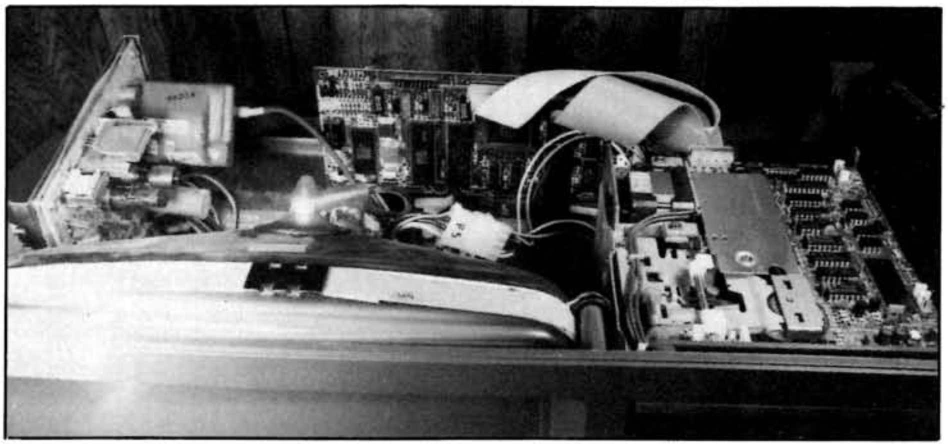


Picture 8
Hard Disk Mounted in All-in-One Floppy Cage

drive for the 20 Meg system (FK-305-37R for the 30 Meg). This system has proven to be highly reliable. The drive has a seek time of 65 Msec, the same as the Seagate. It does not generate as much heat as the Seagate and appears to be more reliable.

Installation is easy and takes but a few minutes. I did some test installations of the UCI system for the Air Force. During these tests I found that when installed in the Z-120, the Seagate drive would bind slightly and not operate if the screws were tightened. The solution was to place small nylon washers between the drive case and the drive. The Fuji drives are a 3 1/2 inches drive in a heavy frame that permits installation in a 5 inch slot and have not had any problems. The Air Force organization I was in purchased and installed 20 of these units in Z-120 computers so that they could run ENABLE. Installation took about 30 minutes per machine.

Picture 9
UCI controller mounted in computer.
Note that the WD controller lies across the tube neck.

