



REMark®

Volume 8, Issue 4 • April 1987

P/N 885-2087 Issue 87

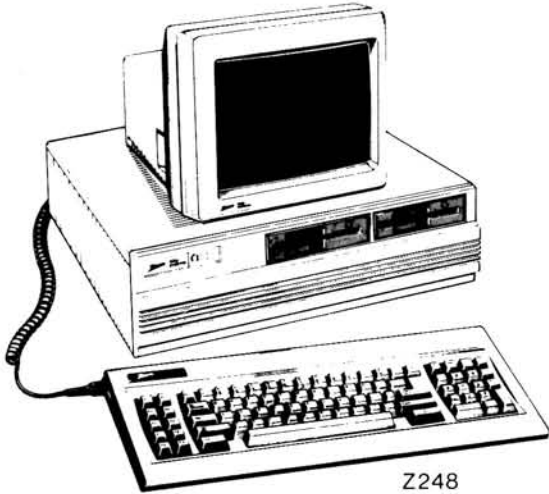
Official magazine for users of **HEATH ZENITH** computer equipment.

Free RAM For Your PC-Compatible?!
See Page 45.

\$2.50



First Capitol Delivers Zenith. Your Way.



Z248



Z181

Specialists in
CAD Systems
Local Area Networking



Z148

Have it your way.

If you're looking for a PC, you couldn't do better than Zenith. And you couldn't buy your Zenith at a better place than First Capitol. We'll put together a system the way YOU want it. And we'll do it for a price that competes with the guy who just sells boxes that fall on his loading dock.

Great Prices. And A Great Machine.

We're committed to Zenith. And to you. the standard Zenith models that come from the factory are a good buy. And we'll be glad to sell you one at a price that will match anyone. But if you want the full potential, we'll fix it up for you. Larger disk drives. More memory. Internal tape back up. Removable winchesters. All at a budget pleasing price.

We make Zenith Scream.

When a new model comes out, we're among the first to get it. Our Zenith fanatics take it apart, prod it and poke it until it screams for mercy. Then we put it back together (usually with some new parts) so that it REALLY screams. With power.

You'll be back.

Check with us. Check elsewhere. We'll put together a custom quotation for you we GUARANTEE you'll find interesting. We think you'll come back to us. There's a reason for the growing list of companies, government agencies, and end users that know us as the source for Zenith. Your Way.



1106 First Capitol Dr.
St. Charles, MO 63301

1-800-TO-BUY-IT (800-862-8948) Orders and quotes

1-314-724-2336 Technical support and order status

The StockSystem Challenge.

We will meet or beat any dealer's price on new stock Zenith systems. And for enhanced systems, no one can touch our combination of value, performance, and custom service.

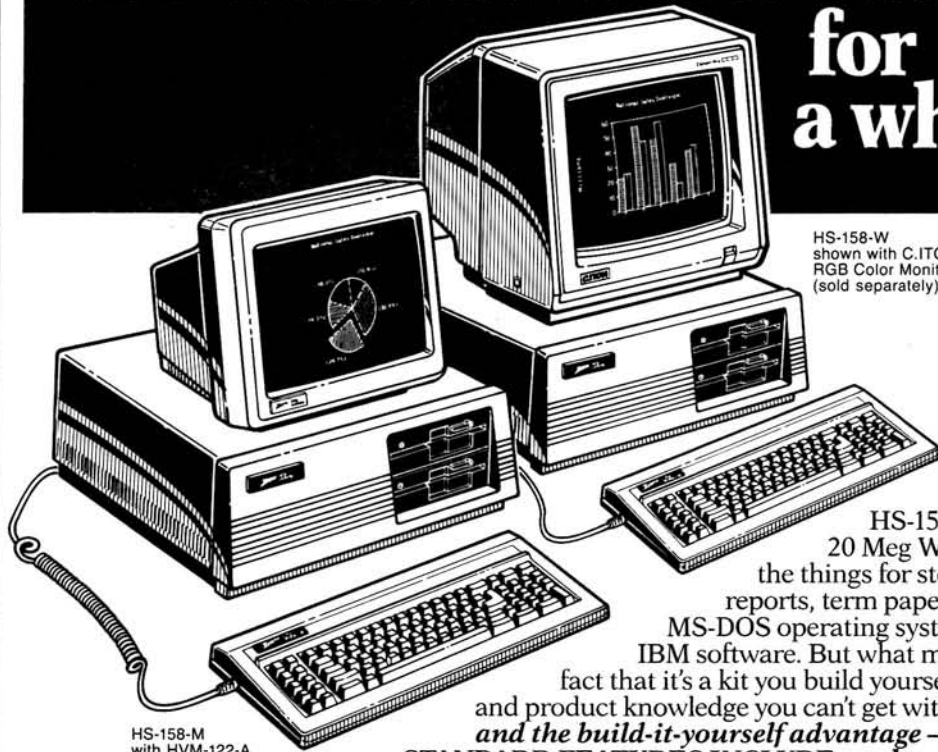
Before you buy

ZENITH | data systems

AUTHORIZED SALES AND SERVICE

Call Us First. And Last.

A whole lotta meg for a whole lot less



HS-158-M
with HVM-122-A
Monitor

HS-158-W
shown with C.I.TOH
RGB Color Monitor
(sold separately)

The IBM PC compatible kit computer with memory to spare

Having enough memory and storage for your business, education, or home use is important. And the Heath/Zenith

HS-158 was designed with that in mind. Its 20 Meg Winchester and 640K memory are just the things for storing volumes of important financial reports, term papers, and budgets. The HS-158 uses the MS-DOS operating system and runs most industry-standard IBM software. But what makes this PC even more special is the fact that it's a kit you build yourself. For that pride of accomplishment and product knowledge you can't get with other PCs. **More memory/storage and the build-it-yourself advantage – a combination that can't be beat.**

STANDARD FEATURES INCLUDE: • color and mono support • serial/parallel port • 6 expansion slots • 5 or 8 Meg speed switch • PC compatibility

HS-158-M with FREE HVM-122-A monochrome monitor, one 5¼" drive, 640K RAM, diagnostic software and DOS **ONLY \$1379***

HS-158-W with one 5¼" drive, 640K RAM, 20 Meg Winchester with controller, diagnostics and DOS. (Monitor not included.) C.I.TOH RGB Color Monitor **ONLY \$369** **ONLY \$1579***

*HUG members get 10% off

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

PHOENIX, AZ
2727 W. Indian School Rd.
602-279-6247

TUCSON, AZ
7109 E. Broadway
602-885-6773

ANAHEIM, CA
330 E. Ball Rd.
714-776-9420

CAMPBELL, CA
2350 S. Bascom Ave.
408-377-8920

EL CERRITO, CA
6000 Potrero Ave.
415-236-8870

LA MESA, CA
8363 Center Dr.
619-461-0110

LOS ANGELES, CA
2309 S. Flower St.
213-749-0261

POMONA, CA
1555 N. Orange Grove Ave.
714-623-3543

REDWOOD CITY, CA
2001 Middlefield Rd.
415-365-8155

SACRAMENTO, CA
1860 Fulton Ave.
916-486-1575

WOODLAND HILLS, CA
22504 Ventura Blvd.
818-883-0531

WESTMINSTER, CO
8725 Sheridan Blvd.
303-429-2292

JACKSONVILLE, FL
8262 Arlington Expressway
904-725-4554

HIALEAH, FL
4705 W. 16th Ave.
305-823-2280

PLANTATION, FL
7173 W. Broward Blvd.
305-791-7300

TAMPA, FL
4019 W. Hillsborough Ave.
813-886-2541

ATLANTA, GA
5285 Roswell Rd.
404-252-4341

HONOLULU, HI
98-1254 Kaahumanu St.
808-487-0029

CHICAGO, IL
3466 W. Devon Ave.
312-583-3920

DOWNERS GROVE, IL
224 Ogden Ave.
312-852-1304

INDIANAPOLIS, IN
2112 E. 62nd St.
317-257-4321

MISSION, KS
5960 Lamar Ave.
913-362-4486

LOUISVILLE, KY
12401 Shelbyville Rd.
502-245-7811

KENNER, LA
1900 Veterans Memorial Hwy.
504-467-6321

BALTIMORE, MD
1713 E. Joppa Rd.
301-661-4446

ROCKVILLE, MD
5542 Nicholson Lane
301-881-5420

PEABODY, MA
242 Andover St. (Rt. 114)
617-531-9330

WELLESLEY, MA
165 Worcester St. (Rt. 9)
617-237-1510

FARMINGTON HILLS, MI
29433 Orchard Lake Road
313-553-4171

EAST DETROIT, MI
18149 E. Eight Mile Rd.
313-772-0416

ST. JOSEPH, MI
2987 Lake Shore Drive
616-982-3215

HOPKINS, MN
101 Shady Oak Rd.
612-938-6371

ST. PAUL, MN
1645 White Bear Ave.
612-778-1211

BRIDGETON, MO
3794 McKelvey Rd.
301-661-1850

OMAHA, NE
9207 Maple St.
402-391-2071

OCEAN, NJ
1013 State Hwy. 35
201-775-1231

FAIR LAWN, NJ
35-07 Broadway (Rt. 4)
201-791-6935

AMHERST, NY
3476 Sheridan Dr.
716-835-3090

JERICHO, LI
15 Jericho Turnpike
516-334-8181

ROCHESTER, NY
937 Jefferson Rd.
716-424-2560

N. WHITE PLAINS, NY
7 Reservoir Rd.
914-761-7690

GREENSBORO, NC
4620C W. Market St.
919-299-5390

CINCINNATI, OH
131 West Kemper Rd.
513-671-1115

CLEVELAND, OH
28100 Chagrin Blvd.
216-292-7553

COLUMBUS, OH
2500 Morse Rd.
614-475-7200

TOLEDO, OH
48 S. Byrne Rd.
419-537-1887

OKLAHOMA CITY, OK
7409 South Western Dr.
405-832-6418

FRAZER, PA
630 Lancaster Pike (Rt. 30)
215-647-5555

PHILADELPHIA, PA
7438 Roosevelt Blvd.
215-268-0180

PITTSBURGH, PA
3482 Wm. Penn Hwy.
412-824-3564

WARWICK, RI
558 Greenwich Ave.
401-738-5150

DALLAS, TX
12022C Garland Rd.
214-327-4835

FORT WORTH, TX
6825-A Green Oaks Rd.
817-737-8822

HOUSTON, TX
1704 W. Loop N.
713-869-5263

SAN ANTONIO, TX
7111 Blanco Rd.
512-341-8876

MIDVALE, UT
58 East 7200 South
801-566-4626

ALEXANDRIA, VA
6201 Richmond Hwy.
703-765-5515

VIRGINIA BEACH, VA
1055 Independence Blvd.
804-460-0997

SEATTLE, WA
505 8th Ave. N.
206-682-2172

VANCOUVER, WA
516 S.E., Chkalov Dr.
206-254-4441 – Washington

503-241-3776 – Oregon
MILWAUKEE, WI
5215 W. Fond du Lac Ave.
414-873-8250

Phone orders accepted
OFFER GOOD THROUGH
APRIL 15, 1987

Your TOTAL SERVICE computer center • Service • Support • Software • Accessories • User Training • Competitive Prices

HZC-289

Units of Veritechnology Electronics Corporation

Heath®

ZENITH®

Computers & Electronics

Managing Editor Jim Buszkiewicz
(616) 982-3837

Software Engineer Pat Swayne
(616) 982-3463

Software Coordinator Nancy Strunk
(616) 982-3838

Production Coordinator Lori Lerch
(616) 982-3794

Secretary Margaret Bacon
(616) 982-3463

HUG Bulletin Board (616) 982-3956

Contributing Editor William 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
Hilltop Road
St. Joseph, MI 49085
(616) 982-3463

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

Articles submitted by users and published in REMark, which describe hardware modifications, are not supported by 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.

Copyright (C) 1987, Heath/Zenith Users' Group



Buggin' HUG

..... 7

Heath/Zenith (Almost) Related Products

Theodore (Ted) Borlowinski 11

The Russian Language On The H-100 And The MX-80 Printer

Michael Scott 13

The Mouse And Turbo Pascal

M.D. Holmberg II 17

Two Terminal Games!!!

Steven W. Vagts 27

A Software Clock/Calendar — CP/M-80 Part 2

Rodney Horner 33

ZPC Update #15

Pat Swayne 39

Mainstream Computing

Joseph Katz 45

A Winchester For The '89 Part Eleven
Peter Ruber 48

H-150 Speed-up Modification Update
Dante Bencivengo 51

HUG Price List
 54

HUG New Products
 55

Printing The Contents Of The Screen On Demand
Bill Rothman 57

C__Power — Part 2
John P. Lewis 71

The Jewel Of The ^F/_X Mile
Jim Buszkiewicz 75

ASCII — We See It Mentioned Often, But What Is It?
Kenneth Mortimer, PE 77

OK Pardner . . . DRAW!
Jim Buszkiewicz 82

Index of Advertisers

This index is provided as an additional service. The publisher does not assume any liability for errors or omissions.

| Reader Service No. | | Page No. |
|--------------------|--------------------------------|----------|
| 101 | American Cryptronics | 74 |
| 102 | Analytical Products | 37 |
| 127 | Bersearch Information Services | 24 |
| 104 | FBE Research Company | 74 |
| 139 | FLR Digital | 43 |
| 105 | First Capital Computer | 2 |
| 129 | Generic Computer Products | 79 |
| 107 | Paul F. Herman | 12 |
| 108 | Hogware | 79 |
| 109 | Hughes Development Systems | 16 |
| 110 | InterContinental Microsystems | 10 |
| 137 | Jay Gold Software | 81 |
| 111 | KEA Systems Ltd. | 74 |
| 112 | Kalltronics | 43 |
| 136 | Lindley Systems | 81 |
| 114 | Micronics Technology | 24 |
| 116 | PMI | 56 |
| 117 | Payload Computer Services | 44 |
| 118 | Rybs Electronics, Inc. | 83 |
| 119 | S&K Technology, Inc. | 12 |
| 120 | Santa Cruz Operations | 6 |
| 121 | Scottie Systems | 12 |
| 122 | Secured Computer Systems | 43 |
| 123 | Software Appl. of Wichita | 50 |
| 105 | Software Wizardry | 38 |
| 124 | Spectre Technologies Inc. | 32 |
| 131 | Veritechnology Elec. Corp. | 3 |
| 138 | X-10 Powerhouse | 84 |

On The Cover: APRIL FOOLS!!! For an explanation of these bazaar products, see the "Heath/Zenith Almost Related Products" on Page 11. Photos by Dan Wilson, Heath Graphics Design.



IN MULTIUSER COUNTRY, ALL ROADS LEAD TO SCO

SCO XENIX™ SYSTEM V

In the heartland of multiuser country, you're licensed for up to 16 users on IBM® PC AT® or PC XT®, respective compatibles, or AT&T PC 6300 Plus!

Discover unprecedented price performance per user.

Enjoy Multiscreen™ vistas in breathtaking **setcolor**. Explore DOS cross-development. Protect your investment with 8086/80286 and XENIX 3.0 applications compatibility. And bring along your widest range of peripherals!

SCO XENIX-NET XENIX-DOS LAN

Travel through SCO XENIX-NET and witness the future of office computing: networked DOS *and* XENIX PC workstations. Visit a unified environment that is both "multiuser" *and* "LAN."

Watch XENIX and DOS systems serve each other and share resources — each doing what it does best — without sacrificing the strengths of either.

SCO uniPATH™ SNA 3270

Gateway to mainframe territory, SCO uniPATH is a must-see for both VARs and MIS/DP managers. Experience the real power of PCs in corporate distributed processing.

Build customized applications with programmatic interfaces, and let your networked multiuser PCs process mainframe data — with session hold and multiple sessions per user, of course.

HOT-LINE SUPPORT DOCUMENTATION USER TRAINING NEXT EXIT

SCO's multiuser traveler's services are unrivaled, and start with SoftCare™ Support, including a toll-free hotline! You can even enjoy 30 days of Introductory SoftCare with the purchase of each product — on us! SCO's excellent documentation is considered a standard for multiuser road guides. And a wide variety of training courses is available for all SCO products.

SCO applications available for all popular UNIX/XENIX systems.

XENIX and Multiplan are registered trademarks of Microsoft Corporation. • IBM, AT and XT are registered trademarks of International Business Machines Corporation. • Multiscreen, SCO Professional and Lyrinx are trademarks and SoftCare is a service mark of The Santa Cruz Operation, Inc. • 1-2-3 and Lotus are registered trademarks of Lotus Development Corporation. • dBASE II is a trademark of Ashton-Tate. • uniPATH is a trademark of Pathway Design. • UNIX is a trademark of AT&T Bell Laboratories. • FoxBASE is a trademark of Fox Software, Inc. • Informix is a registered trademark of Relational Database Systems, Inc. 6/86

©1986 The Santa Cruz Operation, Inc.

The Santa Cruz Operation, Inc., P.O. Box 1900, Santa Cruz, CA 95061

SCO Professional™ 1-2-3 WORKALIKE

Take this exit, and you'll swear you're on Route 1-2-3® — until you notice enhanced features and multiuser access that

Lotus® users only dream about! But don't worry about culture shock — if you like 1-2-3, you'll *love* SCO Pro. Read and write to 1-2-3 DOS files, enjoy full 1-2-3 functionality and macro support, and rest easy with file locking data security.

SCO FoxBASE™ dBASE WORKALIKE

As with SCO Professional, another *deja vu* awaits you on this route — this time, dBASE II™ — but once again, it's better, faster, and multiuser!

Bring along your dBASE II data files, macros, and expertise — they all work here, too! You'll be sharing dBASE II-compatible data with friends, in record and file locking comfort, right out of the box.

SCO Lyrinx™ WORD PROCESSING

Built especially for UNIX™, "what you see is what you get" SCO Lyrinx is a favorite of both new and veteran multiuser travelers. Fully user-configurable, it's a natural attraction to foreign

language visitors, as well as system administrators. You'll appreciate advanced WP features such as spelling corrector, and automatic footnotes and section numbering, as well as direct menu access to UNIX and other destinations.

SCO SALES - INFO 1-800-626-UNIX

Call us today to order any of these and other exciting SCO products such as Multiplan®, Informix®, languages, and business graphics. VARs, ISVs and other qualified customers can "mix

and match" for cumulative discounts across the entire SCO product line. We look forward to booking your next multiuser tour!

SCO
THE SANTA CRUZ OPERATION

(800) 626-8649

(408) 425-7222

TWX: 910-598-4510 SCO SACZ

uucp: ...decvax!microsoft!sco!info

BUGGIN' HUG

Reference Dan Jerome, John Toscano Letter

Dear HUG:

The letter from Dan Jerome and John Toscano in Buggin' HUG in the October 1986 Issue of REMark was just another example of how very important and useful information gets buried over the years and some of us just assume that others know all about it just because we have been doing it for years.

I would like to add a very important bit of information to their letter. H37 Extended Double Density disks can only be formatted with the FORMAT.COM Version 2.03 which came with H/Z CP/M 2.2.03. The earlier version of CP/M 2.2.02 did not support H37 soft sector disks of any type. The later and final release of CP/M Version 2.2.04 had Version 2.04 of FORMAT.COM. This version does not support formatting of H37 Extended Double Density disks. The only version of FORMAT.COM which does support Extended Double Density disks is 2.03. As Dan and John stated in their letter, the option is available even though it is not shown in the selection menu — it says "Which Density? (S=Single, D=Double)" — and you type "E" for Extended Double.

If you would like FORMAT to display Extended Double Density in the selection menu, you can use DDT to change address 07A6H from 0D1H to 094H as follows:

```
DDT FORMAT.COM
NEXT PC
1780 0100
- S7A6
  7A6 D1 94
  7A7 13
- ^C
```

A> SAVE 22 FORMAT.COM

Be sure to apply this patch to FORMAT version 2.03 only and only if you have a backup copy in case you make a mistake. Dan and John are correct regarding the reliability of Extended Double Density. I have been using it for over two years with no problems. Actually, the disk data transfer speed is also improved since as they stated, there is less housekeeping overhead with 5 sectors per track instead of 16.

I have included a document I published last year in the Connecticut Heath Users' Group Newsletter which outlines all the disk formats and capacities for the H17/H37.

This brings me to a related topic. Since Heath/Zenith 8-bit computers are now out of production and hardly any old and no new 8-bit software is being marketed by H/Z (HUG excluded), why doesn't H/Z release to HUG the source code to the 8-bit programs such as FORMAT, CONFIGUR, SETLIP, ASSIGN, BRS, MAKEBIOS, and LIST which came with CP/M. These programs have no proprietary Digital Research code in them and H/Z could legally release them. Programs like SYSGEN and MOVCPM, even though heavily customized by H/Z, have Digital Research code and could not be released by H/Z without their permission (not likely to happen). This would allow the existing 8-bit user base to modify, customize and enhance these programs with the potential of the

enhancements submitted back to HUG for redistribution. I couldn't think of a better way for H/Z to give its final farewell to old 8-bit products at no cost to themselves.

Keep the 8-bit articles coming. I do not plan on replacing (notice I did not say "upgrading") my H-89, probably ever. I do realize that it may have to eventually "co-exist" with a PC clone of some type. But for now, I can do word processing, spreadsheets, database management, and telephone communications. I can compile BASIC, Pascal, C, Forth, Cobol, Algol, Lisp, and Prolog programs, as well as assemble 8080, Z80, 6800 and 6502 assembly language programs. I can program EPROMS and play music and have an occasional game of chess. Yea, yea, I can't run LOTUS or Flight Simulator . . . but the PCs can't run Z-System! Very best wishes!

Rick Swenton
19 Allen Street
Bristol, CT 06010

Heath/Zenith H37 - H17 Disk Formats Under CP/M

Copyright (C) 1985 by Rick Swenton

Below is a table of possible 5-1/4 inch disk formats and usable storage capacity of disks used on the Heath H37 and H17 controllers under CP/M. The storage capacity shown is the amount available to the user, after formatting, to store files. The real storage capacity is higher but not available to the user since tracks are reserved for the CP/M operating system and disk directory.

(To format Extended Double Density, you must use FORMAT .COM Version 2.03. When it asks 'S-Single, D-Double', type 'E' for Extended. This is an undocumented and unsupported feature. FORMAT.COM Version 2.04 does not support the 'E' option.)

| Drive Types | Sides/Density | Storage Capacity | Typical Models |
|-------------|---------------|------------------|---|
| 96 tpi | DSED | 782 k | TM-100-4,SA-465,55-F |
| 96 tpi | DSDD | 624 k | TM-100-4,SA-465,55-F |
| 96 tpi | DSSD | 388 k | TM-100-4,SA-465,55-F |
| 96 tpi | SSED | 386 k | (It's not too common to find a single sided 96 tpi drive) |
| 96 tpi | SSDD | 308 k | |
| 96 tpi | SSSD | 190 k | |
| 48 tpi | DSED | 382 k | TM-100-2,SA-455,55-B |
| 48 tpi | DSDD | 304 k | TM-100-2,SA-455,55-B |
| 48 tpi | DSSD | 188 k | TM-100-2,SA-455,55-B |
| 48 tpi | SSED | 186 k | TM-100-1,FDD-100-5,55-A |
| 48 tpi | SSDD | 148 k | TM-100-1,FDD-100-5,55-A |
| 48 tpi | SSSD | 90 k | TM-100-1,FDD-100-5,55-A |
| 96 tpi | DSSD Hard * | 390 k | TM-100-4,SA-465,55-F |
| 96 tpi | SSSD Hard * | 190 k | (Not too common) |
| 48 tpi | DSSD Hard * | 190 k | TM-100-2,SA-455,55-B |
| 48 tpi | SSSD Hard * | 90 k | TM-100-1,FDD-100-5,55-A |

* These formats require Livingston Logic Labs BIOS-80 which is a replacement BIOS.SYS and configuration utility. It includes patches to the FORMAT.COM program to support formatting the extended H17 Hard Sector disks.

Legend: DSED - Double Sided Extended Density
DSDD - Double Sided Double Density
DSSD - Double Sided Single Density
SSED - Single Sided Extended Density
SSDD - Single Sided Double Density
SSSD - Single Sided Single Density

"Cheap Speed"

Dear HUG:

In response to the article "Cheap Speed" by Pat Swayne in the December '86 Issue of REMark, I wish to add information on the modification of the Z-151 power supply and his reversing of the fan.

I also do not like to have dirt accumulation inside of my computers. However, rather than voiding any warranty on the power supply of the Z-151, I have instead added a thin sheet of foam commonly sold to filter return air in window type air conditioners in the plenum chamber in front of the fan inside of the case. This area is very good and fairly air-tight. A piece of foam is readily cut with scissors to fit in the area.

I made temperature studies with a remote probe to determine if any noticeable rise in internal temperature resulted from the filter installation. There was a one degree increase in the air temperature in the slot area below the CPU card with the filter installed. The temperature did not exceed 100 degrees F at any time with the computer running. My Z-151 is equipped with the Mega-RAM and has the full complement of 256k chips installed. It also supports an Adaptec RLL controller and a Tulin 30 meg drive with formatted 45 meg capacity as drives D: and E:. It also contains a Copy II PC option board, as well as a Everex modem. It runs most of the time from a DataShield power backup. There has been absolutely no temperature instability or difficulty with the computer. Also, there is no dust or grime inside of the unit!!

I believe that Pat does a great job on most of his articles. I realize that the 151 is out of warranty at this time due to age; however, I feel that taking the power supply apart is overkill when a foam filter is well within anyone's ability and does not harm the computer in any manner.

Signed,

Robert E. Clark, M.D.
305 South Sixth Street
Memphis, TX 79245

Simplifying Automation Of CONFIGUR

Dear HUG:

Creation of the files WSSETUP.DAT and WSRETURN.DAT mentioned in Charles F. Ballinger's article, "Automating The Configur Feature" (REMark, October 1986) can be simplified using BASICA. In fact, if you have a hard disk with GW-BASIC or PC ADVANCED BASIC, Version 2 or greater on it, you may prefer to use WSSETUP.BAS (Listing 1) and WSRETURN.BAS (Listing 2). You will need to replace the commands "CONFIGUR <WSSETUP.DAT" and "CONFIGUR <WSRETURN.DAT" in WSLETTER.BAT with "BASICA WSSETUP and "BASICA WSRETURN", respectively.

Listing 1

```
0 'TITLE: wssetup.bas VERSION: 1.00 DATE: 10-9-86
  AUTHOR: J. E. Hill
100 OPEN"o",1,"wssetup.dat"
110 PRINT#1,"AJB";CHR$(13);CHR$(13);CHR$(13);CHR$(13);
    "AIDC";CHR$(13);"D";CHR$(13);CHR$(13);CHR$(13);
    "GE";
120 CLOSE#1
130 SHELL"configur <wssetup.dat >nul"
140 KILL"wssetup.dat"
150 SYSTEM
```

Listing 2

```
0 'TITLE: wsreturn.bas VERSION: 1.00 DATE:10-9-86
  AUTHOR: J. E. Hill
100 OPEN"o",1,"wsreturn.dat"
110 PRINT#1,"AA";CHR$(13);"GE";
120 CLOSE#1
130 SHELL"configur <wsreturn.dat >nul"
140 KILL"wsreturn.dat"
150 SYSTEM
```

If you don't have a hard disk, delete lines 130 and 140 in both WSSETUP.BAS and WSRETURN.BAS and use these routines to produce WSSETUP.DAT and WSRETURN.DAT for use with the original version of WSLETTER.BAT. Any version of GW-BASIC or PC ADVANCED BASIC can be used for this purpose, since the SHELL command is not required.

For those not familiar with BASICA, here is an explanation of how both WSSETUP.BAS and WSRETURN.BAS work:

| Line | Description |
|------|--|
| 0 | A comment to identify the program for future reference. |
| 100 | Opens file buffer #1 for output to WSSETUP.DAT (WSRETURN.DAT) |
| 110 | "Prints" the necessary characters to WSSETUP.DAT (WSRETURN.DAT). When BASICA reads "AJB", "AIDC", "D", and "GE", it outputs the ASCII characters corresponding to the letters within the quotes. When BASICA reads each CHR\$(13), it outputs the ASCII character for carriage return. The semicolons between entries tell BASICA other characters will follow and that no characters are to be outputted in between. The semicolon at the end of the line tells BASICA not to output a carriage return at the end of the PRINT statement. |
| 120 | Closes file buffer #1 and file WSSETUP.DAT (WSRETURN.DAT). |
| 130 | Calls DOS to execute the command within the quotes. Note the >nul. This eliminates the CONFIGUR.COM output that normally goes to the screen. |
| 140 | Deletes WSSETUP.DAT (WSRETURN.DAT). There is no need to keep these files on disk since they are created each time WSSETUP.BAS and WSRETURN.BAS are executed. |
| 150 | Returns control to batch routine. |

It should be noted that WSLETTER.BAT assumes all files or programs required are in the current directory. If this is not the case, before running WSLETTER.BAT, log onto the directory that contains WSSETUP.DAT (or WSSETUP.BAS) and WSRETURN.DAT (or WSRETURN.BAS) and ensure that WSLETTER.BAT and CONFIGUR.COM (and BASICA.EXE or .COM if used) are either in the current directory or in directories included in PATH. Good luck!

Sincerely,

John E. Hill
1117 Haral Place
Cherry Hill, NJ 08034

"Faster Disk Access"

Dear Mr. Swayne:

This letter responds to your short article, "Faster Disk Access", contained in the November 1986 Issue of REMark magazine.

I appreciate direct, hard-hitting articles of this type. Not being your most sophisticated programmer, I was unable to use the information on Faster Disk Access provided in your August Issue of REMark. However, I was able to install "FASTDSK" within 20 minutes, and have noticed significant improvement in the loading and operation of all my programs.

I evaluated your "FASTDSK" program; the minimum improvement in loading time that I have experienced is 12%.

These 'evaluations' (I can't really call them tests, as my equipment is not set up to run technical tests) were conducted on an H/Z-151-2, built from a kit in January 1985. It now has 640 kb of main memory available (704 kb total; I left the 64 kb chips in the 5th bank when I upgraded to 256 kb chips on banks 3 & 4), and runs a NEC V-20 8088-type processor. In this configuration, I achieve the capability of 1.7 times that of your vanilla "Big Blue" PC (NORTON UTILITIES, SI program). As tested, approximately 40.0 kb was allocated to MS-DOS V2.11 Update (Zenith version), 252 kb was allocated to SIDEKICK. This left 293.2 kb for running programs and applications.

The results were:

| w/o FASTDSK LOAD/EXIT secs | APPLICATION | w/FASTDSK LOAD/EXIT | % CHANGE |
|-------------------------------|-----------------|------------------------|-------------|
| 47.6/11.2 | WORD v3.1 | 33.6/8.2 | -29%/-27% |
| 22.9/9.9 | MULTIPLAN v2.01 | 15.9/7.9 | -31%/-20% |
| 33.7/6.1 | PROJECT v2.01 | 29.9/5.0 | -12%/-18% |
| 32.1/3.2 | CHART v2.02 | 21.6/1.6 | -33%/-50% |
| 26.4/3.4 | SUPERCALC 4 | 16.4/2.4 | -38%/-30% |
| 13.4/1.4 | NORTON UTIL | 10.2/.7 | -24%/-50% |
| 38.6/5.5 | BPI PERS ACCTG | 31.2/4.3 | -20%/-22% |
| 11.1/5.8 | "BRIDGE" | 7.3/3.8 | -34%/-34% |

Not bad, for a simple program that took such a short time to create and load!

I suppose that this letter is also a plea for a few more articles which address the "PC-clones" and the associated software and hardware to upgrade PC performance to as close to "AT" capabilities as possible. My personal objective is to extend the life of this machine (H/Z-151) as long as possible, due to the money that it cost me. Given that there are many people like me — not sophisticated programmers and without access to compilers — it appears reasonable to ask that you consider increasing slightly the number of articles which address either purchaseable software/hardware or provide simple, runtime code for applications which upgrade performance.

Thanks for the "FASTDSK" program. It is now loaded as a standard application for my entire family. Do you have any other "good deals" like this, which will speed my computing activities? I would very much appreciate hearing of some.

Yours truly,

Richard C. Fairlamb
6209 Ponderosa
Colleyville, TX 76034

REMark, June 1986, D. Bencivengo

Dear HUG:

Recently, I assembled the H-150 Speed-up Modification, REMark, June 1986, D. Bencivengo. I was excited about the prospect of increasing the speed of my H-151 computer. The modification worked exactly as written in the June REMark magazine. I was un-

happy with the prospect of adding a switch to the front panel of my H-151. I also found myself using the [Ctrl]-[Alt]-[Del] key combination to re-boot the computer, only to encounter the "Timer Interrupt Failure". I began investigating possible solutions to these problems.

Evidently, the MFM-150 ROM tests the system timer during the boot-up procedure. The CPU programs the system timer to interrupt the processor after a certain time delay, the CPU then executes a software timing loop, and waits for the system timer interrupt. When the CPU is running at a speed faster than 4.77 MHz, the CPU completes the software timing loop before the system timer provides the required interrupt, thus the CPU reports a "Timer Interrupt Failure".

Normally, resolution of this problem would require modification of the MFM-150 ROM program. There are several ways to work around this problem. The circuit designed by D. Bencivengo uses an R-C timer network to ensure the H-151 system timer runs at 7.38 MHz for the first few seconds after the hardware RESET. During this time, the MFM-150 ROM completes the system timer test. After the system timer passes the test, the H-150 Speed-up circuit, returns the system timer to the normal 4.77 MHz, this is important to ensure accuracy of the DOS clock. Since this R-C timer requires a hardware RESET to initialize the circuit, the [Ctrl]-[Alt]-[Del] key combination, which does not provide a hardware RESET, will not allow the CPU and the system timer to start at the same speed.

The H-151 is provided with diagnostic LEDs on the CPU board, which indicate the status of the system self-test. By observing these LEDs during the boot-up, it is evident that the MFM-150 ROM uses these LEDs to indicate the status of the self-test anytime the MFM-150 ROM is testing the computer. Specifically, the "INT" lamp indicates when the system timer is being tested. By using the "INT" lamp to control the speed of the H-150 Speed-up circuit, the problem with the [Ctrl]-[Alt]-[Del] reboot is solved.

The "INT" LED is driven by integrated circuit U200 on the CPU board (U201 on some versions of the CPU board). This Integrated circuit is a write only port used exclusively to drive the diagnostic LEDs. Therefore, any software running on the computer is unlikely to write to this port, and software cannot read the status of the LEDs. Therefore, a short program can be written to change the status of the "INT" LED, and thus the speed of the CPU without any worry of interfering with other software which may be used on the computer. Following are the details of the necessary modification to the H-150 CPU Speed-up circuit (refer to REMark, June 1986).

The MFM-150 ROM uses the diagnostic LEDs to report the self-test status of the machine. By coupling the speed switch to these LEDs, the MFM-150 will shift into slower speed whenever the Timer interrupt is being tested. Figures 1 and 2 show the schematic and the circuit layout diagram for the modified H-150 Speed-up board. The modification to the H-150 CPU Speed-up circuit board is as follows:

1. Remove R3, R4, CR1, R5, and C3.
2. Cut the following circuit runs (see Figure 2):
 - a. From U3-10 to U3-5
 - b. From U3-4 to U3-9 (U3-4 still connects to U1-13)
3. Remove J1 and connect U2-13 to ground.
4. Connect U3-8 to U3-6 (This is the speed input).
5. Connect U3-9 to U3-5 to Ground.

6. Connect U3-10 to a 330 Ohm resistor.
7. Connect the other end of the 330 Ohm resistor to a wire to the cathode of an LED (This is the Turbo Speed indicator).
8. Connect the Anode of the Turbo Speed indicator LED to +5 Volts (P2 on the H-150 CPU Speed-up board).
9. Using a piece of 22ga wire, carefully solder one end to the cathode of CR4 (the "INT" LED system CPU board). Connect the other end of the wire to the speed input (P1 on the H-150 CPU Speed-up board).

The Speed indicator LED will light when the CPU is running at the fast speed. This LED may be mounted in the empty space below the disk drives with a drop of super glue with no permanent modification to the computer case. A rectangular style LED works particularly well.

To test the circuit modification, boot-up the computer, the Speed indicator LED should remain off for about two seconds, then the LED should come on indicating the CPU is running at fast speed. The Speed indicator should be illuminated (on) when the "INT" LED on the CPU board is off and vice versa. Now try the [Ctrl]-[Alt]-[Del].

The Turbo board will now shift into lower gear when the interrupts are being tested by the MFM-150 (including the notorious system timer interrupt) and shift into fast speed after the test is complete. The hardware RESET button is optional, depending on your needs.

Now to control the speed of the computer from software. The diagnostic LEDs are controlled by port 00C0 hex. Specifically, Bit 08 controls the "INT" LED and thus the CPU speed. If Bit 08 is set to a "1", the "INT" LED is off and the CPU runs at fast speed. If Bit 08 is reset to "0", the "INT" LED is on, and the CPU runs at 4.77 MHz.

For those people familiar with assembly language programming, the following is the assembly language for two short programs; SLOW.COM, which sets the CPU to run at 4.77 MHz, and FAST.COM, which sets the CPU to run at fast speed.

FAST.COM

CODE SEGMENT

ASSUME CS:CODE

TITLE ASSEMBLY LANGUAGE PROGRAM FOR SETTING
FAST CPU SPEED (FAST.COM)

ORG 100H

```
START: MOV AX,4CFHh
      OUT 0C0h,AL;OUTPUT FF TO PORT C0 hex
      INT 21H ;FUNCTION CODE 4C (TERMINATE)
```

CODE ENDS

END START

SLOW.COM

CODE SEGMENT

ASSUME CS:CODE

TITLE ASSEMBLY LANGUAGE PROGRAM FOR SETTING
SLOW CPU SPEED (SLOW.COM)

ORG 100H

```
START: MOV AX,4CF7h
      OUT 0C0h,AL;OUTPUT F7 TO PORT C0 hex
      INT 21H ;FUNCTION CODE 4C (TERMINATE)
```

CODE ENDS

END START

If the reader does not have experience with assembly language programming, the programs can also be created using the MS-DOS DEBUG utility. The create the SLOW.COM program type the following items as written in *Italic* letters (x represents don't care response):

SLOW.COM

DEBUG

-NSLOW

-E100

xxxx:0100 xx.B8 xx.F7 xx.4C xx.E6 xx.C0 xx.CD xx.21

-RCX

cx 0000

:0007

-W

writing 0007 bytes

-Q

A:REN SLOW SLOW.COM

To create the FAST.COM program type the following items written in *italic* letters (x represents don't care response):

FAST.COM

DEBUG

-NFAST

-E100

xxxx:0100 xx.B8 xx.FF xx.4C xx.E6 xx.C0 xx.CD xx.21

-RCX

cx 0000

:0007

-W

writing 0007 bytes

-Q

A:REN FAST FAST.COM

Continued on Page 83

Bus Specific Networks

NEW LINK 'AGE'

Announcing "Z-100 under NOVELL" with ICM's new Node-Z100/N package linking PCs to Zenith systems.

InterContinental Microsystems introduces new connectivity for Zenith's Z-100 products, to run NOVELL Netware, NOVELL utilities and link IBM PCs and PC-compatibles. ICM's LANS-100 ARCnet controller plugs into the Zenith 100 series workstations to give you compatibility never before possible.

ICM's new Node-Z100/N package includes all the shell driver software you need. And when you link our package with your Z-100s, you can network with PCs, ATs, XTs and compatibles running under NOVELL. It can also be used with ICM PC-bus products including the CPS-PC workstation processor and LAN-CPS (PC-bus) ARCnet controller. How's that for new linkage?

Call ICM for complete details and reseller pricing.

PC, ARCNET, AND S-100 APPLICATIONS



TOMORROW'S BREAKTHROUGHS TODAY

InterContinental Microsystems
4015 Leaverton Court, Anaheim, CA 92807
Phone: (714) 630-0964
Telex: 821375 SUPPORT UD
Easylink: 62562040 BBS (714) 632-8750



ARCnet is a registered trademark of Standard Microsystems Corp. Netware is a registered trademark of NOVELL Inc. IBM PC-AT, IBM PC-XT, IBM PC and PC-DOS are registered trademarks of International Business Machines Corp.

Heath/Zenith

Almost

Related Products

Theodore (Ted) Borlowinski

STACK-UM-HI INC. of Topuhduhheap, NY announced this month a break through in random access memory technology. With the discovery of Randomly Transmitted Memory Data (RTMD), Ride-A-RAMs (RARs) are now possible. RARs are standard 256k memory RAMs that can be stacked vertically on existing RAM chips to increase memory size. According to SUHI, "there's no limit to the amount of memory you can add to your PC, providing your ceiling is high enough, and your glue strong enough!" Installation is very simple. To allow for the RAM chips to stack higher than the top of the cabinet, the first step is to cut a large hole in the top of the computer. The RARs are then stacked on the existing RAM chips, and glued in place. SUHI's motto is, "Stop measuring your memory capacity in bytes. Use feet!". Memory capacities of 100 to 200 megafeet are not uncommon! SUHI, being the original contractors for the Sears Tower, says they've converted all their resources to the manufacture of Ride-A-RAMs, and rumors have it that SUHI's research and development labs are now completing work on the new Ride-A-ROM, which should be introduced by April 1st, 1988.



American Hot Wax Inc. of Buffalo Blaque, AZ unveiled their new V-Disk subsystem this month. According to AHW, this new method of mass storage will revolutionize the 'Winchester' world. The media itself can be written to only once, but can be read any number of times (WORA - Write Once Read Always). Since the media itself is vinyl, and quite inexpensive, this should present no problems. This media comes in three versions, the two program size, the extended two program size, and the long play archive size. The subsystem itself eliminates all possibilities of sector errors by using a visual head positioning system which is viscous damped. Multiple programs can be stacked and automatically loaded in sequential order. AHW says, "If a data error ever does occur, a quarter can be taped to the head mechanism to increase data readability. Doing so, however, will reduce media life. For more information, VICTOR-Disk brochures are available from AHW Inc, or look for the VICTOR-Disk hound symbol at your nearest Hi-Fi dealer.

Soon to be released, is HUG's Applied Instructional Recursive Business Associated Language Library, or HAIRBALL for short. According to its author, this compiled/interpretive language is for those would-be programmers that want the structure of 'C', the bit manipulating of assembly language, the wordiness of COBOL, the chaos of BASIC, and the ease of everyday speech. For PASCAL lovers, the key words 'BEGIN' and 'END' are also accepted, but ignored. For example, to make the compiler print your name 10 times, you would simply type: "PRINT MY NAME 10 TIMES". Your name, of course, would have to be configured into the system (prior to compilation) with the CONFIGUR program (also included). Again, to demonstrate its ease of use, a simple screen oriented editor can be written with the single line: "WHILE PUTTING THE CURSOR WHERE I TELL YOU DO EDITING ON THE SCREEN AND IF DONE SAVE TO DISK ELSE PRINT IT AND EXIT MAYBE". Other Un-ambiguous and ambiguous constructs include: DO-MAYBE, MAYBE-WHEN EQUAL, SOMETIMES-WHEN NOT EQUAL, WHEN EQUAL-MAYBE, and the ever famous YOU__DECIDE WHEN ELSE I__DECIDE MAYBE OR ELSE DO__END OR DO__NOT BEGIN MAYBE?. The lith-o-gram for the last construct is of course dependent upon many non-understandable factors. HAIRBALL comes on 1 Long-Play Archive V-Disk in ARChived format along with the un-ARChiving program which has also been ARChived. Once configured, the author indicates that HAIRBALL can run in as little as 50 kilofeet of RAR, and a monitor is not needed. By this time next year a version of TURBO-HAIRBALL should be available.