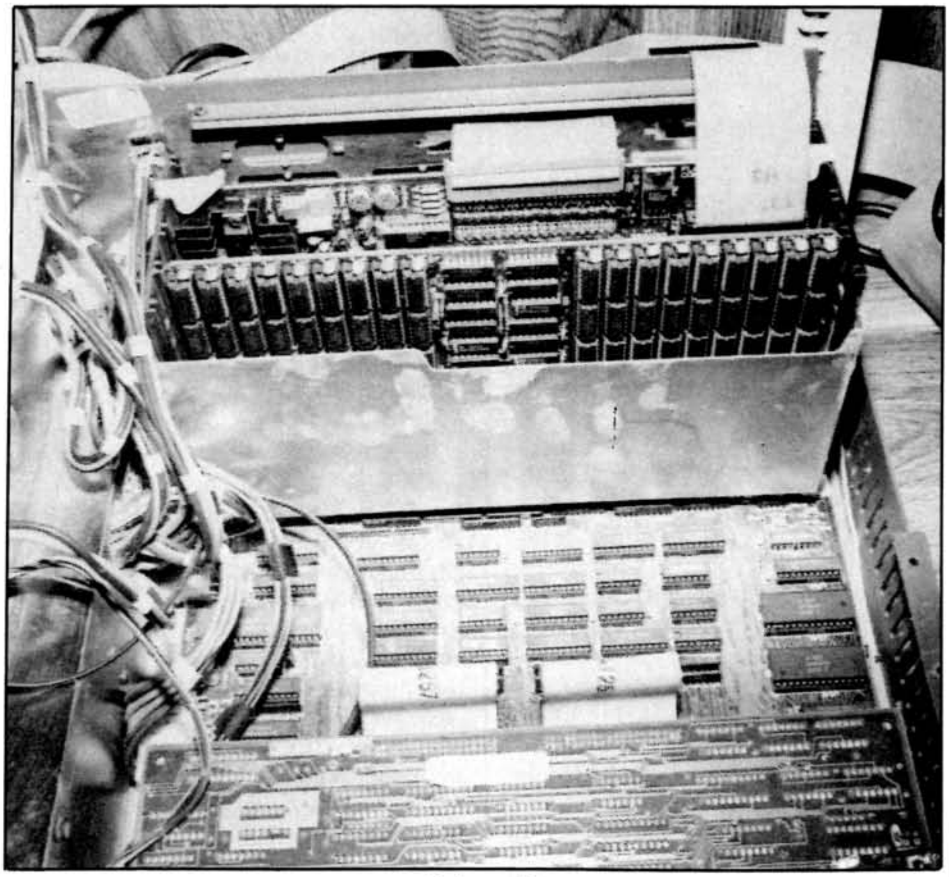


This modification in the Z-120 permits you to maintain both of the installed half height floppy disk drives and have the hard disk mounted below. This will only work if you have the two half height drives, the full height drives use all available space. If you like to see the drive access light, you must drill a small hole in the front plate, although this is not necessary. You must remove the bezel from the hard drive in order for it to fit. The best way to install the hard disk controller card is to mount it directly above the UCI card using double sided sticky tape (see picture 6). UCI recommends mounting this on the hard disk itself, but I found this to be unsatisfactory. The cables could touch the bottom of the floppy disk drive and cause problems.

Mounting the hard disk in the low profile Z-100 will result in the loss of one floppy disk drive. The disk controller card is mounted along side the hard disk instead of above the S-100 controller. Note in the pictures the WD controller is long. It will fit over the neck of the tube. The half size RLL WD controllers are better in that they do not extend beyond the card cage. Both the UCI and the added instruction provided by W.S Electronics give you all the information necessary to install the drives. Note: read the instructions completely before starting. If you wish to boot from the hard drive you must reset S101 on the Z100 motherboard so that switches 1 and 2 are on.

John Pierce of DAYHUG, modified the UCI EASYWIN driver so that it is not necessary to load the entire program to "SHIP" the drive before turning off the system. You only type "EASYWIN/S" and the heads move to a safe location. He also disabled the E, F, G and H areas in the OS areas to prevent possible problems. These modification are shown in listing 5 or John's article.

The only other addition I have made to my computer is the UCI RAM board. I have a 2 Meg board installed and I use this as the target drive when using



Picture 10
UCI RAM Board

ENABLE and any other operation that will be disk intensive. As an example, I did a sort on this board that took 20 seconds to complete. On a hard disk with 65msec access, this same operation took over five minutes. If you do this type operation, you may consider this board. There are two versions of this board in being, I have the first board and it would not operate with the 768k motherboard memory. Again, John Pierce was able to modify the board using a spare gate on the Z-100 floppy disk controller and a jumper on the UCI board. He also had to modify the UCIBRAM driver to work with this modification. UCI has since modified the board and driver to support the 768k upgrade but the new boards and driver are NOT compatible with the old boards. He has also modified the old driver to work with the new board.

All of these modifications are available from W.S. Electronic. This small firm is a Zenith dealer and provides tremendous support to DAYHUG. They also do a lot of business with Wright-Patterson AFB and are familiar with handling government orders. The Z-100 is a great machine and these modifications will increase the usability of the system for many years to come.

Part list
Radio Shack

Switching Power Supply	#273-1080	$4.98
Fuse Holder	#270-364	$1.09
On-Off Switch	#275-6901	$1.89
IC Inserter/Puller	#276-1581	$6.95
AC Line Cord w/GND	#278-1258	$2.99
Power Transformer	#273-1515	$ .69

W.S Electronic
1106 State Route 380
Xenia, OH 45385
(513) 376-4348

UC	EASYWin w/ 20 Meg Hard Disk	$550
	EASYWin w/ 30 Meg Hard Disk	$580
	UCIBRAM basic board (no memory chips)	$225
	8 MHz Speedup kit 2M8100	$132.50
	8 MHz Speedup kit 2M9100	$139
FBE	Memory Upgrade Kit w/ 768k	$175
	SmartWatch w/Z100 extention kit	$45
ZMAX	Memory Upgrade w/768k	$150

NOTE: The following modifications should be made on a backup copy of your software. NEVER make modifications to your master disks.