Also coming from HUG is Pat's new Z370 mainframe emulator software for the H/Z-100. For you die-hard IBM-370 fans, Pat has successfully achieved 99.1% compatibility between the two machines with no loss in execution speed. According to Pat, "emulating the '370 was easy compared to the PC, because the '370 comes much closer to the capabilities of the H/Z-100 than the PC does, so I really didn't have to do much work". For the few programs that aren't compatible, Pat has 479 different patch upgrades planned. Each of these lengthy patches will appear one at a time in upcoming issues of REMark, so now's the time to renew your membership for at least 40 more years.

Now from JABUM INC. of Sterile Falls, ID comes SIT-A-SCREEN. This revolutionary new monitor allows you to use your eyes for something else (or nothing at all) while using your computer. Now you can read the newspaper, drive your car, or even watch TV while composing a letter with your favorite word processor. Using the newly developed Pin Head Apply-Retract Technology, SIT-A-SCREEN is simply placed between you and your favorite easychair

Continued on Page 16 ■



IBM-PC

IN



An H/Z-100

REQUIRES 768K OF RAM

Basic Board W/HUG Software \$149

HUG ZPC V2 & UPGRADE
Both Included
An \$80 Value.

This EXCELLENT COMBINATION is used by many universities, colleges, hospitals, and the U.S. Naval Academy.

Options Available
COM 1 \$44, COM 2 \$39, Clock \$44

No Solder H/Z-100 Mod Kit \$5

8 Mhz V20's \$11
8 Mhz Interrupt Kits, \$12

CHECKS AND M.D. IN US FUNDS, VISA AND MC ACCEPTED
US ORDERS ADD \$4.00 S & H, APO & FPO ADD \$7.00 S & H
CA. RESIDENTS ADD SALES TAX

Write or Call for More Info.

Scottie Systems
2667 Cropley Ave. #123
San Jose, CA 95132
(408) 259-6226

IBM-PC is a registered trademark of the IBM Corp.
ZPC is a product of the Heath Users Group.

Z-100 and PC Compatible Graphics Software

| | | | |
|----------------------------------|---------------------------------|-----------|--|
| Z-100 and PC Compatibles | DOODLER-V Graphics Package | \$ 99.00 | |
| | with MOUSE PACK Mouse Driver | \$ 119.95 | |
| | with MOUSE PACK and LOGIMOUSE | \$ 199.95 | |
| | MOUSE PACK Mouse Driver Package | \$ 29.95 | |
| | with LOGIMOUSE | \$ 119.95 | |
| Font Library Disk for DOODLER | \$ 29.95 | | |
| Texture & Symbol Library Disk | \$ 29.95 | | |
| Logitech LOGIMOUSE C7 Mouse | \$ 99.00 | | |
| (includes PLUS package software) | | | |

| | | | |
|-----------|---------------------------------|-----------|--|
| PC's Only | Generic CADD with/Dot Plot | \$ 99.00 | |
| | with LOGIMOUSE | \$ 189.00 | |
| | IMAGE ACE II Video Capture Sys. | \$ 295.00 | |

Sub-Total
Florida Residents add 5 percent Tax

TOTAL

C.O.D. Orders accepted. Call 813-376-5457



Software Graphics Tools
3620 Amazon Drive
New Port Richey, FL 33553

S & K Technology, Inc. Quality Software for Heath/Zenith Microcomputers

For the Z100...

WatchWord® \$100.00

The ultimate in word processing with speed and power. See subscripts, superscripts, underlining, and boldface directly on the screen. Create your own fonts and special characters. Other features include centering, formatting, automatic horizontal scrolling with long lines, large file capability, split screen, macros, color, and an extensive configuration facility. See reviews in Remark (July 1985) and Sextant (Jan-Feb 1985, Sep-Oct 1985). Requires 192K RAM.

The Resident Speller™ \$100.00

Spelling checker for use with WatchWord. Checks as you type from inside WatchWord or checks a file. Includes a 50,000 word expandable dictionary. Requires 192K RAM to check a file. Requires 300K RAM to check as you type.

Demo disk for both \$ 3.00

For IBM compatibles including the Z150 and Z200 series...

PC WatchWord® (New) \$ 99.95

The ultimate in word processing for the sophisticated user. Most of the features of the Z100 version except for screen fonts. Requires 256K RAM.

PC Resident Speller™ \$ 99.95

Spelling checker for ASCII files such as those created with WatchWord, WordStar, WordPerfect, PeachText, and VolksWriter. Includes Strike. Requires 256K RAM.

Strike™ \$ 49.95

Adds as-you-type spelling checking to your word processor. Works with the word processors above and also with DisplayWrite, MultiMate and PFS:Write. Requires 100K RAM in addition to that used by your word processor.

Demo disk for Strike and The PC Resident Speller \$ 2.00

Demo disk for PC WatchWord \$ 2.00

Texas residents please add state sales tax.

S & K Technology, Inc., 4610 Spotted Oak Woods, San Antonio, Texas 78249, (512) 492-3384

The Russian Language

On The H-100

And The MX-80 Printer

Michael Scott
3554 W. 22nd Avenue
Vancouver, BC
CANADA V6S 1J3

Although this article describes my experiences working with Russian, it is also relevant to working with any foreign language which has "funny-looking" letters. Arabic might be quite a challenge, though, since you'd have to tinker deep in the innards of arcane mediumware to get the cursor to move backwards . . .

My initial reason for upgrading from an H-89 to the 100 was the attraction of having a RAM-based character set, which is totally under the user's control. Initially, I worked on producing a very simple system disk for my wife, who does translations into French. This worked like a charm, so I rolled up my sleeves and started on one for Russian.

The problem has two parts, the first is the screen display, and is well covered by Mark Aagenas' article in REMark (July 1983, p. 19). It is a matter of a few hours to type in the FONTED program he provides (written in BASIC), and to follow his instructions on how to change the ALTCHAR.SYS file to store your fonts. Of course, if you have MS-DOS 2.0 and up, you can use the font utility to do the same thing.

The Russian Language has many letters that look different, and some of those that look like English letters sound quite different. It is a good idea to draw up a table right from the start, listing all the English letters (lower case separately), then all the Russian letters to the right. For each English letter, write opposite it the Russian letter you want to appear on the screen when that key is pressed, and then mark those letters which will have to be created with FONTED, (Z-DOS users) or with FONT (MS-DOS 2 users). Such a table might look like this:

| English | Russian | Remarks |
|---------|---------|---------------------------------|
| | | (x=must be created or modified) |
| A | A | |
| a | a | |
| B | | x |

| | | | |
|-----|---|---|-----------|
| b | | | |
| C | C | x | remap S=C |
| c | c | | remap s=c |
| D | | x | |
| d | | x | |
| * | | | |
| * | | | |
| * | | | |
| V | B | | remap V=B |
| v | | x | remap v=b |
| * | | | |
| * | | | |
| etc | | | |

You will soon come up against one problem you will have to solve according to your own whims or preferences. The English letter "C" exists in Russian, and does not need to be created, however, in Russian this letter sounds like an "S". Do you want to get a Russian "C" when you type "C", or when you type "S"? I decided on the "S", which seems to suit me psychologically, but you may prefer the other arrangement. The "D" is no problem, you just have to create a new font, period. Some letters need to be both created, and to have their keys remapped.

The next problem is obviously the fact that Russian has 33 letters and English only has 26. The solution is to create Russian letters for certain rarely used characters, and I used the "!", "{", "}", "~" keys (shift value quoted) to get four of these.

One Russian letter looks like an "E" with an umlaut (two dots above). This is pronounced "yoh" in Russian, but is not usually printed with the two dots, except in childrens' books and books for foreigners, — thus we can dispense with this letter altogether, and finally the "hard sign" and the "soft sign" appear so rarely in their upper case forms that we can use only the lower case versions. For these letters I used the "&" and the "*" signs on the H-100 keyboard.

Do the project as follows, to avoid confusion:

ONE. Do all the key remapping in one session, save your redefined keys as ALTCHAR.SYS, exit FONTED (or FONT) and REBOOT to load your new definitions. Make a backup of your altchar.sys as say, RUSSIAN.CHR somewhere.

TWO. Do the font alterations separately. Save them as ALTCHAR.SYS.

THREE. Exit, reboot to load the new font, and try typing some Russian text to see if you are happy with what you have. If not you can always go back and change things. Naturally, preprogrammed messages from your editor and other programs will look rather strange, but if you are good at working with DEBUG or similar utilities, you can alter all your favorite software to give you meaningful messages in Russian!

Now we come to the printer. My printer is a Roland PR 1010 (MX-80 clone), and both these machines will allow you to print Russian characters. With the MX-80, you have to have the parallel version with GRAFTRAX, whilst the ROLAND has a neater arrangement whereby you can actually load your font and then use your normal print utilities. Many other printers have similar capabilities, and you should read your manual to see if this can be done, and how. Once you have the basic idea, what follows may be useful to help you print your foreign language documents.

In the MX-80, sending the BASIC string:

```
CHR$(27)+"L"+CHR$(12)+CHR$(0)
```

will put the printer in hi-res graphics mode for twelve horizontal positions of the printhead. This is the width of a character. The next twelve codes will specify which pins are fired at each position, and when all twelve codes have been dealt with, the printer will return to character mode. Unfortunately, because the font itself is ROM based, you have to send each character column-by-column like this, each time you use it.

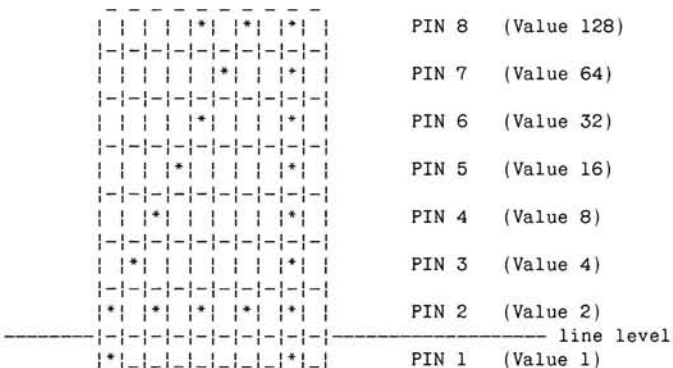
The Roland is a little different, and although the above scheme will still work, you can instead send the sequence:

```
CHR$(27)+"Y"+"A"
```

This informs the printer that you want to change the font of the character "A", and it will use the next NINE codes to build up the new font in RAM. This font remains in effect until the printer is powered-down or sent a reset signal (ESC+"@"). You can change as few or as many codes as you want in this way.

The way the codes specify which pins fire and when is easy to understand, works similarly in both machines and is fairly adequately described in both manuals. Take the following blow-up of the Russian "D" for example:

COLUMN: 1 2 3 4 5 6 7 8 9 10



For this letter, column 1 requires pins 1 and 2, and you add their binary values to obtain the number 3. Column 2 requires only pin 3, and its value is the number 4. Column 3 requires pins 2 and 4, with values of 8 and 2: 8+2=10. Thus you continue to add the values of the separate pins required in each column, to obtain a unique number for each column like this:

| | | | | | | | | | | |
|---------------|---|---|----|----|-----|----|-----|---|-----|----|
| COLUMN | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| Sum of values | 3 | 4 | 10 | 16 | 162 | 64 | 130 | 0 | 255 | 0 |

These numbers, you will not be surprised to hear, are the codes which are sent after the escape sequences to define the required font (for this letter), whether one-off as in the MX-80 or reusable as in the Roland. The reason that the MX-80 requires TWELVE codes to specify a letter and the Roland only wants NINE, is that the MX-80 has to be told about the three column space between letters, otherwise, you will get an effect called "unproportional spacing". Thus, when I had an MX-80, I sent the nine codes of font data followed by three CHR\$(0)s. With the Roland I use only the nine data items. (Note: since reviewing this, I seem to remember that the MX-80 used ten codes, but I no longer have an MX-80 to test this. Using this mode with the Roland, it does require twelve horizontal spaces per letter for the spacing to be uniform. One way to check is to send a document which has been right-justified. If the justification is correct, then you have sent the right number of padding 0s. If not, experiment with the number of codes you send till the result is what you expect.)

If you want to form your own letters, rather than use the ones provided in the listing, then use graph paper to place your dots, add up the values for each column, and write down your sequence of numbers. If you are using the MX-80 version, the tenth through twelfth numbers are always 0, of course, but it is easier to add these in the printing subroutine than type them into each data line.

Once you have loaded your font, any document prepared with your Russian version of ALTCHAR.SYS should print in Russian, using any normal print utility. To change back to English, just switch the printer off and on to erase the RAM and go back to the default English font. Sending an 'ESC @' will do the same thing.

That's it!

Listings:

(Note that the data lines 1000-1430 are the same in both versions. If you are using the font-dumping method (Roland) with a compatible printer, type in lines 1000-1430 from the MX-80 version.)

Listing A

This version is the font-loading version for the Roland printer. PNT\$ is a sequence of characters which produce Russian characters when they are sent to the printer. Due to the difficulty of printing the two sets of characters together, you will have to experiment with the program by dumping the font, then sending the printer PNT\$ itself. This will give you a one-to-one correspondence. It represents the key assignments and font types I use. Note that I also change the 'e' (last character in PNT\$). This is just because I don't like the 'e' in the Roland font and I change it even when I am working in English.

```
10 ' ***INITIALIZATION***
20 '
30 CLS:LOCATE 10,10:SCREEN 0,1
40 PRINT "Russian font program."
50 SCREEN 0,0:PRINT
60 PNT$="VvSsDdFfGgNnIiJjLlRrQqUuWwZz{\~'}|{|tbhkm*&e"
'PNT$="POINTER"
```

```

70 ESC$=CHR$(27)+"Y"
80 '
100 '      ***DOWNLOAD RUSSIAN CHARS***
110 '
120 ' (AS PER ROLAND PR-1010 MANUAL...ESC+Y+(HEX CODE)
    +(9 BYTES))
125 '
130 RESTORE 1020          'SET DATA POINTER
140 FOR Z=1 TO LEN(PNT$)
150 L$=MID$(PNT$,Z,1)
160 LPRINT ESC$+L$;
170 FOR H=1 TO 9 'GET THE ROW OF DATA FOR THIS LETTER
180 READ Y
190 LPRINT CHR$(Y);      'SEND EACH ITEM
200 '
210 NEXT H                'MORE DATA?
240 NEXT Z                'MORE LETTERS?
250 '
260 LOCATE 12,10:PRINT "FONT LOADED...Exiting..."
270 FOR TT=1 TO 700:NEXT TT:END
280 '
290 '
1000 ' *** DATA AREA ***
1010 '
1020 DATA 252,0,146,0,146,0,210,0,12
1030 DATA 60,64,18,64,18,64,18,12,0
1040 DATA 224,0,16,0,16,0,16,0,254
1050 DATA 48,0,8,0,8,0,8,0,62
1060 DATA 3,4,10,16,162,64,130,0,255
1070 DATA 3,0,6,8,50,0,34,0,63
1080 DATA 112,0,138,0,252,0,138,0,112
1090 DATA 28,0,34,34,127,34,34,0,28
1100 DATA 254,130,130,0,128,0,128,0,192
1110 DATA 62,32,32,0,32,0,48,0,0
1120 DATA 254,4,4,8,16,32,64,0,254
1130 DATA 62,4,4,0,8,0,16,0,62
1140 DATA 254,18,18,0,18,0,12,0,254
1150 DATA 0,62,32,42,0,10,4,0,62
1160 DATA 130,68,40,16,254,16,40,68,130
1170 DATA 0,34,20,8,54,8,20,34,0
1180 DATA 2,4,8,112,128,0,128,0,254
1190 DATA 2,4,8,48,0,32,0,62,2
1200 DATA 254,130,128,0,128,0,128,0,254
1210 DATA 62,32,32,0,32,0,32,0,62
1220 DATA 0,68,130,0,146,0,146,68,56
1230 DATA 20,34,0,34,8,34,8,34,28
1240 DATA 254,16,16,124,130,0,130,124,0
1250 DATA 62,34,8,8,28,34,0,34,28
1260 DATA 98,0,146,4,152,0,144,0,254
1270 DATA 18,40,2,44,0,40,0,62,2
1280 DATA 68,130,0,146,0,146,108,0,0
1290 DATA 20,34,0,34,8,42,20,0,0
1300 DATA 252,0,2,0,2,0,254,1,0
1310 DATA 60,0,2,0,2,0,62,1,0
1320 DATA 126,6,2,68,72,80,32,64,126
1330 DATA 62,4,68,64,72,64,80,64,62
1340 DATA 252,0,2,0,252,0,2,0,254
1350 DATA 60,0,2,0,60,0,2,0,62
1360 DATA 252,0,2,0,252,0,2,0,255
1370 DATA 60,0,2,0,60,0,2,0,63
1380 DATA 32,0,32,0,62,32,32,0,32
1390 DATA 62,2,42,0,42,0,42,20,0
1400 DATA 62,2,8,0,8,0,8,0,62
1410 DATA 62,2,8,0,20,0,34,0,0
1420 DATA 62,2,16,8,4,8,16,0,62
1430 DATA 32,0,32,62,10,10,0,10,4
1440 DATA 0,0,62,32,42,0,10,4,0
1450 DATA 28,34,8,34,8,34,8,34,24
1460 '

```

Listing B

This version is for the MX with graphics (or a similar version could be made for another graphics-capable printer).

```

10 ' *** PRINT UTILITY (MX-80 VERSION) ***
15 '
20 ' (TO PRINT RUSSIAN CHARACTERS)

```

```

25 '
30 ' First, prompt for filename. Error if not present--
40 ' flag and return Open file for sequential input. Only
50 ' so you can check it's there. Close sequential file.
60 ' Open for random access. This is so the machine will
70 ' read delimiters literally. Can't in seq. mode. Check
80 ' each letter. If 'normal'-- pass through. If 'special'
90 ' then Pass to subroutine which sets the data pointer at
100 ' the appropriate point for that letter, print it and
110 ' return. On EOF, prompt for another file or exit.
120 '
130 CLS:LOCATE 10,20
140 PRINT "RUSSIAN PRINTING PROGRAM"
150 PRINT
160 PNT$="VvSsDdFfGgNnIiJjLlRrQqUuWwZz|\~' }|[tvhkm*&e"
170 '
180 LOCATE 14,20:PRINT "DOCUMENT NAME?"
190 LOCATE 15,20:PRINT "'Q' WHEN DONE"
200 LOCATE 16,20:LINE INPUT "--";A$:
    IF A$="Q" OR A$="q" THEN SYSTEM
210 OPEN "I", #1, A$:IF ERR = 53 THEN 1450
    'MESSAGE & ERROR AREA
220 CLOSE #1
230 OPEN "R", #1, A$
240 FIELD #1, 128 AS N$
250 CLS: LOCATE 16,20:PRINT "PRINTING DOCUMENT: "A$;
    GOTO 270
260 IF EOF (1) THEN CLOSE #1:CLS:GOTO 180
270 GET #1
280 FOR F=1 TO LEN(N$)
290 Z$=MID$(N$,F,1):IF INSTR(PNT$,Z$)>0
    THEN GOSUB 310 ELSE LPRINT Z$;
300 NEXT F:GOTO 260
310 LPRINT CHR$(27)+"L"+CHR$(12)+CHR$(0);
320 Z%=INSTR(PNT$,Z$)
330 ON Z% GOTO 340,350,360,370,380,390,400,410,420,430,
    440,450,460,470,480,490,500,510,520,530,540,550,560,
    570,580,590,600,610,620,630,640,650,660,670,680,690,
    700,710,720,730,740,750,760,770
340 RESTORE 1000:GOTO 800
350 RESTORE 1010:GOTO 800
360 RESTORE 1020:GOTO 800
370 RESTORE 1030:GOTO 800
380 RESTORE 1040:GOTO 800
390 RESTORE 1050:GOTO 800
400 RESTORE 1060:GOTO 800
410 RESTORE 1070:GOTO 800
420 RESTORE 1080:GOTO 800
430 RESTORE 1090:GOTO 800
440 RESTORE 1100:GOTO 800
450 RESTORE 1110:GOTO 800
460 RESTORE 1120:GOTO 800
470 RESTORE 1130:GOTO 800
480 RESTORE 1140:GOTO 800
490 RESTORE 1150:GOTO 800
500 RESTORE 1160:GOTO 800
510 RESTORE 1170:GOTO 800
520 RESTORE 1180:GOTO 800
530 RESTORE 1190:GOTO 800
540 RESTORE 1200:GOTO 800
550 RESTORE 1210:GOTO 800
560 RESTORE 1220:GOTO 800
570 RESTORE 1230:GOTO 800
580 RESTORE 1240:GOTO 800
590 RESTORE 1250:GOTO 800
600 RESTORE 1260:GOTO 800
610 RESTORE 1270:GOTO 800
620 RESTORE 1280:GOTO 800
630 RESTORE 1290:GOTO 800
640 RESTORE 1300:GOTO 800
650 RESTORE 1310:GOTO 800
660 RESTORE 1320:GOTO 800
670 RESTORE 1330:GOTO 800
680 RESTORE 1340:GOTO 800
690 RESTORE 1350:GOTO 800
700 RESTORE 1360:GOTO 800
710 RESTORE 1370:GOTO 800
720 RESTORE 1380:GOTO 800

```

```

730 RESTORE 1390:GOTO 800
740 RESTORE 1400:GOTO 800
750 RESTORE 1410:GOTO 800
760 RESTORE 1420:GOTO 800
770 RESTORE 1430:GOTO 800
780 '
790 ' *** SBR TO SEND FONT DATA ***
795 '
800 FOR Y=1 TO 9
810 READ J
820 LPRINT CHR$(J); 'ONE HORIZONTAL COLUMN OF DATA
830 NEXT Y
835 LPRINT CHR$(0)+CHR$(0)+CHR$(0);
      'ADD 3 '0's FOR SPACE
840 RETURN
850 '
860 ' *** DATA AREA ***
870 '
1000 DATA 254,0,146,0,146,0,210,0,12
:
:
: (See Listing A)
:
:
1430 DATA 28,0,42,0,42,0,42,0,24
1440 '
1450 PRINT "DOCUMENT ";A$;" DOES NOT EXIST -- REPEAT!"
1460 FOR T=1 TO 2000:NEXT T 'SHORT DELAY LOOP
1470 RESUME 180
1480 '

```

*

Continued from Page 11



Intensity Mis-Adjusted

or car seat. The intensity and contrast controls, located on the left and right sides, respectively, regulate the 'pin action' from a 'mild tickle' to 'permanant injury'. SIT-A-SCREEN boasts a resolution of 60 pins/inch and presently supports normal and intensified (ouch!) characters. When ordering, specify model number SAS-6000 for the standard model or SAS-6000S for the split-screen version with monochrome tattoo adaptor. Both models come with Typing-Hooter, Hooter-Tooter and Turbo-Vibrate demo software packages.

Finally, from the Heath Parts Department, a kit of parts will soon be available to upgrade your H11 computer to the new Z-386, 32-bit computer. This kit will consist of a new cabinet, motherboard, power supply, CPU, I/O, video, memory, keyboard, disk drives and mounting hardware. Rumor has it that the kit will sell for around \$12000, and it is still unknown whether the H8 or H89 can be successfully converted.

*

HDS

ANNOUNCES

COMING SOON

THE Z100 VIDEO ENHANCEMENT

Display 16 colors on a standard monitor

Simple software interface

Easy to install

The **VI** video enhancement board from **HDS** will enable your Z100 (not PC) to display 16 colors. This is done by adding an extra bank of video ram. In addition, a palette is provided which maps the 16 colors to any combination of 16. This can be used to create simple animation effects.

The **VI** board sits above the video board and plugs into the main connector in parallel. In addition, a web is used to obtain 26 signals which are not available through the main connector. The web contains modules and sockets into which certain video board chips are placed.

Note: **NO** soldering or trace cutting is involved.

A Z100 computer with full video ram (with 64K chips) is required. To display all 16 colors, an IBM* compatible color monitor is needed.

To use a composite monochrome monitor, you can add the **VIM** option. This package contains chips to be plugged into the **VI** board to provide a weighted gray scale.

For more information, write or call today. Phone inquiries handled Mon-Fri, 8AM to 5PM Central time.

HUGHES DEVELOPMENT SYSTEMS
10101 S. W. FREEWAY, SUITE 400
HOUSTON TEXAS 77074
(713) 772-2840

*IBM is a registered trademark of International Business Machines Corp.

The Mouse And Turbo Pascal

M.D. Holmberg, II
5146C Gunn Avenue
Scott AFB, IL 62225

A while back I purchased the Microsoft Mouse, and within an hour or so I was writing programs using the Mouse. With the Mouse you can dress up your programs as never before imagined. Instead of remembering which key does what, you can design your program to respond to the Mouse cursor and a button click. When it comes to graphics programs, you can do things that were never before possible with the keyboard, such as drawing lines in the screen with smooth curves. The Microsoft Mouse is available from Heath for \$135.00 [MS-5063-11]. The package includes the Mouse itself, documentation, and drivers for the H/Z-100 and the H/Z PC. This tutorial is mainly geared to the H/Z PC. Listing 1 contains the necessary code for controlling the Mouse. It will work on the H/Z-100 with no modifications. Listing 2 is an example program demonstrating the use of the Mouse. It does, however, contain some PC specific code. For those using the H/Z-100 computer, refer to Listing 3 instead of Listing 2.

What Is A Mouse?

For those of you who are not familiar with the mouse, it is a pointing device using the basic principles of the light pen, but instead of pointing a pen to a particular location on the screen, the mouse is rolled along the table and a cursor moves around on the screen. The mouse cursor is defined by software, so it can be anything you desire. Such as an arrow, cross, X, check mark, or a pointing hand. The mouse also contains buttons. Some mice have one button (such as the MacIntosh) and some have three buttons (such as the Logitech Mouse, which is Microsoft compatible). The Microsoft Mouse has two buttons. These buttons are software controlled just as the keyboard is. A typical example of usage of the mouse is:

Suppose you wrote a program that contains a menu to perform certain functions, such as:

1. Edit
2. List
3. Print

Typically, you would perform these functions by pressing a key, such as (1,2,3) or maybe (F1, F2, F3). Now that you have a mouse you can do the same thing by moving it around the table until the mouse cursor is on the word 'EDIT' and clicking the button. This is a very simple example, but I think it explains what the mouse does.

Programming The Mouse

Writing programs to use the mouse is not very hard once you understand how the driver works. It contains 15 different functions which let you gain control of the mouse. This is done by using interrupt vector #51. All this means is that every time you make a call to the mouse, the system executes a subroutine in a predefined area of memory. You also pass parameters to this subroutine so it will know what you want it to do. Below is a list of the mouse functions:

- Reset mouse
- Display mouse cursor
- Hide mouse cursor
- Get mouse cursor and/or button status
 - Set cursor position
 - Get button pressed information
 - Get button released information
 - Set minimum/maximum horizontal position
 - Set minimum/maximum vertical position
- Set graphics cursor
 - Set text cursor
 - Read motion counters
 - Set subroutine address and interrupt mask
 - Turn on light pen emulation
 - Turn off light pen emulation

The functions marked with '•' are the ones we will be discussing in this article. As I have mentioned before, to gain control of the mouse you will call the mouse driver and pass parameters so that it will know what to do. The mouse driver uses the 8088 internal

Listing 1

```

{ myMouse.lib      calling routine for microSoft mouse
  written by: M D. Holmberg, II --  OCT 30 1986
}
const { functions }

```

```

resetMouse=0;
showMouse=1;
hideMouse=2;
buttonStatus=3;
mousePosition=3;
setCursor=4;
buttonPressed=5;
buttonReleased=6;
setMinMaxHoriz=7;
setMinMaxVert=8;
setCrafCursor=9;
setTextCursor=10;
readMotion=11;
setSubAddr=12;
litePenOn=13;
litePenOff=14;
setMickey=15;

{ cursors }
checkCursor = 1;
handCursor = 2;
xCursor = 3;
crossCursor = 4;
hourGlassCursor = 5;
arrowCursor = 6;
watchCursor = 7;

cursor : array[ 1..7, 0..1, 0..1, 0..15 ] of integer =

{ check } ((($f00,$f0e0,$f0c0,$f081,$f03,$f0607,$000f,$001f,
$c03f,$f07f,$ffff,$fff,$fff,$fff,$fff,$fff,$fff,$fff),
($0000,$0006,$000c,$0018,$0030,$0060,$00c0,$0180,
$0700,$0000,$0000,$0000,$0000,$0000,$0000,$0000)),
{ hand } (($elff,$elff,$elff,$elff,$elff,$e000,$e000,$e000,
$0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000),
($1e00,$1200,$1200,$1200,$1200,$13ff,$1249,$1249,
$f249,$9001,$9001,$9001,$8001,$8001,$8001,$ffff)),
{ x } (($07e0,$0180,$0000,$c003,$f00f,$c003,$0000,$0170,
$07e0,$ffff,$fff,$fff,$fff,$fff,$fff,$fff,$fff),
($0000,$700e,$1c38,$0660,$03c0,$0660,$1c38,$700e,
$0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000)),
{ cross } (($fc3f,$fc4f,$fc3f,$0000,$0000,$0000,$fc4f,$fc3f,
$fc3f,$fff,$fff,$fff,$fff,$fff,$fff,$fff),
($0000,$0180,$0180,$0180,$7ffe,$0180,$0180,$0180,$0180,
$0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000)),
{ hour } (($0000,$0000,$0000,$8001,$c003,$e007,$f00f,$e007,
$c003,$8001,$9000,$0000,$0000,$0000,$0000,$ffff),
{ glass }

```

```

{ arrow }
($0000,$7ffe,$6006,$300c,$1818,$0c30,$0660,$03c0,
$0660,$0c30,$1998,$33cc,$67e6,$7ffe,$0000,$0000)),
(($3fff,$1fff,$0fff,$07ff,$03ff,$01ff,$00ff,$007f,
$003f,$001f,$01ff,$10ff,$30ff,$f87f,$f87f,$fc3f),
($0000,$4000,$6000,$7000,$7800,$7c00,$7e00,$7f00,
$7f80,$7fc0,$7c00,$4600,$0600,$0300,$0300,$0180)),
(($e007,$e007,$e007,$e007,$c003,$8001,$8001,$8001,
$8001,$8001,$c003,$e007,$e007,$e007,$e007,$e007),
($0f0,$0ff0,$0ff0,$0ff0,$1008,$2084,$2084,$2086,
$2f86,$2004,$2204,$1008,$0fff,$0fff,$0fff,$0fff));

```

```
z100:boolean=false;
```

```

type
mouseResult = record
  Installed : boolean;
  numberButtons : integer;
  button1 : boolean;
  button2 : boolean;
  anyButton : boolean;
  vPos : integer;
  hPos : integer;
  charRow : integer;
  charCol : integer;
  button : boolean;
  buttonCount : integer;
  hMotion : integer;
  vMotion : integer;
end;

```

```
var mouseReturn : mouseResult;
```

```
procedure mouse(ml,m2,m3,m4 : integer);
```

```
{
```

```
calling the mouse procedure:
```

when any of the mouse routines are called the return values are put into 'mouseReturn'. if a function does not have a return it remains unchanged, otherwise 'mouseReturn' is reinitialized and loaded with new values.

if you need to keep a value it is suggested that you save before calling another mouse function.

Boolean variables:

```
mouseReturn.installed: true if installed
                    false if not installed
```

```
mouseReturn.button1 : \
button2 : \
anyButton: true if button pressed,
button : /      false if button not pressed
```

'cursorSegment' when calling function 9 (Set graphics cursor)
be sure this variable has the address of your cursor.

Microsoft mouse functions

function 0 -- mouse reset.

call: m1 := mouseReset.

return: mouseReturn.installed := true or false
.numberButtons := 2;

function 1 -- Display Cursor.

call: m1 := displayMouse.

return: none.

function 2 -- Hide Cursor.

call: m1 := hideMouse.

return: none.

function 3 -- Cursor Position and button status.

call: m1 := buttonStatus or mousePosition.

return: mouseReturn.button1 := true or false
.button2 := true or false
.anyButton := true or false
.hPos := horiz position
.vPos := vert position
.charRow := character row position
.charCol := character column position

function 4 -- Set Cursor Position.

call: m1 := setCursor.
m3 := new horiz position
m4 := new vert position

return: none

function 5 -- Button Pressed Information.

call: m1 := buttonPressed
m2 := 0,1 (left button, right button)

return: mouseReturn.button := true or false
.buttonCount := # of presses
.hPos := horiz position of last press
.vPos := vert position of last press

function 6 -- Button Released Information.

call: m1 := buttonReleased
m2 := 0,1 (left button, right button)

return: mouseReturn.button := true or false
.buttonCount := # of releases
.hPos := horiz position of last release
.vPos := vert position of last release

function 7 -- Set min/max horiz position.

call: m1 := setMinMaxHoriz
m3 := minimum horiz position
m4 := maximum horiz position

return: none

function 8 -- Set min/max vertical position.

call: m1 := setMinMaxVert
m3 := minimum vertical position
m4 := maximum vertical position

return: none

function 9 -- Set Graphics Cursor.

call: m1 := setGrafCursor
m4 := cursor constant

return: none

function 10 -- Set Text Cursor.

call: m1 := setTextCursor
m2 := 0,1 (software, hardware cursor)
m3 := screen mask
m4 := cursor mask

return: none

function 11 -- Read Motion Counters.

call: m1 := readMotion

return: mouseReturn.hMotion := horiz counter value
.vMotion := vert counter value

function 12 -- Set Subroutine address and interrupt mask

function 13 -- Turn on lite pen emulation

function 14 -- Turn off lite pen emulation

function 15 -- Set Mickey to pixel ratio

```

type RegisterSet=Record
  AX,BX,CX,DX,BP,SI,DI,DS,ES,Flags: Integer;
End;

var reg : registerSet;
mouseFunction : integer;

procedure initReturn;
begin
  with mouseReturn do
    begin
      installed := false;
      numberButtons := 0;
      button1 := false;
      button2 := false;
      anyButton := false;
      hPos := 0;
      vPos := 0;
      button := false;
      buttonCount := 0;
      hMotion := 0;
      vMotion := 0;
      charCol:=0;
      charRow:=0;
    end;
  end;

begin
  if m1 in [0,3,5,6,11] then initReturn;
  mouseFunction := m1;
  reg.AX := m1;
  reg.BX := m2;
  reg.CX := m3;
  reg.DX := m4;
  reg.ES := cSeg;
  if mouseFunction = setGrafCursor then
  case m4 of
    checkCursor : begin
      reg.dx := ofs(cursor[1,0,0]);
      reg.bx := 6;
      reg.cx := 8;
    end;
    handCursor : begin
      reg.dx := ofs(cursor[2,0,0]);
      reg.bx := 5;
      reg.cx := 0;
    end;
    xCursor : begin
      reg.dx := ofs(cursor[3,0,0]);
      reg.bx := 7;
      reg.cx := 4;
    end;
    crossCursor : begin
      reg.dx := ofs(cursor[4,0,0]);
      reg.bx := 7;
      reg.cx := 4;
    end;
  end;
end;

```

```

hourGlassCursor : begin
  reg.dx := ofs(cursor[5,0,0]);
  reg.bx := 7;
  reg.cx := 7;
end;

arrowCursor : begin
  reg.dx := ofs(cursor[6,0,0]);
  reg.bx := 0;
  reg.cx := 0;
end;

watchCursor : begin
  reg.dx := ofs(cursor[7,0,0]);
  reg.bx := 7;
  reg.cx := 7;
end;

end;
intr(51,reg);
case mouseFunction of
  0 : begin
    if reg.AX = -1 then mouseReturn.installed := true;
    mouseReturn.numberButtons := reg.BX;
  end;
  3 : begin
    if reg.BX and 1 = 1 then mouseReturn.button1 := true;
    if reg.BX and 2 = 2 then mouseReturn.button2 := true;
    if mouseReturn.button1 then mouseReturn.anyButton := true;
    if mouseReturn.button2 then mouseReturn.anyButton := true;
    mouseReturn.hPos := reg.CX;
    mouseReturn.vPos := reg.DX;
    if not z100 then mouseReturn.charRow:=mouseReturn.vPos div 8+1 else
      mouseReturn.charCol:=mouseReturn.hPos div 8+1;
    end;
  5 : begin
    if reg.AX and (m2+1) = m2+1 then mouseReturn.button := true;
    mouseReturn.buttonCount := reg.BX;
    mouseReturn.hPos := reg.CX;
    mouseReturn.vPos := reg.DX;
  end;
  6 : begin
    if reg.AX and (m2+1) = m2+1 then mouseReturn.button := true;
    mouseReturn.buttonCount :=reg.BX;
    mouseReturn.hPos := reg.CX;
    mouseReturn.vPos := reg.DX;
  end;
  11 : begin
    mouseReturn.hMotion := reg.CX;
    mouseReturn.vMotion := reg.DX;
  end;
end;
end;

```


registers for parameter passing. It also returns values in these same registers. The registers affected are AX, BX, CX, DX, and ES. Each function requires that some or all of these registers contain a particular value. In some cases, it may only be the function number. For example, the function to reset the mouse requires 0 in AX.

To isolate the programmer from all this register business I have constructed a Turbo Pascal procedure that does all this stuff for you. It is called 'MOUSE' and requires four parameters. The code for this is contained in Listing 1. For example, to reset the mouse you would type `mouse(0,0,0,0)`;

The Mouse Procedure

If you will note Listing 1, you will see how to call the mouse to make fantastic programs. In the constants section, you will see all of the mouse functions and cursor numbers followed by predefined cursors. These cursors came from the mouse documentation except for the last one which is the watch cursor. This is the only thing I like about the Macintosh so I included it here. Also is a boolean variable initialized to false, if you are using this file on an H/Z-100, you must set it to true or else the mouse procedure will not calculate the character row correctly.

Next you will note is the TYPE declaration. I have defined a record called 'mouseResult'. To the right of each field you will see the function numbers that return a result. For example: 'installed' is a boolean value set by function '0' which is reset mouse.

In the variable declaration section you will see the variable mouseReturn. Everytime the mouse procedure is called, 'mouseReturn' will contain information about the mouse. It is important to note that each time a call is made, that 'mouseReturn' is reinitialized. If you need to save some information, you need to copy it into your own variable. A good example of this is; when you start up a program, you should reset the mouse. After this call, 'mouseReturn.installed' will either be set to true or false. If you copy this value, you can check it before subsequent calls to the mouse or you can abort the program.

Cursor Position/Button Status

You will notice that Listing 1 contains documentation on what is passed to and returned from each call. I will, however, explain in detail function #3 which returns a lot of information. First of all, not all of the information in 'mouseReturn' comes from the mouse driver. Some of it is calculated by the mouse procedure. This function is called as: `mouse(3,0,0,0)`; If you prefer, you may use the constants at the top of Listing 1, you may find your program easier to understand if you do. In this case, the constants you should use are `buttonStatus` or `mousePosition`. Such as: `mouse(buttonStatus, 0,0,0)`; You should note that when a parameter is not required by the driver, I have used zeros.

On return from the call the following information is returned:

`mouseReturn.button1` — true if left button pressed.
`mouseReturn.button2` — true if right button pressed.
`mouseReturn.anyButton` — true if either button pressed.
`mouseReturn.hPos` — pixel column of mouse cursor.
`mouseReturn.vPos` — pixel row of mouse cursor.
`mouseReturn.charRow` — character row of mouse cursor.
`mouseReturn.charCol` — character column of mouse cursor.

As you can imagine this is indeed a very powerful function.

Other Things

You can very quickly write a simple program using these functions with Turbo Pascal. Here is an example:

```
program sample;

{$i mymouse.lib} { assuming you typed in listing 1 and }
                 { called it mymouse.lib }

begin
  hiRes; { PC hires mode }
  mouse(resetMouse,0,0,0); { this resets the mouse }
  if not mouseReturn.installed then exit;
  { abort if the mouse or driver is }
  { not installed}
  mouse(showMouse,0,0,0); { make mouse cursor show }
  writeln('I Love my PC ... ');
  repeat
    mouse(buttonStatus,0,0,0); { get button status }
  until mouseReturn.anyButton; { quit if any button }
  { pressed }
  mouse(hideMouse,0,0,0); { make mouse cursor disappear }
end.
```

As you can see, this is a simple program because most of the mouse stuff is already taken care of thanks to Listing 1.

Note: You should be aware that you can use the mouse in the high resolution mode, or in the text mode, but in the text mode, you can only use one of the 255 PC characters.

A Sample Mouse Program

To demonstrate how neat the mouse is I have included Listing 2, which uses several of the mouse functions. This is the two player game of checkers. I chose this game to demonstrate the mouse because it is not a complex game, and everybody knows how to play. Listing 2 contains six procedures plus the main program which I will explain in detail.

Init

This procedure initializes the variables of the program. You will note that I cheated. It probably isn't considered good programming practice, but it works well. To initialize the 64 squares on the board (`board[1..64]`), I declared the variable `initStr` to occupy the same place in memory as 'BOARD'. That is the reason for the global variable 'DUMMY', this is the string length byte. The characters in `initStr` represent the red, black, empty, and illegal spaces at the start of play. Also, the variables, `redStack` and `blackStack`, are set to zero. These represent the pieces that are removed from the board during the play of the game.

DrawBoard

This procedure draws the playing board on the screen. It uses the constant `template[1..8,1..8]` to detect if a square is red or black. It also draws the game title and some mouse controls which will be discussed later. Note that at the beginning of the procedure is: `'mouse(hideMouse,0,0,0)'` and at the end of the procedure is: `'mouse(showMouse,0,0,0)'`. These calls to the mouse need to be here, because whenever you change the contents of the screen and you move the mouse cursor, it will leave behind whatever was on the screen before you changed it. This is how the mouse cursor appears to move over the top of a screen image without damaging it. It is constantly saving the contents of video memory and replacing it when the cursor moves. You will also find through experimentation that if the mouse cursor is showing and you scroll up the screen, you will have several mouse cursors on the screen, only one of which is active.

Square

This procedure is called by 'drawBoard' and all it does is draw a red or black square.