IO.SYS Modifications for High Capacity Drives

By John C. Pierce
System Software Consultant
work 513-252-2402 home 513-879-1332

Z100's which have high capacity 5 1/4" drives installed instead of 8" drives, can be used to both read and write disks in several formats. The two most common formats are the 8" DS/DD format and the AT format. These drives cannot be used in single sided mode and the modifications to IO.SYS presented here take advantage of that fact - i.e., a modified IO.SYS will no longer process single sided 8" disks.

The 8" DS/DD format (which I will call the Z100 HD format) uses 8 1024 byte sectors on each track of a 77 cylinder disk.

8 sectors/track * 1024 bytes/sector * 2 tracks/cylinder * 77 cylinders yields 1,250,304 bytes per formatted disk.

The AT DS/DD format uses 15 512 byte sectors on each track of an 80 cylinder disk.

```
15 * 512 * 2 * 80 yields 1,213,952 bytes per formatted disk.
```

A third format, which I will call Extended Z100 HD, is the same as the Z100 HD format but extended to 80 cylinders.

```
8 * 1024 * 2 * 80 yields 1,299,456 bytes per formatted disk.
```

The modification files BPB.SET and CODE.SET contain the modifications for IO.SYS 2.18 and 2.21 which will allow your system to automatically adjust to whichever of these three formats is present on a disk. CODE22.SET and BPB.SET are used with IO.SYS 2.22. CODE3.SET and BPB.SET are used with IO.SYS 3.10. They are dynamic enough that you can have one format in drive C: and another in drive D:, even if you have only one physical drive.

The contents of the four files are shown in Figure 1. All numbers are in hexadecimal. The easiest way to build the required files is using DEBUG. For example, to create BPB.SET use the following instructions:

```
nBPB.SET (c/r)                 to set the file name
e100 00 04 ..... 07 00 (c/r)   to enter the 26 bytes
rCX (c/r)
1A (c/r)
rBX (c/r)
00 (c/r)                       to set to write 26 bytes
w                              to create the file.
```

All of these files, plus those for the FORMAT.COM and DISKCOPY.COM modifications which follow, are available from DAYHUG on their MARCH 1987 Disk Of the Month (DOM) for $5 which includes shipping.

In all of the following installation instructions, all files are assumed to be on the default drive.

To install temporarily (until you reboot), enter DEBUG and issue the following set of commands:

```
2.18 and 2.21    2.22          3.10
  nBPB.SET       nBPB.SET      nBPB.SET      table modifications
  140:9AB        140:9A3       140:A0B       load to proper place
  nCODE.SET      nCODE22.SET   nCODE3.SET    code modifications
  140:706        140:6FE       140:732
  q              q             q             quit
```

To install permanently, remove the SHR flags from a copy of IO.SYS and issue the following set of commands:

```
2.18 and 2.21    2.22          3.10
  DEBUG IO.SYS   DEBUG IO.SYS  DEBUG IO.SYS
  nBPB.SET       nBPB.SET      nBPB.SET
  1AAB           1AA3          1B0B          note the change from above
  nCODE.SET      nCODE22.SET   nCODE3.SET
  1806           17FE          1832
  rcx            rcx           rcx
  3A01           3A11          3F9E          this should be what was in CX
  nIO.SYS        nIO.SYS       nIO.SYS       after the DEBUG IO.SYS command
  w              w             w
  q              q             q
```

Then reset the SHR flags on the resulting file. FLAGS from the Programmer's Utility Pack is the preferred method to manipulate directory flags, but if you don't have it, use DEBUG to set the 12th byte of the IO.SYS directory entry to zero to remove the flags and back to 07h or 27h to restore them.

These mods have only been tested on IO.SYS 2.18, 2.21, 2.22, and 3.10. I will adapt them to any other 2.xx or 3.xx version of IO.SYS if you send me a copy of your operating system to work with.