Listing 2

```

program checkers;
{$C-U-}
{$i myMouse.lib}

var
  dummy:byte;
  board:array[1..8,1..8] of char;
  { e=empty, r=red, b=black }
  { R=red King B=black king }
  { i = illegal }
  { board[row,col] }

  redStack:integer;
  blackStack:integer;

procedure init;
var
  counter,counter1:integer;
  initStr:string[64] absolute dummy;

begin
  initStr:='iriririr'+
  'riririri'+
  'iriririr'+
  'eieieiei'+
  'ieieieie'+
  'bibibibi'+
  'ibibibib'+
  'bibibibi';

  redStack:=0;
  blackStack:=0;

end;

procedure drawBoard;
const
  template:array[1..8,1..8] of byte=
  ((1,0,1,0,1,0,1,0),
  (0,1,0,1,0,1,0,1),
  (1,0,1,0,1,0,1,0),
  (0,1,0,1,0,1,0,1),
  (1,0,1,0,1,0,1,0),
  (0,1,0,1,0,1,0,1),
  (1,0,1,0,1,0,1,0),
  (0,1,0,1,0,1,0,1));
  sqr:string[10]=#219#219#219#219#219#219#219#219#219#219;

var
  counter,counter2,counter3:integer;
  col,row:integer;

procedure square(x,y:integer;f:boolean);
var
  counter,counter1:integer;

begin
  x:=x*7-6;
  y:=y*3-2;
  for counter:=0 to 2 do
  begin
    gotoxy(x,y+counter);
    if f then for counter1:=1 to 7 do write(#219) else
      for counter1:=1 to 7 do write(' ');
  end;
end;

```

```

begin
  mouse(hideMouse,0,0,0);
  hiRes;
  hiResColor(green);
  init;
  col:=1;
  row:=1;
  counter3:=1;
  for counter:=1 to 8 do
  for counter2:=1 to 8 do
  begin
    if template[counter,counter2]=0 then square(counter2,counter,false) else
      square(counter2,counter,true);
  end;
  gotoxy(59,1);
  write('C h e c k e r s');
  gotoxy(59,2);
  write('-----');
  gotoxy(59,4);
  write('Restart');
  gotoxy(59,5);
  write('Quit');
  gotoxy(59,20);
  write(' S t a c k');
  gotoxy(59,21);
  write('-----');
  gotoxy(59,22);
  write(#176#176#176#176#176#176#176#176#176#176#219#219#219);
  gotoxy(59,23);
  write(#176#176#176#176#176#176#176#176#176#176#219#219#219);
  gotoxy(59,24);
  write(#176#176#176#176#176#176#176#176#176#176#219#219#219);
  gotoxy(59,25);
  write('-----');
  gotoxy(60,23);
  write(redStack:2);
  gotoxy(68,23);
  write(blackStack:2);
  mouse(showMouse,0,0,0);
end;

procedure setUp;
var
  counter,counter1:integer;
  col,row:integer;

begin
  mouse(hideMouse,0,0,0);
  for counter:=1 to 8 do
  for counter1:=1 to 8 do
  begin
    col:=counter1*7-6;
    row:=counter*3-2;
    gotoxy(col+2,row+1);
    case board[counter,counter1] of
      'r':write(#176#176#176#176);
      'R':write(#176'K'#176);
      'b':write(#219#219#219#219);
      'B':write(#219'K'#219);
      'e':write(' ');
    end;
  end;
end;

```

```

end;
gotoxy(60,23);
write(redStack:2);
gotoxy(68,23);
write(blackStack:2);
mouse(showMouse,0,0,0);
end;

procedure doMove;
var
  col, row:integer;
  oldCol,oldRow:integer;
begin
  col:=mouseReturn.charCol;
  row:=mouseReturn.charRow;
  if (Col in [1..56]) and
    (Row in [1..24]) then else exit;
  if col mod 7=0 then col:=col div 7 else col:=col div 7+1;
  if row mod 3=0 then row:=row div 3 else row:=row div 3+1;
  if board[row,col] in ['r','b','R','B'] then
  begin
    oldCol:=col;
    oldRow:=row;
    end else exit;
  repeat
    mouse(buttonStatus,0,0,0);
    if mouseReturn.button1 then
      begin
        col:=mouseReturn.charCol;
        row:=mouseReturn.charRow;
        if col mod 7=0 then col:=col div 7 else col:=col div 7+1;
        if row mod 3=0 then row:=row div 3 else row:=row div 3+1;
        end;
      until not mouseReturn.button1;
      if (col in [1..8]) and (row in [1..8]) then
        begin
          if board[row,col] = 'e' then
            board[row,col] := board[oldRow,oldCol];
            case row of
              8:if board[row,col]='r' then
                begin
                  board[row,col]:='R';
                  redStack:=redStack-1;
                end;
              1:if board[row,col]='b' then
                begin
                  board[row,col]:='B';
                  blackStack:=blackStack-1;
                end;
            end;
            board[oldRow,oldCol]:='e';
            setup;
          end;
        end;
      if (mouseReturn.charRow in [21..25]) and
        (mouseReturn.charCol in [58..71]) then

```

```

begin
  case board[oldRow,oldCol] of
    'r':redStack:=redStack+1;
    'b':blackStack:=blackStack+1;
    'R':redStack:=redStack+2;
    'B':blackStack:=blackStack+2;
  end;
  board[oldRow,oldCol]:='e';
  setup;
end;

procedure doRestart;
begin
  gotoxy(10,10);
  write(' ');
  gotoxy(10,11);
  if redStack = 12 then write(' Black W I N S ') else
  write(' Red W I N S ');
  mouse(setGrafCursor,0,0,watchCursor);
  delay(3000);
  drawBoard;
  setup;
  mouse(setGrafCursor,0,0,handCursor);
end;

begin
  mouse(resetMouse,0,0,0);
  if not mouseReturn.installed then
  begin
    write('Mouse not installed ... ');
    textMode;
    exit;
  end;
  mouse(setGrafCursor,0,0,handCursor);
  mouse(showMouse,0,0,0);
  drawBoard;
  setup;
  repeat
    if (redStack=12) or (blackStack=12) then doRestart;
    mouse(buttonStatus,0,0,0);
    if (mouseReturn.button1) and (mouseReturn.charRow=4) and
      (mouseReturn.charCol in [59..65]) then
      begin
        drawBoard;
        setup;
        end else if mouseReturn.button1 then doMove;
      until (mouseReturn.button1) and (mouseReturn.charRow=5) and
        (mouseReturn.charCol in [59..62]);
      mouse(hideMouse,0,0,0);
      textMode;
    end;
  end;

```


SetUp

This procedure is called after a move has been made and looks at that contents of the variable 'BOARD' for the placement of the pieces. Note that we also hide and show the mouse to keep the screen in tact.

DoMove

This is the heart of the play of the game and the mechanics of using the mouse. This procedure is called from the main program and 'buttonStatus' has just been called. Remember that the global variable 'mouseReturn' contains the results from each call. Upon entry to this routine we save the character coordinates for later use, then we check to see if we're within the limits of the playing board. Then, we mathematically convert the row and column coordinates to fit the array indexes of the variable 'BOARD'. So now, row and col are equal to a square and we check the contents of that square. Now comes the crux of moving a checker. Here we enter into a repeat loop which continues until the mouse button is released. Within the loop we are constantly making a call to the mouse and returning the button status. If the button is not pressed, then the loop is terminated. After that, we check the coordinates and if it is a legal square or if we are removing a piece from the board, then we fix up our variables to reflect that.

DoRestart

This routine is called only if all the checkers have been removed from the board; either red or black. At this time, a banner is written in the middle of the screen informing the players who won. At this time, we change the mouse cursor to a watch, indicating that the computer is busy. We then make a call to 'drawBoard' and 'setUp' to continue with another game.

The Main Program

This section is very similar to the simple program mentioned above. First, we do a mouse reset. If the mouse or driver is not installed, a message is printed and the program exits. Then a call is made to the mouse to set the graphics cursor. The one I used is the hand cursor. Since checkers is usually played with your hand and not an arrow it seemed to make sense. Next, a repeat loop is entered which executes the main play and the mouse controls. We also check to see if all the pieces have been removed from the board. Within this loop you can move a piece, take a piece off the board, restart the game, or quit the game. The last thing we do is get rid of the mouse cursor and put the PC into the text mode.

Running The Program

Using Turbo Pascal, enter Listing 1 and Listing 2 (or Listing 3 if you are using an H/Z-100). Name Listing 1 'Mymouse.lib', and Listing 2 'Checkers.pas'. If you made no typos, you should have a clean compile. Be sure you compile 'Checkers.pas'. 'MyMouse.lib' will be called in during the compile. This program will run in memory, or if you wish, you may compile to a 'COM' file.

The play of the game is simple. Move the mouse cursor to the checker you wish to move, then while pressing the left button, move it to the square you want to move it to and release the button. If it was a legal square, the checker will move there. In the bottom right of the screen is the stack. This area represents checkers removed from the board. To remove a checker from the board, move the mouse cursor to the piece you wish to move and press the button. Then move the mouse cursor to the stack area while pressing the button. You will then see the red or black counter increment. If you move to the king row, the checker will turn into a king and the appropriate stack counter will be decremented.

Notice that the upper right section of the screen contains the words 'Restart' and 'Quit'. These are mouse controls. If you click the mouse on restart, the game will be restarted. If you click the mouse on quit, the program will end.

Notes

Be sure that the mouse is connected and the driver is installed. The checkers program uses the extended character set in the hiRes mode so be sure that 'graftable' (from your MS-DOS disk) has been run prior to program execution. It would be a good idea if 'graftable' is included in your autoexec batch file.

I have enjoyed writing this article on programming the mouse, and I hope you enjoy using your mouse.

H/Z-150/160 & H-89 SPEED MODS

- H-89 2/4 MHz, No trace cuts! Z80A and Software Included. Assembled \$34.95, Kit \$24.95, Board \$20
- H/Z-150/160 4.77/6.77 MHz. Hardware Reset included FREE. Satisfaction Guaranteed \$34.95

SOFTWARE for H/Z-150, 100, 89, H-8!

- Paycheck \$39.95 - Perfect Printer \$19.95

H-89 20 MEGA BYTE WINCHESTER!

- Boots from the Hard Disk. ST225 Drive. ONLY \$595

H-150/160 20 MEG WINCHESTER \$400!

ORDER NOW by writing or calling:

Micronics Technology (904) 897-4257

449 Barbados Way Niceville, FL 32578

Checks, VISA, MC. Shipping \$2. Winchester \$15.

EVERYTHING YOU NEED... \$279⁰⁰

Now it's easy to program the Heath-Zenith HERO-1* Robot with an Apple* II. HERO* Macros for the S-C Software 6800 Cross Assembler program in Heath's Robot Interpreter Language with easily remembered mnemonics.

For example, the line: 1130> MVWRIM GRIP, OPEN, 60, FAST instructs the HERO* to open his gripper 60 units at fast speed. Motor position is expressed in base-10.

The HERO* Macros come with 30 pages of documentation. Transfer to HERO* with ROBI... an affordable interface for the robotics experimenter... is simple.

- ROBI is a complete package. No additional hardware required for Apple* or HERO*.
- ROBI installs quickly in an Apple* II, II+, or IIe. Once installed, no hardware changes are needed. Within minutes, you will be programming HERO*.
- With ROBI and the Cross Assembler, the programmer uses Apple*'s memory to write the program, and HERO*'s memory to run the program.
- Not "copy protected," archival copies may be made as needed.
- ROBI offers expansion potential.

VISA and MasterCard accepted.

BERSEARCH
Information Services

26160 Edelweiss Circle
Evergreen, Colorado 80439

The Cross Assembler with
HERO* Macros sells for
\$100.00; the ROBI Interface
sells for \$199.00. Both as
a package - \$279.00.

To order, or for more
information, call
(303) 670-6137.

APPLE* is a trademark of Apple Computer. HERO* is a trademark of Heath Electronics.

Listing 3

```

program checkers;
{$C-,u-}
{$I myMouse.lib}

var
  dummy:byte;
  board:array[1..8,1..8] of char;
  { e=empty, r=red, b=black
  { R=red King B=black king
  { i = illegal
  { board[row,col]

  redStack:integer;
  blackStack:integer;

procedure init;
var
  counter,counter1:integer;
  initStr:string[64] absolute dummy;
begin
  initStr:='iriririr'+
  'riririri'+
  'iriririr'+
  'ieieieie'+
  'ieieieie'+
  'bibibibi'+
  'ibibibib'+
  'bibibibi';
  redStack:=0;
  blackStack:=0;
end;

procedure drawBoard;
const
  template:array[1..8,1..8] of byte=
  ((1,0,1,0,1,0,1,0),
  (0,1,0,1,0,1,0,1),
  (1,0,1,0,1,0,1,0),
  (0,1,0,1,0,1,0,1),
  (1,0,1,0,1,0,1,0),
  (0,1,0,1,0,1,0,1),
  (1,0,1,0,1,0,1,0),
  (0,1,0,1,0,1,0,1));
  sqr:string[10]='#27'p
  '#27'q';

var
  counter,counter2,counter3:integer;
  col,row:integer;

procedure square(x,y:integer;f:boolean);
var
  counter,counter1:integer;
begin
  x:=x*7-6;
  y:=y*3-2;
  for counter:=0 to 2 do
  begin
    gotoxy(x,y+counter);
    if f then for counter1:=1 to 7 do write('#27'p '#27'q') else
      for counter1:=1 to 7 do write(' ');
  end;
end;
end;

```

```

begin
  mouse(hideMouse,0,0,0);
  clrscr;
  write('#27'yl '#27'xl');
  init;
  col:=1;
  row:=1;
  counter3:=1;
  for counter:=1 to 8 do
  for counter2:=1 to 8 do
  begin
    if template[counter,counter2]=0 then square(counter2,counter,false) else
      square(counter2,counter,true);
  end;
  gotoxy(59,1);
  write('C h e c k e r s');
  gotoxy(59,2);
  write('-----');
  gotoxy(59,4);
  write('Restart');
  gotoxy(59,5);
  write('Quit');
  gotoxy(59,20);
  write(' S t a c k');
  gotoxy(59,21);
  write('-----');
  gotoxy(59,22);
  write('#27'Fiii '#27'G '#27'p '#27'q');
  gotoxy(59,23);
  write('#27'Fiii '#27'G '#27'p '#27'q');
  gotoxy(59,24);
  write('#27'Fiii '#27'G '#27'p '#27'q');
  gotoxy(59,25);
  write('-----');
  gotoxy(60,23);
  write(redStack:2);
  gotoxy(68,23);
  write(blackStack:2);
  mouse(showMouse,0,0,0);
end;

procedure setUp;
var
  counter,counter1:integer;
  col,row:integer;
begin
  mouse(hideMouse,0,0,0);
  for counter:=1 to 8 do
  for counter1:=1 to 8 do
  begin
    col:=counter1*7-6;
    row:=counter*3-2;
    gotoxy(col+2,row+1);
    case board[counter,counter1] of
      'r':write('#27'Fiii '#27'G');
      'b':write('#27'Fiii '#27'G');
      'R':write('#27'p '#27'q');
      'B':write('#27'p '#27'qK '#27'p '#27'q');
      'e':write(' ');
    end;
  end;
end;

```

```

end;
gotoxy(60,23);
write(redStack:2);
gotoxy(68,23);
write(blackStack:2);
mouse(showMouse,0,0,0);
end;

procedure doMove;
var
  col, row:integer;
  oldCol,oldRow:integer;
begin
  col:=mouseReturn.charCol;
  row:=mouseReturn.charRow;
  if (Col in [1..56]) and
    (Row in [1..24]) then else exit;
  if col mod 7=0 then col:=col div 7 else col:=col div 7+1;
  if row mod 3=0 then row:=row div 3 else row:=row div 3+1;
  if board[row,col] in ['r','b','R','B'] then
    begin
      oldCol:=col;
      oldRow:=row;
      end else exit;
    repeat
      mouse(buttonStatus,0,0,0);
      if mouseReturn.button1 then
        begin
          col:=mouseReturn.charCol;
          row:=mouseReturn.charRow;
          if col mod 7=0 then col:=col div 7 else col:=col div 7+1;
          if row mod 3=0 then row:=row div 3 else row:=row div 3+1;
          end;
        until not mouseReturn.button1;
        if (col in [1..8]) and (row in [1..8]) then
          begin
            if board[row,col] = 'e' then
              begin
                board[row,col] := board[oldRow,oldCol];
                case row of
                  8:if board[row,col]='r' then
                    begin
                      board[row,col]:='R';
                      redStack:=redStack-1;
                    end;
                  1:if board[row,col]='b' then
                    begin
                      board[row,col]:='B';
                      blackStack:=blackStack-1;
                    end;
                end;
                board[oldRow,oldCol]:='e';
                setUp;
              end;
            end;
            if (mouseReturn.charRow in [21..25]) and
              (mouseReturn.charCol in [58..71]) then
              begin
                case board[oldRow,oldCol] of

```

```

'r':redStack:=redStack+1;
'b':blackStack:=blackStack+1;
'R':redStack:=redStack+2;
'B':blackStack:=blackStack+2;
end;
board[oldRow,oldCol]:='e';
setUp;
end;

procedure doRestart;
begin
  gotoxy(10,10);
  write(' ');
  gotoxy(10,11);
  if redStack = 12 then write(' Black
  W I N S
  Red
  W I N S
  ');
  gotoxy(10,12);
  write(' ');
  mouse(setGrafCursor,0,0,watchCursor);
  delay(3000);
  drawBoard;
  setUp;
  mouse(setGrafCursor,0,0,handCursor);
end;

begin
  z100:=true;
  write(#27'x5'#27'x1'#27'm40');
  mouse(resetMouse,0,0,0);
  if not mouseReturn.installed then
    begin
      writeln('Mouse not installed ... ');
      exit;
    end;
  mouse(setGrafCursor,0,0,handCursor);
  mouse(showMouse,0,0,0);
  drawBoard;
  setUp;
  repeat
    if (redStack=12) or (blackStack=12) then doRestart;
    mouse(buttonStatus,0,0,0);
    if (mouseReturn.button1) and (mouseReturn.charRow=4) and
      (mouseReturn.charCol in [59..65]) then
      begin
        drawBoard;
        setUp;
      end else if mouseReturn.button1 then doMove;
    until (mouseReturn.button1) and (mouseReturn.charRow=5) and
      (mouseReturn.charCol in [59..62]);
    mouse(hideMouse,0,0,0);
    write(#27'y5'#27'y1');
    clrscr;
  end.

```

*

Two Terminal Games!!!

Steven W. Vagts
9509 Gray Mouse Way
Columbia, MD 21046

Got two terminals? How about using them for two or more person games! Here's some help getting started.

Many of us computer fanatics have, by now, collected more than one computer — usually over the strong objections of the budget conscious spouse. One of the primary excuses I used was the argument that I could get the two terminals to talk to each other (hit on the educational benefits) and could then program real “two person” games, such as “Battleship” and “Scrabble”. Some secrecy of the other person's, or team's, moves or letters is desired. Also, it's a real pain swapping seats or being hunched over, sharing a terminal with someone else. The other excuse being that, in a family of five, a second terminal would come in handy when the children become interested in “toying” with the computer. My youngest, 6, enjoys just typing on the computer while it's off line, while the others are rapidly becoming more involved in games.

Anyway, after convincing my wife some time ago that I needed to get a Heathkit H-89 because I needed the disk drive (my first was an H-88), I had to rely heavily on the excuses above to keep from selling the H-88 and eventually upgrade it to an H-89 also. Having finally done that though, the die was cast and it was time to follow through with my promises — get the two to intelligently talk to each other.

Hardware

The H/Z-89 with the Serial Interface Accessory installed is designed to interface with accessories that use the RS-232C standards of the Electronic Industries Association (EIA) and 25 pin 'D' connectors. This RS-232C standard places all equipment into one of two general categories:

DTE — Data Terminal Equipment, and
DCE — Data Communication Equipment

Computers and modems are two types of DCE; while terminals, printers, and most peripherals are DTE. The Heath H-89 Operation Manual cautions, however, “Always connect a DTE to a DCE. Never connect two like types together.” For various reasons, though, I had to be difficult. I wanted to use the DTE330Q port on both computers.

Reading through the various books I've managed to collect through the years and reviewing my past copies of SEXTANT and REMark, however, quickly showed how little was actually written on the subject. Most references only addressed DTE to DCE (Data Communication Equipment) connections. Thankfully, I had obtained “MAPLE”, Modem APpLication Effector, from the National Heath Users' Group (HUG) recently and its author, Dr. William C. Parke, had provided just the information needed.

If you intend to use a DTE to DCE connection, you need only purchase a cable having a male and female end with straight through connections to pins 1 thru 8 and 20. For DTE to DTE connections, however, a cable must be made or purchased that have lines 2 and 3 reversed, lines 4 and 5 reversed, and lines 6 and 20 reversed. Line 7 is connected straight though. If a shielded cable is used, ground the shield at one end to pin 1. Also, if you are not intending to use handshaking, you need not connect some of these lines — short pins 4 to 5 and 6 to 20 at both connectors and use a three wire cable connecting pins 2 to 3, 3 to 2 and pin 7 to pin 7.

A word of caution, however. Depending on the remote computer's clock rate, distance from the host computer, and the speed of the software, not using handshaking may cause loss of data. For simplicity, the programs addressed in this article do not utilize handshaking and operate satisfactorily over a 6 foot cable at 9600 baud. So if you notice a loss of data, try reducing the baud rate in both programs.

Cable connections are summarized in Figure 1.

| Interface Signal: | With Handshaking: | Without Handshaking: |
|-------------------|-------------------|----------------------|
| Chassis Ground | 1 ---- | 1 ---- |
| Data out to in | 2 ----> 3 | 2 ----> 3 |
| Data in from out | 3 <---- 2 | 3 <---- 2 |
| RTS to CTS | 4 ----> 5 | 4 \ / 4 |
| CTS from RTS | 5 <---- 4 | 5 / \ 5 |
| Signal Ground | 7 ---- 7 | 7 ---- 7 |
| DSR from DTR | 6 <---- 20 | 6 \ / 20 |
| DTR to DSR | 20 ----> 6 | 20 / \ 6 |

Figure 1

Software

Now that we have theoretically taken care of the hardware aspects of two terminal operation, let's tackle the software part. Both computers will need to run their own program. One needs to act as the master, letting the other know when a response is needed. The other terminal must act as the slave, providing information when needed and printing the information it receives on the screen. These terminals will be referred to as the "Master" and the "Slave".

The master's part of the software will be included in the game or utility program you want run. The slave's program is nothing more than an input/output routine. Let's review the slave's program first.

The slave's program, "TXCVR", is written in 8080 assembly language for speed — remember, I'm trying to run the communications at the normal H-89 baud rate of 9600. While ready made communications programs, such as MAPLE, work acceptably, they all have much more capability than is needed for games. They also usually have some small quirks that I would rather not have to program around, such as the inability to transmit ESCape codes, or differences in how the delete or backspace keys work. Additionally, you generally don't have the source code to modify these programs as necessary. No, I wanted a minimum size program that runs quickly and can be readily modified to work with a specific type of game program. TXCVR met these needs.

A very straightforward program, it begins with the usual equates. H/Z-89 I/O ports and standard baud rate divisors are also provided, though not all used. The program initializes the 330Q port for 9600 baud rate. Those of you wishing to use the 320Q port need to modify this area of the program accordingly. Also, if the baud rate turns out to be too high, that may need to be changed.

TXCVR uses several of the built-in functions of the CP/M operating system, the system calls, to perform the basic input/output (I/O) functions of the program, but some I/O functions required direct action. The system calls used here and others are described in detail in the CP/M 2.2 Interface Guide.

After printing a sign-on message, the program enters a loop consisting of the portions of the program labeled INPUT, where the slave's keyboard input is checked and any character found is displayed on the slave's screen, and PORTIN, where the modem port is checked for characters present. Characters found at either location are checked for a few special characters and printed on the screen, CONOUT routine. Characters are transmitted to the master terminal via the PORTOUT routine.

While system calls are great, they do not work for all circumstances. For example, according to the CP/M Interface Guide, function 3 for Reader Input reads the next character from the logical reader into register A. However, control does not return until the character has been read. Since we want this loop to continue if nothing is found ready at the modem port, we had to get more elementary and check the line status register for the modem port ourselves, then if a character were there, read it.

I had problems with function 4, Punch Output, that for unexplained reasons worked unreliably, so I used direct output there also. Maybe one of you can enlighten me as to why?

Finally, BKSP (backspace) and DELET (delete) routines were added to have the backspace and delete keys operate in their

normal conventions. Typing a CTRL-E on the slave ends the program and returns the slave terminal to CP/M.

The master terminal's program, "TERM2", is an MBASIC program, because of BASIC's popularity among game programmers. TERM2 is a simple program for testing and demonstration purposes. Portions of it would need to be placed into your game program, along with other changes that will become evident as we discuss TERM2.

Before going into a description of this program, however, let's discuss what the basic idea is behind this program. As I mentioned previously, the master terminal controls the flow of information between the two terminals. While it doesn't need information from the slave, it would be nice to allow the slave to still print information typed at its keyboard to be printed on its screen without interfering with the operation of the master. This information could be saved in a buffer, for transmission when the master desires it, or may be discarded when the master wants a specific answer, so only information typed since the request would be given to the master.

Additionally, in MBASIC there are two ways of transferring information to the slave, whereas it only has one method of data input from a port — the INP(I) function. The INP function will return the byte read from port I. I must be in the range 0 to 255, and for the modem port, 330Q, I=216 decimal. INP is the complementary function to OUT.

The OUT I,J statement is the first method for sending a byte to an output port. The integer expression I is the port number, and the integer expression J is the data to be transmitted. Both I and J must be in the range 0 to 255.

The second method to send information out a port is through the use of the LPRINT statement. Similar to the PRINT command, LPRINT is used to print data on the line printer. Oh, I can hear your wheels turning from here. No, let's not disconnect the printer and plug the slave in there — but you're close. No, let's not run the CONFIGUR program (or I/OMOD if you're running C.D.R. CP/M), unless we have to. There's a much simpler way. If you're thinking IOBYTE, you're right!

Turning to the CP/M Alteration Guide, under the section entitled, "The BIOS Entry Points", a discussion on the IOBYTE is hidden. If you recall from running the CONFIGUR program, CP/M uses four logical device names; CON:, RDR:, PUN:, and LST:. During CONFIGUR you can assign one of four physical devices to each of these logical devices. The logical device we are interested in is LST:, which is the principal listing device. It may be a TTY: — teletype machine, CRT: — the terminal's screen, LPT: — line printer, or UL1 — a user defined device using ETX/ACK protocol. Your choice is stored in a location in memory called the IOBYTE at 0003H.

At this location, the IOBYTE is split into four distinct fields of two bits each, called the CONSOLE, READER, PUNCH and LIST fields and is where the information chosen from CONFIGUR is stored. The MBASIC LPRINT command sends the information to the device given in the LIST field.

The LIST field consists of bits 6 and 7 (most significant bits) of the IOBYTE at location 0003H and its value can be in the range 0 to 3:

- 0 = Teletype device (TTY:)
- 1 = CRT device (CRT:)
- 2 = Line printer (LPT:)
- 3 = User defined list device (UL1:)

Knowing this, we can use the MBASIC command, PEEK(I), to read the value of the IOBYTE, or the command, POKE I,J, to change the value, thus changing the physical to logical device matching without running CONFIGUR.

So, why bother with all this when we have the simple OUT command, you ask? Well, by the time we prepare a string of characters to be sent via the OUT command, the process is slowed down considerably. This is sometimes evident during normal printer operations. If you construct a string variable by adding several other strings together, M1\$=NAME\$+" has \$" +STR\$(VALUE)+" to spend on "+FRUIT\$+"", for example, then sending it to the printer via the LPRINT M1\$ command, it is considerably slower than sending it via the command LPRINT NAME\$;" has \$";VALUE;" to spend on ";FRUIT\$;"". You can see this if you try each version of TERM2 to be discussed here.

TERM2 starts out initializing the modem port in a similar manner to the TXCVR program, then prints "STANDBY FOR TRANSMISSION." on the slave and "YOUR TERMINAL IS READY. TYPE A LINE FOR TRANSMISSION." on the master. The master has control. After typing a line and typing RETURN, the line is sent to the other terminal and the process is reversed. The slave terminal enters a line and upon a RETURN, control is passed back to the master.

As presently written, the program will use the LPRINT command to send the information to the slave terminal. The DOUT\$ variables for using the OUT command are placed in REM statements immediately following the LPRINT statements. Simply replace the LPRINT statements with the appropriate DOUT\$ variable to use the other method of sending information to the slave.

For those of you who followed my articles on PAINT.ASM in the March and April '86 issues of REMark, here's another example for displaying Paint Screens. This is an area where the use of LPRINT really shines. The screen is displayed on the slave just as fast as it is on the master. The use of the OUT command is so slow, especially while displaying to both screens at the same time, that it is better to display the screen first on the master; then run the routine again, sending the screen to the slave. It was this objectionable situation that caused me to investigate alternatives — such as LPRINT.

Incidentally, while I'm on the subject of PAINT.ASM, I had inadvertently left a remark in the XCAPE routine, given in the Part 1 article, that "A=0 for some reason!". I had found the reason in the CP/M Interface Guide, but forgot to remove the comment. CONSO is my name for Function 6, Direct Console I/O. Upon entry to function 6 with register E containing 0FFH, if no character is ready, A returns with the value of 00, otherwise A contains the next console input character. I hope I didn't cause some confusion with that comment.

Getting back to TERM2, line 190 is the POKE command discussed earlier. The number 41 sets the LST: device to TTY:. Line 1200 sets the LST: device back to LPT: with the number 169, just before we exit the program.

Because LPRINT uses the TTY: device, you will need to run the CONFIGUR program on the master terminal to configure the TTY: logical device to the 330Q port. It doesn't matter what baud rate you set while doing this, because TERM2 initializes the 330Q port to 9600 baud anyway. Other than for LPRINT, neither TERM2 nor TXCVR care what the logical and physical device match is, since each initializes the 330Q port and works directly with the port address for input and output. If LPRINT is not used, there-

fore, you won't need to reconfigure this port. I haven't been able to figure out how to make this change via my program rather than using CONFIGUR. Any ideas?

Line 5130 clears any characters entered on the slave's keyboard while it was not responding to the master. Line 390, using the INKEY\$ command does the same for the master terminal.

As you work TERM2 on the master terminal and TXCVR on the slave, you will notice one apparent bug. On the slave, when a carriage return is entered following a line to transmit, it doesn't automatically do a line feed. I did this intentionally. There are many circumstances where an automatic line feed would be very difficult to work around. It is better to do a plain carriage return on the slave, and let the master terminal sent a line feed back to the slave if one is needed. If you find this objectionable, TXCVR can easily be modified to do an automatic line feed in the same manner that the delete and backspace functions were fixed. In my game programs, though, I have found the automatic line feed is not needed.

Because the line lengths of the PAINT screens may exceed 80 characters, the commands WIDTH 255 and LPRINT WIDTH 255 are necessities. Again, both are reset to 80 character lengths prior to exiting the program. The program will exit on typing a CTRL-C, END or whatever you decide to test for in line 350.

The primary routines you should include in your BASIC programs are:

Lines 100-190: If you use CONFIGUR to initialize TTY:, UR1:, and UP1 to 9600 baud and 330Q, then this routine will be unnecessary.

Lines 1100-1190: If your program uses PAINT screens.

Lines 5000-5030: If you plan to use the OUT command, rather than LPRINT.

Lines 5100-5220: To input data from the slave terminal.

Intersperse LPRINT commands in your program as you would PRINT commands, and you're well on your way to twin screen fun. I wonder. If I connected four together, would I then have theater in the round?

I'd appreciate any comments, suggestions and criticisms anyone wishes to make, either through REMark magazine or directly to me. I'd be happy to answer questions if a self-addressed, stamped envelope is included.

Well, I hope I have managed to assist you into the world of two terminal operations. I've already converted some of my programs to two terminal operation and they sure beat the old way. I hope you have as much success. NOW, if I could just talk my wife into a third . . .

Equipment And Information Sources Mentioned

In addition to REMark Magazine & Heath/Zenith Computers:

SEXTANT Magazine
Sextant Publishing Co.
716 E Street, S.E.
Washington, DC 20003

MAPLE — Modem APpLication Effector

Author:
Dr. William C. Parke
1820 S Street, N.W.
Washington, DC 20009

Program:
 Heath Users' Group
 Hilltop Road
 St. Joseph, MI 49085

CP/M is a registered trademark of
 Digital Research Corporation
 P.O. Box 579
 Pacific Grove, CA 93950

C.D.R. Systems Incorporated
 7210 Clairemont Mesa Blvd
 San Diego, CA 92111

Also, a gremlin crept into the end of my March 86 REMark article on PAINT.ASM, Part 1. For some reason Floppy Disk Services was omitted. It is listed here.

Model TWOET 455 Dual Disk Drive System
 Floppy Disk Services
 39 Everett Drive Bldg. D
 Lawrenceville, NJ 08648

TXCVR Program

```

; VAGTS TXCVR PROGRAM
;
; For use with games using 2 terminals, this program
; is run in the second terminal.
;
; Following codes are from the CP/M INTERFACE GUIDE
CONIN EQU 1 ;CONSOLE INPUT
CONOUT EQU 2 ;CONSOLE OUTPUT
CONSIO EQU 6 ;DIRECT CONSOLE I/O
PRINT EQU 9 ;PRINT STRING
BDOS EQU 5 ;BDOS VECTOR
;
; H/Z-89 I/O PORTS
CRTP EQU 350Q ;SYSTEM CONSOLE PORT
LPTP EQU 340Q ;PRINTER PORT
MODP EQU 330Q ;MODEM PORT
TTYTYP EQU 320Q ;TTY - REMOVED IF PARALLEL BD
;
; BAUD RATE DIVISORS FOR 8250'S
B300 EQU 384
B600 EQU 192
B1200 EQU 96
B2400 EQU 48
B4800 EQU 24
B9600 EQU 12
;
START EQU $
LXI H,0 ;ZERO HL REG
DAD SP ;LOCATE STACK
LXI SP,STACK ;SET OUR OWN
PUSH H ;SAVE CP/M'S STACK
PUSH PSW ;SAVE CP/M'S FLAGS
;
; SET THE MODEM PORT
LXI H,B9600 ;PUT BAUD RATE DIVISOR IN HL
XCHG ;PUT IN DE
MVI A,083H ;SET DIVISOR LATCH ACCESS BIT
OUT MODP+3 ;TO A "1"
MOV A,E ;SET BAUD RATE LEAST SIG BYTE
OUT MODP
MOV A,D ;SET BAUD RATE MOST SIG BYTE
ANI 00FH ;CLEAR STOP BITS FLAG
OUT MODP+1
MVI A,003H ;RESET PORT TO DIVISOR LATCH ACCESS
OUT MODP+3 ;SET NO PARITY, 8BIT, 1STOPBIT
XRA A
OUT MODP+1 ;DISABLE PORT'S INTERRUPTS
IN MODP ;READ ANY GARBAGE
;

```

```

SIGNON LXI D,MSG1
CALL PMSG
;
INPUT MVI E,0FFH ;REQ INPUT CHAR
MVI C,CONSIO
CALL BDOS
ORA A ;CHECK IF ZERO
JZ MDATA ;NO CHAR WAITING
MOV E,A ;SAVE CHAR
CPI 005H ;CTRL-E? ENDS PROGRAM
JZ EXIT
CPI 008H ;BACKSPACE?
JZ BKSP1
PUSH D
CPI 07FH ;DELETE?
CZ DELET
POP D
CALL PORTOUT
MDATA CALL PORTIN
JMP INPUT
;
; CHECK THE MODEM PORT FOR CHARACTERS
PORTIN IN MODP+5 ;CHECK THE LINE STATUS REG
ANI 001H ;CHECK IF DATA READY
RZ
IN MODP ;READ DATA
MOV E,A
CPI 008H ;BACKSPACE?
CZ BKSP2
CPI 07FH ;DELETE?
CZ DELET
MVI C,CONOUT
CALL BDOS
JMP PORTIN
;
PORTOUT PUSH D ;SAVE CHAR
IN MODP+5 ;CHECK THE TXMTR HOLDING REG
ANI 020H ;CHECK IF EMPTY
JZ PORTOUT
POP D
MOV A,E
OUT MODP
MVI C,CONOUT
CALL BDOS
RET
;
BKSP1 MOV A,E
OUT MODP
CALL BKSP2
JMP INPUT
;
BKSP2 MVI C,CONOUT
CALL BDOS
DELET MVI E,01BH ;ESCAPE CHAR
MVI C,CONOUT
CALL BDOS
MVI E,04EH ;'N'
MVI C,CONOUT
CALL BDOS
MVI E,0
RET
;
PMSG PUSH PSW
PUSH H
MVI C,PRINT
CALL BDOS
POP H
POP PSW
RET
;
EXIT LXI D,MSG2
CALL PMSG
MVI C,CONIN
CALL BDOS ;A=CHAR
CPI 059H ;Y?
JZ EXIT1
CPI 079H ;y?

```

TERM2 Program

```

10 REM -----
20 REM TERM2 - TWO TERMINAL TEST AND DISPLAY PAINT SCREENS ROUTINE
30 REM -----
50 E$=CHR$(27): E1$=E$+"E": Y$=E$+"Y": BS$=CHR$(8):
   UP$=E$+"A"
60 CR$=CHR$(10)+CHR$(13)
70 DEF FNPOSIT$=Y$+CHR$(34+Y1)+CHR$(61+2*Y2)
100 REM Initialize the modem port.
110 BP=216: REM Base Port = 330Q
120 LBAUD=12: REM 128(300): 192(600): 96(1200): 48(2400):
   24(4800): 12(9600)
121 MBAUD=0: REM 1(300) and 0 for higher bauds.
130 OUT 219.131: REM Set divisor latch, BP+3
140 OUT 216.LBAUD: REM Set baud rate LS.
150 OUT 217.MBAUD: REM Set baud rate MS, BP+1
160 OUT 219.3: REM Set no parity, 8 bits, 1 stop bit, BP+3
170 OUT 217.0: REM Disable Interrupt Enable Register, BP+1
180 D=INP(216): REM Read any garbage.
190 POKE 3,41: WIDTH LPRINT 255:
   REM Whole line unnecessary if no LPRINT.
200 MSG1$="STANDBY FOR TRANSMISSION."
210 MSG2$="YOUR TERMINAL IS READY. TYPE A LINE FOR TRANSMISSION."
220 MSG3$="What PAINT data file do you wish displayed? "
300 PRINT "After each carriage return, the line will be
   sent to the other"
310 PRINT "Terminal. The other terminal will then be
   checked for INPUT."
320 LPRINT MSG1$
321 REM DOUT$=MSG1$+CR$: GOSUB 5010
330 PRINT MSG2$
340 INPUT "", MSG$
350 IF MSG$="END" THEN 1200: REM Use CTRL-C or type END
   to exit program.
360 IF MSG$="q" THEN FLAG=0: GOSUB 1110: GOTO 330
370 LPRINT MSG$
371 REM DOUT$=MSG$+CR$: GOSUB 5010
380 GOSUB 5110: PRINT DINB$
390 A$=INKEY$: IF A$="" THEN 320 ELSE 390
1000 REM Other terminal wants PAINT SCREEN.
1010 FLAG=1: LPRINT MSG3$: GOSUB 5130
1011 REM FLAG=1: DOUT$=BS$+MSG3$: GOSUB 5010: GOSUB 5130
1020 A$=LEFT$(DINB$,LEN(DINB$)-1): GOSUB 1120: RETURN
1100 REM GET NAME OF FILE WITH STORED DATA
1110 PRINT UP$:MSG3$: INPUT "", A$
1120 IF FLAG=1 THEN LPRINT E1$ ELSE PRINT E1$
1121 REM IF FLAG=1 THEN DOUT$=E1$: GOSUB 5010 ELSE PRINT E1$
1130 WIDTH 255: OPEN "I",#1,A$
1140 FOR X=1 TO 24
1150 IF EOF(1) THEN 1180
1160 LINE INPUT#1,Z$
1170 IF FLAG=1 THEN LPRINT Z$ ELSE PRINT Z$

```

```

JZ EXIT1
LXI D,MSG4
CALL PMSG
JMP INPUT
;
EXIT1 LXI D,MSG3
CALL PMSG
POP PSW ;GET CP/M FLAGS
POP H ;GET CP/M STACK
SPHL ;SET IT
RET ;RETURN TO CP/M
;
MSG1 DB 01BH,045H,' Welcome to TXCVR.',10,13
DB ' This program is used to enable this terminal (slave)
   to be',10,13
DB 'used with another terminal (master) that is running a
   program',10,13
DB 'designed for two terminal operation. All characters
   typed',10,13
DB 'on the slave will be displayed on its screen and sent
   to the',10,13
DB 'master, including ESCape characters.',10,13
DB ' CTRL-E ends the program.',10,13,'$'
MSG2 DB 01BH,078H,031H,01BH,059H,038H,025H
DB 'Do you wish to return to CP/M <Y/N>? '$'
MSG3 DB 01BH,059H,020H,020H,01BH,079H,031H,01BH,045H,' '$'
MSG4 DB 01BH,059H,020H,020H,01BH,079H,031H,' '$'
;
XFLAG DB 0
DS 64 ;SPACE FOR STACK
STACK EQU $ ;PUT STACK HERE
END START

```

```


1171 REM IF FLAG=1 THEN DOUT$=Z$+CR$: GOSUB 5010 ELSE PRINT Z$
1180 NEXT X: CLOSE#1: WIDTH 80
1190 RETURN
1200 WIDTH LPRINT 80: POKE 3,169: STOP
1201 REM STOP
5000 REM Output a line to the modem port (330Q)
5010 REM FOR I=1 TO LEN(DOUT$)
5020 REM OUT 216.ASC(MID$(DOUT$,I,1))
5030 REM NEXT: RETURN
5100 REM Input a line from the modem port (330Q)
5110 PRINT MSG1$: REM Data in buffer reset.
5120 LPRINT MSG2$
5121 REM DOUT$=MSG2$+CR$: GOSUB 5010
5130 DINB$="": D=INP(216): REM Get rid of any prior garbage.
5140 WAIT 221.1
5150 D=INP(216)
5160 IF D=8 AND LEN(DINB$)<1 THEN 5140
5170 IF D=8 THEN DINB$=LEFT$(DINB$,LEN(DINB$)-1): GOTO 5140
5180 IF D=127 THEN DINB$=DINB$: GOTO 5140
5190 IF D=64 THEN GOSUB 1010: GOTO 5120
5200 DINB$=DINB$+CHR$(D)
5210 IF D<>13 THEN 5140
5220 RETURN

```

*

LONG & LOUD!

Sideways & Banner Printing Utility for Dot-Matrix Printers



| | January | February | March | April |
|-----------------------|-----------------|-----------------|-----------------|-----------------|
| REVENUES | | | | |
| Widget Sales (units) | 3454 | 2345 | 3210 | 4322 |
| Price each | 1.28 | 1.28 | 1.28 | 1.28 |
| Widget Sales (\$) | 4417.12 | 3001.40 | 4115.20 | 5532.16 |
| Gadget Sales (units) | 4221 | 3432 | 3434 | 4547 |
| Price each | 2.34 | 2.34 | 2.34 | 2.34 |
| Gadget Sales (\$) | 9877.14 | 8077.48 | 8035.34 | 10684.78 |
| TOTAL REVENUES | 14354.26 | 11079.28 | 12150.74 | 14216.94 |

SHOUT YOUR MESSAGE IN A BANNER!

For any CP/M or MS/DOS computer (IBM compatibility is not required), just...

\$34.95

Special Offer: one MS/DOS and one CP/M version for only... \$49.95

We've improved our popular TWIST & SHOUT! package and given it a new name! **LONG & LOUD! Version 2.0** is easier to use and install, includes new typefaces in both LONG (four sizes) and LOUD (Times, Sans Serif, Olde English, Script and Symbols — in both upper and lower case). **LONG** lets you print out your spreadsheets (or any file) the long way (sideways) on your dot-matrix printer. No more cutting and pasting to put together a fragmented printout. **LOUD** prints giant banners in letters from two to eight inches high. Make banners & posters with ease!

PRESTO!™ Multi-Function Software Supercharger for CP/M NEW for Heath/Zenith

Profiles magazine wrote, "PRESTO still has the edge over Write Hand Man in features and general polish..." And now we've improved it even more! PRESTO adds features to any program you run. Just hit a special trigger key and PRESTO suspends your current program and opens a window on-screen. You can then call up a floating point calculator, a programmers calculator (hex, binary, octal, decimal), a notepad, a perpetual calendar, a Rolodex™, and perform screen dumps. Hit another key and you're right back where you left your original program.

PRESTO! (Version 3) uses almost 5K less memory than previous versions, yet includes great new features like:

NEW CP/M Commands: From within any program you can now do a directory, copy and rename files, erase files, and type files to the screen.

NEW Keyboard Macro Processor: Throw away SmartKey and XtraKey because PRESTO now includes its own key processor. The keys module includes powerful features like the ability to automatically load special key definitions for each program you use. One key can do the work of hundreds — a real time saver!

And best of all — the price is just **\$39.95**. Available for all Heath/Zenith CP/M computers using H19/89 type terminal. Versions are also available for Morrow, Osborne, Kaypro and Otrona. Specify computer and hard or soft sector format.

Rembrandt

Complete Business Graphics Toolkit™

Finally there's an easy and fun way to create graphics on your H/Z-89, H/Z-90, H/Z-100 (CP/M only) computer or any H/Z-19 equipped machine.