Figure 1. File Contents for IO.SYS Modification

```
          Contents of BPB.SET (1A bytes)

00 04 01 01 00 02 C0 00-00 05 FD 02 00 00 02 01
01 00 02 E0 00 60 09 F8-07 00

          Contents of CODE.SET (64 bytes)

2E C6 45 0F 50 EB 3D 57-B8 01 00 E8 BE 01 5F 2E
80 3E E0 31 FD BE AB 09-74 20 2E 80 3E E0 31 F9
74 0A BE 84 09 2E C6 45-0F 4D EB 0E BE B8 09 2E
C7 45 12 00 02 2E C6 45-11 0F 2E 89 36 4C 07 E9
A1 00 90 90 F8 C3 00 00-8A 47 01 98 D1 E0 8B F8
2E 8B BD 59 0E 80 7F 01-02 73 03 EB 21 90 80 7F
01 04 72 A3

          Contents of CODE22.SET (64 bytes)

2E C6 45 0F 50 EB 3D 57-B8 01 00 E8 BE 01 5F 2E
80 3E F0 31 FD BE A3 09-74 20 2E 80 3E F0 31 F9
74 0A BE 7C 09 2E C6 45-0F 4D EB 0E BE B0 09 2E
C7 45 12 00 02 2E C6 45-11 0F 2E 89 36 44 07 E9
A1 00 90 90 F8 C3 00 00-8A 47 01 98 D1 E0 8B F8
2E 8B BD 51 0E 80 7F 01-02 73 03 EB 21 90 80 7F
01 04 72 A3

          Contents of CODE3.SET (70 bytes)

2E C6 45 0F 50 EB 3D 57-B8 01 00 E8 F2 01 5F 2E
80 3E 30 34 FD BE 0B 0A-74 20 2E 80 3E 30 34 F9
74 0A BE E4 09 2E C6 45-0F 4D EB 0E BE 18 0A 2E
C7 45 12 00 02 2E C6 45-11 0F 2E 89 36 78 07 E9
AD 00 90 90 F8 C3 00 00-4E 4F 20 4E 41 4D 45 20
20 20 20 00 8A 47 01 98-D1 E0 8B F8 2E 8B BD 00
0F 80 7F 01 02 73 03 EB-21 90 80 7F 01 04 72 97
```

FORMAT Modifications for High Capacity Drives

FCODE.SET, FTABLE.SET, and FBOOT.SET contain the modifications for FORMAT.COM which allow it to format high cacacity disks in the Extended Z100 HD format (see IO-SYS.DOC). They have been tested in FORMAT.COM 2.18, 2.21, and 2.22 (which are identical except for the version number).

FCODE3.SET, FTABLE3.SET, and FBOOT.set are the modifications for FORMAT.COM 3.03, supplied with MS-DOS 3.10.

The contents of all of the files is shown in Figure 2. The IO.SYS modification write-up, above, gives an example of how to use this data to create the required files for your system.

To modify a copy of FORMAT.COM, execute the following commands:

```
      2.xx                                        3.03

    DEBUG FORMAT.COM                            DEBUG FORMAT.COM
    nFCODE.SET                                  nFCODE3.SET
    1F3A                                        1197A
    nFTABLE.SET                                 nFTABLE3.SET
    115D3                                       120AE
    nFBOOT.SET                                  nFBOOT.SET
    11C24                                       126FF
    rcx                                         rcx
    356A                                        40C5
    nFORMAT.COM                                 nFORMAT.COM
    w                                           w
    q                                           q
```

The new FORMAT.COM will produce the Extended Z100 HD format when the /9 switch (/8 switch in v3.03) is used while formating drive C: or D:. The new FORMAT.COM will no longer be able to format single sided disks on drives C: or D:.

Figure 2. File Contents for FORMAT.COM Modifications

Contents of FCODE.SET (0D bytes)

```
20 00 75 03 E9 F1 00 BE-D0 15 E9 EB 00
```

Contents of FTABLE.SET (0B bytes)

```
A3 00 A0 00 72 04 01 02-00 F4 17
```

Contents of FBOOT.SET (13 bytes)

```
00 04 01 01 00 02 C0 00-00 05 0A 08 0B 00 00 05
 00 01 1C
```

Contents of FCODE3.SET (0D bytes)

```
20 00 75 03 E9 F1 00 BE-2B 1A E9 EB 00
```

Contents of FTABLE3.SET (0B bytes)

```
A3 00 A0 00 72 04 01 02-00 4F 1C
```

DISKCOPY Modifications for High Capacity Drives

DCODE1.SET, DCODE2.SET, DCODE3.SET, DTABLE.SET, and FBOOT.SET contain the modifications to DISKCOPY.COM which allow it to copy high cacacity disks in the Extended Z100 HD format (see IO- SYS.DOC). They have been tested in DISKCOPY.COM 2.18, 2.21, and 2.22 (which are identical except for the version numbers).