No extra hardware required! It works with a standard unmodified machine yet also supports the TMSI SuperSet ROM, and the Font19 Character ROM.

Freehand drawing: You can easily draw lines, boxes, circles and write on the screen in large characters. Full block operations are also supported — move, delete, fill, copy and more! Your graphic creations can be saved to disk and recalled at any time for further editing. Layout forms, design logos, draw diagrams and pictures. It's easy and fun to use.

Business graphics: REMBRANDT lets you create horizontal and vertical bar charts, pie charts and xy plots (scatter graphs). Use hand-entered data or read numerical data from virtually any source including dBase II, SuperCalc, MBasic, Wordstar and ASCII files.

Slide shows: Sequence your graphics on-screen using eleven cinematic special effects like wipes, fades and spirals. Produce electronic 'slide shows' without any programming.

Print your graphics: Print your graphic screens on most dot-matrix and daisy wheel printers. Interface with all word processors so that your reports can include charts, graphs or any graphic creation — intermixed with your text!

Compatible: It even reads, displays and prints *Ed-A-Sketch* files!

Affordable: Even with all of this power, REMBRANDT is available for an amazingly low price of... **\$39.95**

REMBRANDT runs on H/Z-89's, 90's, 100's and H/Z-19 equipped machines.

Other Stuff: MILESTONE Business Project Planner (CP/M and MS/DOS) \$99.95, MEDIA MASTER Disk Conversion (Z-100, PC-DOS) \$39.95, MEDIA MASTER PLUS Disk Conversion & CP/M Emulator \$59.95, ACCELERATE 8/16 including V20 chip \$99.95

We accept VISA, MASTERCARD and AMERICAN EXPRESS

Order by mail or call our 24 hour toll free order line from anywhere in the US or Canada:

800-628-2828 (Extension 918)

Technical questions, orders: 818-716-1655 (9-5 PST)

Add \$4 per order postage/handling. Overseas, add \$12. US funds only. CA residents add 6% tax (LA County 6.5%)

TECHNOLOGIES, INC.
22458 Ventura Blvd., Suite E
Woodland Hills, CA 91364

A Software Clock/Calendar CP/M-80

Part 2

Rodney Horner
4871 Hialeah Drive
Plum Borough, PA 15239

Conclusion And Other Undocumented Items

This is the second article that describes how to use an undocumented clock/calendar that is part of Heath CP/M-80. The first article described how to make the clock work and how to use it with BASIC and Fortran. I will describe how to use the clock/calendar with Microsoft COBOL-80.

I assume that the work from part one of the article has been done and the clock is running with the correct time and date. You will remove the existing ACPDAT.REL module from your COBOL run-time library and replace it with a working program.

Using The New ACPDAT Module

COBOL is an industry standard language that has a predefined method for inputting the date, time and day. The program ACPDAT.MAC is written to meet the ANSI standard and get the correct DATE, TIME and julian DAY of the year into your programs. The format of the ANSI standard statements follow.

To input the time:

Where the variable SYSTEM-TIME is defined like . . .

```
01 SYSTEM-TIME.
   05 SYS-HOUR      PIC 99.
   05 SYS-MINUTE    PIC 99.
   05 SYS-SECOND    PIC 99.
   05 SYS-HUNDREDTHS PIC 99.
```

the procedure division statement would be

```
ACCEPT SYSTEM-TIME FROM TIME
```

To input the date:

Where the variable SYSTEM-DATE is defined like . . .

```
01 SYSTEM-DATE.
   05 SYS-YEAR      PIC 99.
   05 SYS-MONTH     PIC 99.
   05 SYS-DAY       PIC 99.
```

the procedure division statement would be

```
ACCEPT SYSTEM-DATE FROM DATE
```

To input the Julian day of the year:

Where the variable SYSTEM-DAY is defined like . . .

```
01 SYSTEM-DAY.
   05 SYS-YEAR      PIC 99.
   05 SYS-JULIAN-DAY PIC 999.
```

the procedure division statement would be

```
ACCEPT SYSTEM-DAY FROM DAY
```

As a language extension, I implemented the subroutines DATE and DAY. They are similar to the ANSI ACCEPT from DATE and DAY, except that the routines must be called and the returned SYS-YEAR is a four digit year, instead of only the last two digits of the year.

I would recommend that the subroutines be used instead of the ACCEPT statements whenever comparisons are required on dates that will cross over the change in the century. The format of the subroutine statements follow.

To input the date:

Where the variable SYSTEM-DATE is defined like . . .

```
01 SYSTEM-DATE.
   05 SYS-YEAR      PIC 9999.
   05 SYS-MONTH     PIC 99.
   05 SYS-DAY       PIC 99.
```

the procedure division statement would be

```
CALL "DATE" USING SYSTEM-DATE
```

To input the Julian day of the year:

Where the variable SYSTEM-DAY is defined like . . .

```
01 SYSTEM-DAY.
   05 SYS-YEAR      PIC 9999.
   05 SYS-JULIAN-DAY PIC 999.
```

the procedure division statement would be

```
CALL "DAY" USING SYSTEM-DAY
```

Installing The New ACPDAT Module

The COBOL standard requires the use of the hundredths of a second. The hundredths of a second is not available from the software clock. My ACPDAT.MAC program will use 00 for the hundredths or will generate somewhat random numbers for the hundredths, depending on an option in the assembly of the ACPDAT program. See the ACPDAT.MAC source listing for the line near the beginning of the program with the label SIMUL. If SIMUL is EQUated with 0, the hundredths of a second will always be 00. If SIMUL is EQUated with 1, the hundredths of a second will be returned as the sum of the current minutes and seconds. This sum will give the hundredths a somewhat random appearance.

Enter the ACPDAT.MAC program with an editor and assemble it with M80. After it is assembled with no errors, recheck your source code. The next step is time consuming. You do not want to do it over.

The original ACPDAT module in your COBOL run time libraries must be replaced with the new ACPDAT module. This should be done to both the COBLBX.REL and COBLIB.REL libraries. Do the COBLBX.REL library first. It is shorter.

In the next paragraph, the 'Work library disk' is the disk that you normally use to hold your COBLIB.REL or COBLBX.REL libraries when you link your COBOL programs with L80. If you are using an old version of COBOL, LIB80.COM may be called LIB.COM.

This is the procedure I used to do this. On Drive A put your ACPDAT.REL file created by the preceding assembly. PIP the library, COBLBX.REL or COBLIB.REL, from your 'Work library disk' to Drive A, then delete it from the library disk. Now PIP LIB80.COM to your 'Work library disk'. The 'Work library disk' should be on Drive B. The following command lines will copy your COBOL library back to your 'Work library disk' until it reaches the original ACPDAT module. When the original ACPDAT module is detected, your new ACPDAT module will be copied instead. Then, the rest of the original library will be copied to your 'Work library disk'. Your 'Work library disk' should be back to normal, except that it contains a revised library.

```
B:LIB80 B:NEWLIB=A:COBLBX<..ACPDAT-1>,A:ACPDAT,  
A:COBLBX<ACPDAT+1..>/E  
REN B:COBLBX.REL=B:NEWLIB.REL
```

Repeat the preceding operation for the COBLIB.REL library after testing the COBLBX.REL module with a program. When you are updating the COBLIB.REL, substitute COBLIB for COBLBX in the command lines above.

To test the new libraries, just use the new statements in one of your own COBOL programs. Compile and link the program the normal way. You will notice that the hundredths of a second are either always 00 or 'Random numbers' simulated by my ACPDAT.MAC program.

I assume all went well and your COBOL compiler was successfully upgraded.

The Program

I will not give a complete discription of the program; however, I will cover some of its main features. Persons familiar with 8080 assembly language may wish to review the listing of the ACPDAT module. The interface to these modules is described in the Microsoft COBOL-80 manual.

Each time an ACCEPT from DATE, DAY or TIME statement is executed, the ACPDAT.MAC program reads the date or time from memory, converts the date to ASCII characters and moves the ASCII string to 'Program memory'. Execution then returns to the main program.

The interface to the routines ACTIME, ACDATE and ACDAY are all supplied by Microsoft Corporation. The routines are all similar. The starting memory location of the address to receive the date and time is supplied to the routines in the 'HL' register. That address will be the starting address of either the working-storage, linkage item or file-buffer item of where you are accepting the data into. Each of the routines performs a CALL to a subroutine which actually converts the date or time read from the operating system to their ASCII representation. They then use a supplied MOVE subroutine to move the finished ASCII string containing the date and time to the address in your program.

The subroutines DATE and DAY are the same except they have externally defined entry points instead of the being 'Built into COBOL' as the other routines.

The routines INTIME, INDATE and INDAY read the system time, date and day and convert them from the binary representation to an ASCII or usage DISPLAY representation. Each of the routines is fairly simple in its operation except for INDAY which works out the julian day of the year. Each of the paragraphs that start with 'CK' form a routine which converts the binary day of the year in the 'HL' register to ASCII. While the 'CK' routines are not long they are difficult to follow and will only work in this limited application. If any readers have a better 8080 sixteen bit to ASCII routine send it into REMark.

You will notice in the listing at labels DAYX and DATEX that the first two digits of the year, '19', are coded into the program. Those of you still using your 89s at the turn of the century should change the '19' to a '20' and recompile everything. An alternative would be to modify the ACPDAT program to automatically load the first two digits of the year. An algorithm to determine that the century changed could be as follows. Normally leave the '19' in place; however, if the year is less than '86', then move a '20' into DAYX and DATEX.

Other Undocumented Items

The software manuals supplied by Heathkit are quite good, but I found some loose ends. One such loose end was in the COBOL manual which states 'If you have . . . access to a system clock . . .' and in the CP/M BIOS listing which shows a 'Time of day handler'. I tied those items together to produce these two articles. The result is an improvement to the COBOL compiler and a clock/calendar accessible to languages which support the PEEK statement.

I suggest that HUGgies send any undocumented features to REMark for publication, perhaps for a new regular series called 'Undocumented Features'. This was previously done to produce the successful ongoing series of articles 'My Favorite Subroutines'.

The 'Undocumented Feature' column could contain any features you find in software that are useful, but are not documented. Perhaps there is something that works different in real life than it is described. Have you ever used a feature exactly as described, but it failed to work or only works sometimes? Maybe other HUGgies could send in a solution to those problems.

The following contains my entire backlog of undocumented features:

The Fortran-80 compiler (Version 3.4) supports the old 'EXIT' subroutine. 'EXIT' was used to stop a Fortran program. It is equivalent to the present 'STOP' statement. I use the 'EXIT' subroutine instead of 'STOP' because it does not display that annoying 'STOP' message on the screen as does the more traditional 'STOP' statement in this version of Fortran. To use the 'EXIT' subroutine simply use

```
CALL EXIT
```

where you would normally use

```
STOP
```

No parameters are required for the 'EXIT' subroutine.

The COBOL-80 compiler (Version 4.65) supports the syntax of the COBOL 'RERUN' and 'SEGMENT-LIMIT' statements. While the statements do not perform any function in COBOL-80, it is nice to know they are there if you need them to satisfy the syntax requirements for a school programming project or any such similar use. More importantly, the word 'RESTART-FILE' is a reserved word which is not listed in the reserved word list. While the implementation may vary, the Rerun facility is used to be sure that records are actually written, rewritten or deleted at some periodic interval in processing and not just laying in a file buffer. A checkpoint file is kept to indicate the last time this was successfully done. This is for emergency file recovery purposes. The segment-limit is the segment number of the highest numbered procedure division section which will permanently execute in memory. In COBOL-80 the segment-number is always equal to 49 regardless of the number actually entered. The syntax of the statements follow:

In the OBJECT-COMPUTER paragraph:

```
SEGMENT-LIMIT IS segment-number
```

In the I-O-CONTROL paragraph:

```
RERUN ON RESTART-FILE [EVERY record-count RECORDS OF file name]...
```

The Cobol Users Guide, Appendix D describes 'Extensions For File Handling Under CP/M-80'. The descriptions for using the four subroutines 'DSKRES', 'FILENQ', 'KILL' and 'RENAM' are fine except the STATUS item is not described. The STATUS appears to be a two byte alphanumeric item similar to the COBOL file status which may be entered in the FILE-CONTROL paragraph in the environment division. Then, if the operations are successful, '00' is returned in the STATUS variable. When 'File-name' or 'Old-filename' does not exist, '30' is returned in the STATUS variable. When 'New-filename' exists, '50' is returned in the STATUS variable. If the action of any of these subroutines causes a BDOS error, such as a disk not in its drive or a non-existent drive is referenced, the CP/M BDOS will intercept the error and your program will end. These subroutines will accept, but not properly use 'wild cards' in file names.

This covers all of the undocumented features that I found. Do you have any?

ACPDAT.MAC

```

TITLE ACPDAT
: (ACPDAT.MAC)
: REPLACEMENT ACPDAT MODULE FOR
: MICROSOFT COBOL-80 & HEATH CP/M-80
```

```

ENTRY $ACPD,DATE,DAY
EXT $EVAL,$GETOP,$FLAGS
EXT $ESKEY,$MOVE

; SUBSTITUTE THE ADDRESS GIVEN BY THE
; FINDMEM PROGRAM FOR THE #### BELOW

SEC EQU 0####H ; SECONDS
MIN EQU SEC+01H ; MINUTES
HOU EQU SEC+02H ; HOURS
DA EQU SEC+03H ; DAY
MON EQU SEC+04H ; MONTH
YEA EQU SEC+05H ; YEAR

SIMUL EQU 1 ; 0 HTHS = 00
; 1 HTHS RANDOM

$ACPD: PDP H ; MICROSOFT CODE
INX H
MOV A,M
INX H
ANI 7
STA $FLAGS
CALL $EVAL
LDA $FLAGS
CPI 2
JM ACDATE
JZ ACDAY
CPI 4
JC ACTIME
JZ ACLINE
ACESC: XCHG
LHLD $ESKEY
XCHG
ACESC1: MOV M,D
INX H
MOV M,E
JMP $GETOP
ACLINE: LXI D,3030H ; END
JMP ACESC1 ; MICROSOFT CODE

; ANSI ACCEPT...FROM TIME

ACTIME: PUSH H ; SAVE DEST
CALL INTIME ; READ TIME
POP H ; RECALL DEST
XCHG ; 'D' IS DEST
LXI H,THH ; 'H' IS SOURCE
LXI B,08H ; 'B' IS LENGTH
CALL $MOVE ; MOVE SUBROUTINE
JMP $GETOP ;

; ANSI ACCEPT...FROM DAY

ACPDAT.MAC
ACDAY: PUSH H
LXI H,DAYY
CALL INDAY
POP H
XCHG
LXI H,DAYY
LXI B,05H
CALL $MOVE
JMP $GETOP

; DAY SUBROUTINE, RETURNS YYYYDDD

DAY: PUSH H
LXI H,DAYY
CALL INDAY
POP H
XCHG
LXI H,DAYX
LXI B,07H
JMP $MOVE

; ANSI ACCEPT...FROM DATE
```

```

ACDATE: PUSH H
        CALL INDATE
        POP H
        XCHG
        LXI H,DYY
        LXI B,06H
        CALL $MOVE
        JMP $GETOP

```

```
; DATE SUBROUTINE, RETURNS YYYYMMDD
```

```

DATE:   PUSH H
        CALL INDATE
        POP H
        XCHG
        LXI H,DATEX
        LXI B,08H
        JMP $MOVE

```

```
; AFTER 1999, CHANGE THE '19' TO '20'
```

```

THH:   DS 02H ; TIME
TMM:   DS 02H
TSS:   DS 02H
TFF:   DB '00'

```

```

DAYX:  DB '19' ; JULIAN
DAYY:  DS 02H
DA1:   DS 01H
DA23:  DS 02H

```

```

DATEX: DB '19' ; DATE
DYY:   DS 02H
DMM:   DS 02H
DDD:   DS 02H

```

```
; SUBROUTINE INTIME, INPUTS ACTUAL TIME
```

```

INTIME: LDA HOU ; HOUR
        LXI H,THH
        CALL DISPL
        LDA MIN ; MINUTE
        LXI H,TMM
        CALL DISPL
        LDA SEC ; SECOND
        LXI H,TSS
        CALL DISPL
        IF SIMUL ; HUNDREDTHS
        LDA SEC ; SIMULATION
        LXI H,MIN
        ADD M ; MIN + SEC
        CPI 100D
        JC REDX
        SUI 100D ; > 99, SUB 100
REDX:  LXI H,TFF
        CALL DISPL
        ENDIF
        RET

```

```
; SUBROUTINE INDAY, INPUTS JULIAN DAY
; COMPUTES JULIAN FROM CURRENT DATE
```

```

INDAY: LDA YEA ; YEAR
        LXI H,DAYY
        CALL DISPL
        LXI H,0D ; ZERO 'HL'
        MVI D,0D ; ZERO 'DE'
        LDA MON
        CPI 1D ; JAN
        JZ MON0
        CPI 2D ; FEB
        JZ MON1
        CPI 3D ; MAR
        JZ MON2
        CPI 4D ; APR
        JZ MON3
        CPI 5D ; MAY
        JZ MON4

```

```

CPI 6D ; JUN
JZ MON5
CPI 7D ; JUL
JZ MON6
CPI 8D ; AUG
JZ MON7
CPI 9D ; SEP
JZ MON8
CPI 10D ; OCT
JZ MON9
CPI 11D ; NOV
JZ MON10

```

```
; ADD DAYS FOR ALL PRECEEDING MOS
```

```

MVI E,30D
DAD D ; NOV'S DAYS
MON10: MVI E,31D
        DAD D ; OCT'S DAYS
MON9:  MVI E,30D
        DAD D ; SEP'S DAYS
MON8:  MVI E,31D
        DAD D ; AUG'S DAYS
MON7:  MVI E,31D
        DAD D ; JUL'S DAYS
MON6:  MVI E,30D
        DAD D ; JUN'S DAYS
MON5:  MVI E,31D
        DAD D ; MAY'S DAYS
MON4:  MVI E,30D
        DAD D ; APR'S DAYS
MON3:  MVI E,31D
        DAD D ; MAR'S DAYS
MON2:  CALL LEAP
        DAD D ; FEB'S DAYS
MON1:  MVI E,31D
        DAD D ; JAN'S DAYS
MON0:  LDA DA ; DAY'S IN THE
        MOV E,A ; CURRENT MONTH
        DAD D

```

```
; 'HL' CONTAINS THE BINARY JULIAN DAY
; PIC 999 ASCII CONVERSION FOLLOWS
```

```

CK300: MOV A,H ; IF > 299 THEN
        CPI 1D ; H = 1, DEC 256
        JNZ CK200 ; AND
        MOV A,L ; L > DECIMAL 44
        CPI 44D
        JC CK200 ; IF REALLY > 299
        SUI 44D ; SUB 300 FROM A
        LXI H,DA23 ; USE 2 DIGIT
        CALL DISPL ; ASCII CONVERSION
        MVI A,'3' ; FIRST DIGIT = 3
        STA DA1
        RET

```

```

CK200: MOV A,L ; IF > 199
        CPI 200D ; AND < 256
        JC CK100
        SUI 200D
        LXI H,DA23
        CALL DISPL
        MVI A,'2'
        STA DA1
        RET

```

```

CK100: MOV A,L ; IF > 99
        CPI 100D
        JC CK000
        SUI 100D
        LXI H,DA23
        CALL DISPL
        MVI A,'1'
        STA DA1
        RET

```

```
CK000: MOV A,H ; IF H = 1 NOW
```



```

CPI    1D      ; ACTUAL VALUE
JZ     CK256   ; 256 TO 299

MOV    A,L     ; IF > 0
LXI    H,DA23
CALL   DISPL
MVI    A,'0'
STA    DA1
RET

CJ256: MOV    A,L     ; IF 256 TO 299
      ADI    56D     ; ADD 56 TO VALUE
      LXI    H,DA23
      CALL   DISPL
      MVI    A,'2'
      STA    DA1
      RET

      ; LEAP YEAR CHECK
LEAP:  MVI    E,28D  ; FEB = 28
      LDA    YEA
      CPI    0      ; YEA = 0 NO LEAP
      JZ     NLEAP
      ORI    1111100B ; LAST BYTES 0
      CPI    1111100B ; THEN NO LEAP
      JNZ    NLEAP
      MVI    E,29D  ; ELSE FEB = 29

      ; SUBROUTINE INDATE INPUTS ACTUAL DATE
INDATE: LDA    YEA   ; YEAR
      LXI    H,DYY
      CALL   DISPL
      LDA    MON    ; MONTH
      LXI    H,DMM
      CALL   DISPL
      LDA    DA
      LXI    H,DDD  ; DAY
      CALL   DISPL
      RET

      ; CONVERTS VALUE IN A TO PIC 99 DISPLAY
      ; RESULT TO ADDRESS IN 'HL'
DISPL: MVI    C,0
DISP1: SUI    10D
      JC     DISP2
      INR    C
      JMP    DISP1
DISP2: ADI    10D
      PUSH  PSW
      MOV    A,C
      ADI    30H
      MOV    M,A
      POP    PSW
      ADI    30H
      INX    H      ; INCREMENT HL
      MOV    M,A
      RET

END

```

*

**Are you reading
a borrowed copy of REMark?
Subscribe now!**

ANALYTICAL PRODUCTS 805/688-0826
213 Teri Sue Lane Buellton, CA 93427

We Support the H89!

4MHz mod for H89

Easy to install plug-in module. No trace cutting or soldering required. Includes CP/M software.

Assembled and tested — specify disk format . . \$45
HDOS software \$ 5

6MHz mod for H89

Provides maximum high speed performance presently available for the H89 or H89A.

Assembled and tested — specify disk format . . \$59

REP3 — Automatic Key Repeat

Modernize your H89 or H19. Hold any key down for half a second and the key begins repeating. Simple plug-in installation.

Kit \$35
Assembled \$45

TIM2 — Real Time Clock

Installs in the left hand expansion slots of the H89. Includes battery backup. Requires soldering 4 wires to the CPU board.