DCODE1-3.SET, DCODE2-3.SET, DCODE3-3.SET, DTABLE-3.SET, and FBOOT.set are the modifications for DISKCOPY.COM 3.04, supplied with MS-DOS 3.10.

The contents of all of the files is shown in Figure 3. The IO.SYS modification write-up, above, gives an example of how to use this data to create the required files for your system.

To modify a copy of DISKCOPY.COM, execute the following commands:

```
  2.xx                                        3.04
DEBUG DISKCOPY.COM                          DEBUG DISKCOPY.COM
nDCODE1.SET                                 nDCODE1-3.SET
14AD                                        1BF9
nDCODE2.SET                                 nDCODE2-3.SET
1174A                                       1223A
nDCODE3.SET                                 nDCODE3-3.SET
11D09                                       12865
nDTABLE.SET                                 nDTABLE-3.SET
11DE3                                       1296E
nFBOOT.SET                                  nFBOOT.SET
12434                                       12FBF
rcx                                         rcx
3D7A                                        4985
nDISKCOPY.COM                               nDISKCOPY.COM
w                                           w
q                                           q
```

The new DISKCOPY.COM will no longer be able to copy single sided disks on drives C: or D:.

Figure 3. File Contents for DISKCOPY.COM Modification

Contents of DCODE1.SET (3 bytes)

```
E9 71 18
```

Contents of DCODE2.SET (0D bytes)

```
20 00 75 03 E9 F1 00 BE-E0 1D E9 EB 00
```

Contents of DCODE3.SET (2D bytes)

```
0D 0A 45 72 72 20 2D 20-4D 41 50 20 69 6E 73 74
61 6C 6C 65 64 0D 0A 24-8B 54 09 1F 81 FA 00 05
 75 06 81 0E A7 14 20 00-A8 01 E9 7A E7
```

Contents of DTABLE.SET (0B bytes)

```
A3 00 A0 00 72 04 01 02-00 04 20
```

Contents of DCODE1-3.SET (3 bytes)

```
E9 81 1C
```

Contents of DCODE2-3.SET (0D bytes)

```
20 00 75 03 E9 F1 00 BE-EB 22 E9 EB 00
```

Contents of DCODE3-3.SET (2D bytes)

```
0D'0A'45'72'72'20'2D'20-4D'41'50'20'69'6E'73'74
61'6C'6C'65'64'0D'0A'24-8B'54'09'1F'81'FA'00'05
75'06'81'0E'97'16'20'00-A8'01'E9'6A'E3
```

Contents of DTABLE-3.SET (0B bytes)

```
A3 00 A0 00 72 04 01 02-00 0F 25
```

*

Classified Ads

REACH THOUSANDS OF DEDICATED HEATH/ZENITH COMPUTER OWNERS!!

Classified Ad Rates

There is a minimum of $5.00 for 10 words or less. Each additional word is $.40. Payment for all classified advertising must be in advance.

Display Classified is available at $35.00 per column-inch. Minimum one inch. Display Classified is commissionable to recognized agencies.

How To Count Words

Count one word each for initials, standard abbreviations, whole numbers, name, address, city, state, zip, area code and telephone number. Dimensions (such as 6 × 9) are counted as one word. Box or department numbers are counted as one word each. All classified ads (not display) are set in the same size type. The first several words, depending on the ad, are set in all caps.

Continuity Discounts

Run your ad at least 3 times during the year; and receive a discount off the regular price.

Run 3 insertions — 3% off, 4 insertions — 4% off, 5 insertions — 5% off, . . . 12 insertions — 12% off.

Issue & Closing Date

Issue Date	Closing Date	Issue Date	Closing Date
January	November 15	July	May 15
February	December 15	August	June 15
March	January 15	September	July 15
April	February 15	October	August 15
May	March 15	November	September 15
June	April 15	December	October 15

REMark Magazine has the right to refuse any ad for any reason.

Classified Order Blank

First	10	words	at	a
cost	of	$5.00	each	insertion.
11-$5.40	12-$5.80	13-$6.20	14-$6.60	15-$7.00
16-$7.40	17-$7.80	18-$8.20	19-$8.60	20-$9.00
21-$9.40	22-$9.80	23-$10.20	24-$10.60	25-$11.00
26-$11.40	27-$11.80	28-$12.20	29-$12.60	30-$13.00
31-$13.40	32-$13.80	33-$14.20	34-$14.60	35-$15.00
36-$15.40	37-$15.80	38-$16.20	39-$16.60	40-$17.00
41-$17.40	42-$17.80	43-$18.20	44-$18.60	45-$19.00
46-$19.40	47-$19.80	48-$20.20	49-$20.60	50-$21.00

Enclosed is a check or money order of $________ for ________ words.
(Minimum order: 10 words for $5.00. Each additional word $.40.)
Please insert this advertisement in the________________________________issue.

Signature____

(Please type or print)

Name________________________________Phone____________

Company____

Address____

City______________________________State__________Zip__________

MAILING ADDRESS: Make check or money order payable to: Rupley's Advertising Service, Dept. — REM, 240 Ward Avenue, P.O. Box 348, St. Joseph, MI 49085, (616) 983-4550.

C SCIENTIFIC LIBRARIES Personal CSL $85, Personal Statistics $99, and Personal Matrix $99. Personal libraries are subsets from CSL (440 functions) available for $275, source code optional $75. Each routine is designed and documented for use by technical specialist. Call (408) 867-6575 or write Eigenware, 13090 La Vista Drive, Saratoga, CA 95070.

USED, BUT CHECKED and verified good, HS-218, Syquest, 10-meg cartridges. Original cost, $99.95. Order yours today for only $25 each. Call (616) 982-3838 EST.

NEW HEATH SL-5410 convenient entry way motion sensor light control. These units retail for $44.95. Now buy them for $7 each! Call (616) 982-3838 EST.

FOR SALE: SEAGATE 4051 40 meg. HARD DRIVE $275, MITSUBISHI 4851 FLOPPIES for parts 3/$12, Z-151 POWER SUPPLIES $25 ea., 80287-6 COPROCESSOR $130, Z-171 and Z-181 MODEMS from $20, Z-100 VIDEO BOARDS $20 ea. Call (616) 849-1800, Surplus Trading Co., 2700 N. M63, P.O. Box 1082, Benton Harbor, MI 49022.

THIS COULD BE YOUR AD! For as low as $5.00, you could advertise personal items in this space. And for as low as $35.00, you could advertise commercial items. People read these ads, you did!

Announcement!

HUG MEMBERS ONLY!!

The HUG-386 and HUG-386-C upgrade kits will be available shortly. Wheelin' Dealin' Jim has managed a super-fantastic deal on these two products for Heath Users' Group members who originally purchased an H-241 or H-248; ***one-thousand two-hundred dollars*** off the regular purchase price! That's right! If you originally purchased an H-241 or H-248, and you're a HUG member, you can get $1200.00 off the regular retail price of either of these two upgrade kits!

The HUG-386 and HUG-386-C are upgrade kits that let you upgrade your H-241 or H-248 series computers up to a full H-386. Now, how do you determine which upgrade kit to buy? The H-386-C includes a dual winchester/floppy controller, while the H-386 does not include any disk controller. Since the old H-241 controller is not '386 compatible, you'll probably want the "C" model if you're upgrading a '241. If you're upgrading a '248, your decision will depend on whether you need a new dual controller or not.

Here are the three ways you can order your upgrade:

Write-In Orders

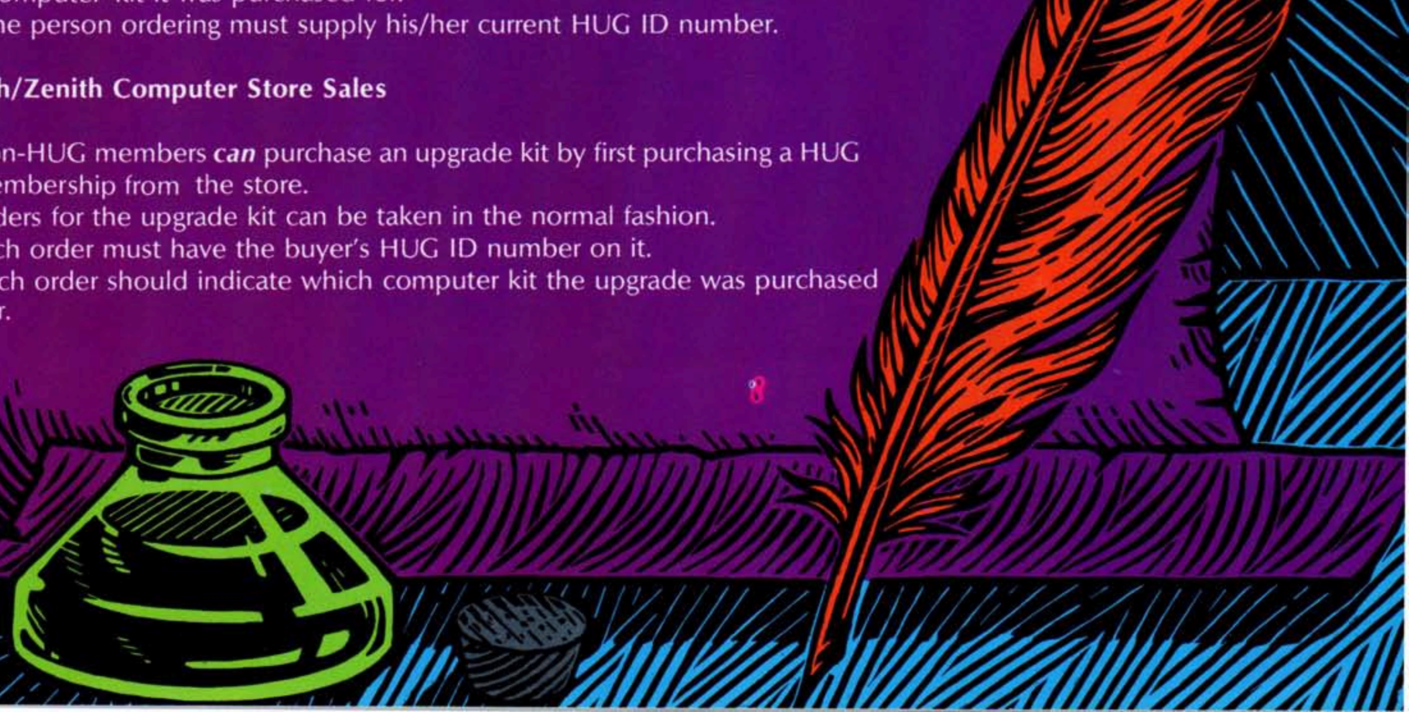
- Non-HUG members ***can*** order by including payment (with the upgrade kit order) for one year's membership in the Heath Users' Group.
- All orders should be submitted to the Heath Users' Group.
- Each order must indicate the model number of the upgrade kit desired, and which computer kit it was purchased for.
- Each order must have the persons HUG ID number written on it.

Phone-In Orders

- Non-HUG members ***can*** order by first ordering a one year's membership in the Heath Users' Group.
- All orders must be phoned in to (616) 982-3838 from 8 AM to 4:30 PM EST.
- Each order must indicate the model number of the upgrade kit desired, and which computer kit it was purchased for.
- The person ordering must supply his/her current HUG ID number.

Heath/Zenith Computer Store Sales

- Non-HUG members ***can*** purchase an upgrade kit by first purchasing a HUG membership from the store.
- Orders for the upgrade kit can be taken in the normal fashion.
- Each order must have the buyer's HUG ID number on it.
- Each order should indicate which computer kit the upgrade was purchased for.



Don't be left alone with your Heath/Zenith Computer, join HUG today!!

Heath/ZENITH
Users'
Group

P.O. Box 217
Benton Harbor, MI 49022-0217