Kit \$65
Assembled and tested \$75
Software on disk — specify format \$10

DATESTAMPER

Product of Plu*Perfect Systems. Provides automatic time and date stamping for CP/M 2.2 files.

For CP/M — specify disk format \$45

ZCPR3 for H89 and H8

We are licensed by ECHELON to distribute the Z-System. Includes ZCPR3 system, ZRDOS+, supporting utilities, ZCPR3 manual by R Conn, ZRDOS manual and additional instructions. Comes installed with a bootable disk ready to run.

Z-System — specify format and hardware \$98

CDR Super RAM 89

Main board w/512k RAM \$239
System w/Megabyte RAM \$369
Expansion Board w/512k \$133
Real time clock option \$ 45
SCSI hardware/software option \$160

CDR Controller for H89

FDC-880H — include CP/M s/n \$349
CDR/H17 adaptor cable \$ 20

CALL OR WRITE FOR CATALOG PRICES SUBJECT TO CHANGE

Terms: Check or Money Order — Visa, M/C — C.O.D.
Add \$3 per order for shipping and handling
California residents add 6% tax



WILDFIREtm

An INCREDIBLE PERFORMANCE IMPROVEMENT from Software Wizardry

Wildfire has spirit! It will spur your Z-151/161 to run faster than the 8mHz Z-158! Wildfire uses the reliable NEC-V20 chip. It outdoes the 80286 processor chip additions for compatibility with the IBM(tm)PC and costs up to \$600 less!

The heart of Wildfire is a daughter board that mounts on your processor board, saddled between it and the video board. Several higher-speed chips are included to replace socketed chips on your processor board. A high speed/low speed switch that mounts on your machine's front panel allows you to trot at normal speed, or break into a real gallop when you really want to ride!

Software Wizardry brings you Wildfire, complete with front panel, installation instructions, and reliable daughter-board design, for only \$249!

If you order now, you can try Wildfire with a 15-day guarantee. If not satisfied, you can return it within 15 days for a full refund.

Contact your local dealer, or order direct from Software Wizardry by calling (314)724-1738. Dealer inquiries invited.



1106 First Capitol Dr.
St. Charles, MO 63301
(314)724-1738

Also from Software Wizardry...

For the Z-151/152/161

RamPal

The Original Memory Upgrade Kit for only... **\$39.95**

RamPal allows you to put up to 640K RAM on your standard memory board. This simple one-chip replacement requires no modification to existing hardware. One bank of 41256 chips is required, two for full 640K memory.

Z-100 Upgrade Package Deal

Reliable Memory/Speed Upgrade for only... **\$299**

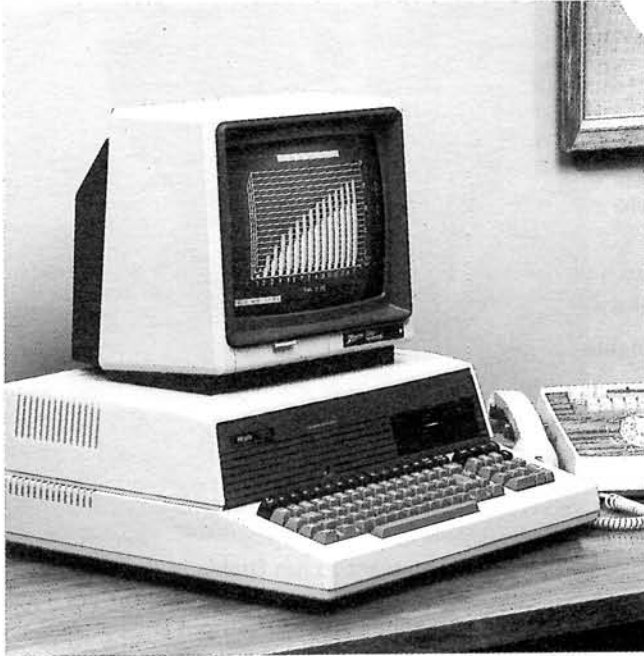
Our special package includes the RamPal 100 Memory Upgrade, 27 RAM chips, and the 8mHz Speed Upgrade. Now you can have compatibility at a price you can afford. Order part number FCC-MEM100FULL.

8mHz Speed Upgrade

Reliable Z-100 Speed Upgrade for true 8mHz Operation. **\$149**

This upgrade will bring your Z-100 up to the operating speed of the latest Z-100 version. Plus, you can switch down to 5mHz for programs that require it. Includes over 25 chips, circuit board, and back panel for 5 or 8mHz operation.

ZPC Update #15



Pat Swayne
HUG Software Engineer

This is the fifteenth 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 installment of ZPC Update, I will present patches for a bug discovered in ZPC versions 2 and 2.1. I will also present patches for PC Write version 2.6, Enable version 2, patches and instructions for a number of releases of Generic CADD not covered previously, a new patch for Javelin version 1.1 (for a different release of that version than was patched in the February Update), and instructions for using the Turbo Editor Toolbox under ZPC. For those of you who have wondered why some programs such as Framework II that are supposed to run in color do not, I will present an explanation of the problem and a solution. This article will also include some listings somehow left out of the February Update. But first, I have a few things to say on ZPC in general.

Time Out For An Editorial

Since ZPC version 2 came out, I have released only one non-ZPC HUG disk (Debug Utilities, 885-3038-37). That is not good. I have some ideas in my head that need to get out, and the only way I can find the time to work on them is to slow down on ZPC. So unless I discover a great breakthrough that will considerably improve the present offering, I will not work on a new version of ZPC.

The main thing that takes up my time around here these days, and prevents me from moving on, is having to answer phone calls and letters about ZPC and other subjects. I am happy to help when I can, but I am not magic, and do not have the answers to every question, or the time to do everything on an individual basis. Here

are some things that you should keep in mind if you are thinking of calling me.

I am not a hardware expert. If you built the ZHS circuit and can't get it to work, or if you modified your H/Z-100 to hold 768k of RAM or run at 8 MHz and it doesn't work properly, I may not be able to help. (I will have more to say on the ZHS circuit later.)

I do not have the time to tutor people individually. If you cannot figure out how to run ZPC from the information in the manual, or how to run a particular program, or how to use DEBUG to patch programs, etc., please try to figure out these things yourself, or try to get local help (a friend who knows). Probably 98 or 99 per cent of the people who call asking for tutorial help already have the answer in printed form in one of their manuals, a REMark article, etc. Never throw away your old REMarks!

The ZPC approach to IBM compatibility is a hobbyist approach, for those who don't mind doing a little hacking to get things working. It was never intended to be a turnkey system. If you want a turnkey system, you should consider getting a hardware PC emulator, or a separate PC-compatible computer. People who can't afford to buy a new computer or expensive adapter at the drop of a hat (like me) should compensate by learning to hack a bit. It's the same in any area of life. If you can't afford to buy your way through, you must LEARN (read, study) to make do with what you have.

Getting back to the ZHS circuit, let me state that the original concept of ZPC was a software-only approach to IBM compatibility. Perhaps I should re-emphasize this, in view of what I have heard about people trying to get ZHS boards to work. Some people have had trouble getting any kind of board to work, whether my design or the Scottie board. If your computer has been modified to run at

a faster speed, but not by using the approach presented in REMark for older motherboards, or by upgrading a new motherboard via the HA-108 kit, you may not be able to get a ZHS board to work. If you have used the REMark method or the HA-108 kit, you may be able to get the board to work by increasing your I/O wait states. But even if you can't get the board to work, there is one modification that could be beneficial to every ZPC user, and that is the single jumper wire on the video board that is described in the article supplied with ZPC.

If you use ZPC without hardware support, it means that you will have to patch most of the programs that you will want to run under it. I will probably continue publishing patches for programs in REMark as I work them out. But I KNOW that patches have been developed for far more programs out there by other folks than I have presented here. WOULDN'T IT BE NICE if people would send those patches in for publication here, as Paul Herman did with the patches for Generic CADD (later in this article), and as Timothy Wilmarth did for Clipper (in last month's "Buggin' HUG" column).

I have one final comment for this editorial section, concerning the price of ZPC. ZPC Version 2 is \$60.00, and the upgrade disk is \$20.00, which means that getting the "latest" version will cost \$80.00. If you think that is a bit high, remember that a while back a PC emulator program called IBEEm was available that sold for \$79.95, and did not provide anywhere near the amount of PC compatibility provided by ZPC. And in any case, you may not need the upgrade disk. But consider that if you threw away all of your last year's REMarks, and suddenly you need the patch for some programs that were published in Update articles last year, you can pay \$25.00 for the bound volume for last year's REMarks, or you can pay \$20.00 for the upgrade disk, which contains the patches you need (and you don't have to type them in). Maybe at the end of this year I will offer an upgrade to the upgrade disk. We would have you send in your old upgrade disk with \$5.00, and you would get a new one with all of this year's patches added to the previous stuff. Nancy will hate me, but at least it is a way to make the upgrade disk more valuable.

ZPC Bug Patches

HUG member John Collins discovered a bug in the COM port emulation section of ZPC versions 2 and 2.1. His letter will appear in "Buggin' HUG", and will explain how to correct the problem in the assembly source code. For those of you who do not want to mess with the source code, I have developed a patch that you can install with PATCHER. To make the patch, add the appropriate lines from the following to your PATCHER.DAT file.

```
ZPC version 2.0.3 COM patch
Insert your ZPC disk.
ZPC.COM
1ED9,E9,FA,18,90
37D6,59,32,E4,E9,DF,E6
z
ZPCSM version 2.0.2 COM patch
Insert your ZPC disk.
ZPCSM.COM
1C9A,E9,4C,16,90
32E9,59,32,E4,E9,8D,E9
z
ZPC version 2.1.3 COM patch
Insert your ZPC disk.
ZPC.COM
1ED9,E9,05,1B,90
39E1,59,32,E4,E9,D4,E4
z
ZPCSM version 2.1.2 COM patch
Insert your ZPC disk.
```

```
ZPCSM.COM
1C9A,E9,56,18,90
34F3,59,32,E4,E9,83,E7
z
```

Make the patch to your copy of ZPC.COM or ZPCSM.COM on your ZPC system disk or partition using PATCHER as instructed in the ZPC manual.

PC Write Version 2.6

To run PC Write version 2.60 under ZPC, add these lines to your PATCHER.DAT file:

```
PC WRITE version 2.60
Insert the disk containing ED.EXE.
ED.EXE
503F,90,90,90,90,90
5048,90
505B,90
z
```

Patch the program by following the instructions for PATCHER in your ZPC manual.

Enable Version 2

To use Enable version 2 under ZPC, first install the program as directed in the enable manual. When you get to the video driver installation section, select the first driver from the menu (Standard IBM). When you have finished the installation procedure, a file called V000.COM will have been created on your Utility disk (if you are running from floppies) or on your enable partition. This file must be patched if you want to run Enable in the default video mode and use graphics. If you do not need graphics (for graphs), you can run Enable in mode 7 (PC 7) without any patches.

To patch Enable, copy these lines to your PATCHER.DAT file.

```
ENABLE version 2.0 (1-09-87)
Insert the Utility disk containing V000.COM.
V000.COM
A73,90
161D,90,90,90,90,90,90,90,90,90,90,90
1766,90,90,90,90,90,90,90,90,90,90,90
178F,90,90,90,90,90,90,90,90,90,90,90
180E,90,90,90,90,90,90,90,90,90,90,90
BF1,0,0
BF4,B0
z
```

Now you can patch the V000.COM file to make Enable run in the default ZPC video mode.

Generic CADD

It seems that there are a number of releases of Generic CADD version 2 and version 2.02, and the code that needs patching in each release is different. Paul F. Herman of Software Graphics Tools sent me a number of patches for some versions that I do not have access to, and I will present them here. If the date on your CADD .EXE file matches the date on one of these patches, it should work for you. Here are the the PATCHER.DAT lines for the new releases that we have.

```
GENERIC CADD V. 2.02 11-26-86
Insert the disk containing CADD.EXE.
CADD.EXE
146E3,B0
1474D,B0
1478A,B0
147F3,B0
14844,0,0
148AC,0,0
14927,0,0
z
```

GEN. CADD w/DE-1 V. 2.02 12-15-86
 Insert the disk containing CADD.EXE.
 CADD.EXE
 1234C,B0
 123B6,B0
 123F3,B0
 1245C,B0
 124AD,0,0
 12515,0,0
 12590,0,0
 z

GEN. CADD w/AD V. 2.02 10-28-86
 Insert the disk containing CADD.EXE.
 CADD.EXE
 13468,B0
 134D2,B0
 1350F,B0
 13578,B0
 135C9,0,0
 13631,0,0
 136AC,0,0
 z

G. CADD w/DE-1,AD V. 2.02 12-15-86
 Insert the disk containing CADD.EXE.
 CADD.EXE
 1236C,B0
 123D6,B0
 12413,B0
 1247C,B0
 124CD,0,0
 12535,0,0
 125B0,0,0
 z

After you add these patches to your PATCHER.DAT, you can select the appropriate one and make the patch.

Paul Herman discovered a problem common with all versions of Generic CADD that I had not seen because my system is modified. If you run it on an unmodified system, the screen will be blank when it exits, and you will not be able to see any characters you type, etc. The condition can be cleared up by issuing an Escape-z sequence, so one way to fix the problem is to run CADD from a batch file that issues the sequence. To create the batch file, enter

```
COPY CON GC.BAT
```

at the system prompt, and enter these lines:

```
PC
CADD
Z100
ECHO <F8>z
```

Type Control-Z when you have typed these lines. The <F8> means to press the F8 key. That is the way you enter an escape character while you are under the control of DOS. It will be echoed as “^[".

The problem is caused by some writes to some illegal ports, so if you have modified your video card with the single jumper wire as explained in the article supplied with ZPC, you will not experience the problem. You can also patch out the illegal ports, but I only have the patches for two releases: the release that was patched in the Dec. '86 Update article, and the release dated 11-26-86. The lines below in the first column are for the Dec. '86 version, and the lines in the second column are for the 11-26-86 version.

| | |
|---------|----------|
| D803,90 | 14429,90 |
| D829,90 | 1444F,90 |
| D82E,90 | 14454,90 |
| D84E,90 | 14474,90 |
| D87A,90 | 144A0,90 |
| D87F,90 | 144A5,90 |
| D9BA,90 | 145E0,90 |
| DA65,90 | 1468B,90 |

Add these lines to the appropriate patch in PATCHER.DAT immediately below the CADD.EXE line, before the other patches. If you have a release besides the two mentioned above, and would like to patch out the illegal ports yourself, use DEBUG to search for all instances of ADD DX,8 (hex codes 83,C2,08), and if the code is followed one or two lines later with an OUT DX,AL instruction, the OUT instruction should be patched to NOP (90). Similarly, search for ADD DX,9 (83,C2,09) and ADD DX,0A (83,C2,0A), and patch OUT instructions that follow to NOP. In a couple of places following the ADD DX,9 instructions, you will find an INC DX instruction a few lines down, with another OUT instruction following it. Patch this OUT to NOP as well.

Javelin Version 1.1 Patch

The patch for Javelin version 1.1 that was presented in the February Update was for a release dated 11-12-86 (this is the date on the file JAV.EXE on the Startup disk, as revealed by the DIRectory command). Since then, I have developed a patch for another release, dated 7-14-86. I also discovered an important omission in the February Update. Before you use PATCHER to patch the JAV0.OVL file on the startup disk, you must set the date and time (with the DOS DATE and TIME commands) to match the date and time on the JAV0.OVL file (to the minute). That is because the file must have the same date and time on it after the patch that it had before. If it does not, Javelin will not run. If you have already done the February patch (and you have the 11-12-86 release), and did not set the date and time before running PATCHER, just do the patch again. If you have the 7-14-86 release and did the February patch, start over with an unpatched copy. Never put two different patches on the same program.

To patch the 7-14-86 release for use with ZPC, copy the following lines to your PATCHER.DAT file.

```
JAVELIN V. 1.1 (7-14-86) Startup
Insert the Startup disk.
JAV0.OVL
B488,90,90,90,90,90,90
B49A,90,90,90,90,90,90
B4D7,90,90,90,90,90,90
B4E9,90,90,90,90,90,90
B53C,90,90,90,90,90,90
B55A,90,90,90,90,90,90
B5C1,90,90,90,90,90,90
B5DF,90,90,90,90,90,90
B61A,90,90,90,90,90,90
B636,90,90,90,90,90,90
B6C8,90,90,90,90,90,90
B6DA,90,90,90,90,90,90
B70E,90,90,90,90,90,90
B722,90,90,90,90,90,90
B765,90,90,90,90,90,90
1E226,90,90,90,90,90,90,90,90,90,90,90
1E237,90
1E23D,90
1E24B,90,90,90,90,90,90,90,90,90,90,90
z
```

You will also need the patch for the Program disk from the February article, so add the PATCHER lines from that article, starting with "Javelin version 1.1 Program disk". Then follow the instructions in the February Update for making the patches.

Using The Turbo Editor Toolbox

You can use the Turbo Editor Toolbox (an accessory for Turbo Pascal) under ZPC if you modify the files SCREEN.ED and SCREEN.MS. In both of those files, remove the procedures MoveToScreen and MoveFromScreen and replace them with the following:

```

procedure MoveToScreen(Var Source, Dest; Length: Integer);
Begin
  Move(Source, Dest, Length);
End;

procedure MoveFromScreen(Var Source, Dest; Length: Integer);
Begin
  Move(Source, Dest, Length);
End;

```

Then recompile the programs FIRST-ED.PAS and MS.PAS. If you have a Heath/Zenith PC-compatible computer in addition to your H/Z-100, you will find that the altered programs will still work on them. The original versions of the changed procedures contained code to prevent screen flicker, and since Heath/Zenith machines do not flicker, that code is not needed.

Running Framework II In Color

The reason why Framework II will not run in color under ZPC is because it sets video mode 2, which is defined as the 80 column black-and-white text mode. On a real PC, you can still get colors in this mode, but under ZPC, it is a monochrome mode. I have written a small program called M2TO3.COM that maps any attempt to set video mode 2 into mode 3 (80 column color text), and that program is included on the ZPC Upgrade disk (HUG p/n 885-3042-37). For those of you who do not have the upgrade disk and do not mind doing a little typing, you can type in and run the following BASIC program, and it will generate M2TO3.COM.

```

10 REM THIS PROGRAM CREATES M2TO3.COM
20 DEFINT A-I:OPEN "0",1,"M2TO3.COM
30 S=0:S1 = 8638 :FOR I=1 TO 103
40 READ B:S=S+B:PRINT #1,CHR$(B);
50 NEXT I:IF S<>S1 THEN PRINT "TYPING ERROR!":END
60 CLOSE #1:LOCATE 23,1:PRINT "DONE!":SYSTEM
70 DATA 235,41,144,0,0,0,0,88,88,11
80 DATA 192,116,27,61,2,0,116,5,46,255
90 DATA 46,3,1,83,180,15,156,46,255,30
100 DATA 3,1,60,7,91,184,2,0,116,234
110 DATA 64,235,231,51,192,142,216,190,64,0
120 DATA 191,3,1,86,252,165,165,94,199,4
130 DATA 9,1,140,76,2,14,31,186,79,1
140 DATA 180,9,205,33,186,43,1,205,39,13
150 DATA 10,77,50,84,79,51,32,105,115,32
160 DATA 105,110,115,116,97,108,108,101,100,46
170 DATA 13,10,36

```

To use M2TO3, just run it after you run ZPC. It should be run only once each time you boot up MS-DOS, and once you have loaded it, you will not be able to use ANSISYS or SETZPC. You can set up things with SETZPC before you run M2TO3, if you wish.

Source Code

Here are the source code listings for M2TO3 and two programs that were missing from the February Update.

```

PAGE ,132
MAP VIDEO MODE 2 TO VIDEO MODE 3
;
;
; THIS PROGRAM CAUSES VIDEO MODE 3 TO BE SET
; WHEN AN APPLICATION REQUESTS VIDEO MODE 2,
; SO THAT A COLOR DISPLAY IS MAINTAINED. ALSO
; MAPS MODE 0 TO 1.
;
; BY P. SWAYNE, HUG SOFTWARE ENGINEER 03-DEC-86

```

```

CODE SEGMENT
ASSUME CS:CODE,DS:CODE,ES:CODE,SS:CODE
ORG 100H
START: JMP SETUP

```

```

I10ADR DW 0,0 ;INT 10H ADDRESS
DB 'XX'
INT10: OR AX,AX ;REQUESTING MODE 0?
JZ FIXMOD1 ;IF S0, FIX IT
CMP AX,2 ;REQUESTING MODE 2?
JZ FIXMODE ;IF S0, FIX IT
INT10X: JMP CS:DWORD PTR I10ADR ;ELSE, EXIT
FIXMODE: PUSH BX
MOV AH,15
PUSHF
CALL CS:DWORD PTR I10ADR ;GET CURRENT MODE
CMP AL,7 ;MODE 7?
POP BX
MOV AX,2
JZ INT10X ;IF S0, KEEP REQUEST TO 2
FIXMOD1: INC AX ;ELSE, CHANGE TO COLOR MODE
JMP INT10X
ENDRES:
SETUP: XOR AX,AX
MOV DS,AX ;POINT TO INT. SEGMENT
MOV SI,10H*4 ;POINT TO INT 10 VECTOR
MOV DI,OFFSET I10ADR;PUT IT HERE
PUSH SI
CLD
MOVSW ;SAVE INT10 VECTOR
MOVSW
POP SI
MOV WORD PTR [SI],OFFSET INT10 ;INSERT NEW VECTOR
MOV 2[SI],CS
PUSH CS
POP DS ;FIX DS
MOV DX,OFFSET ITSIN
MOV AH,9
INT 21H ;SAY "M2TO3 INSTALLED"
MOV DX,OFFSET ENDRES
INT 27H ;EXIT, M2TO3 RESIDENT
ITSIN DB 13,10,'M2TO3 is installed.',13,10,'$'
CODE ENDS
END START
PAGE ,132
; VIDCON -- A PROGRAM TO DISABLE/ENABLE THE
; VIDEO CONTROL PORT ON A ZHS CARD.
;
; TO USE THIS PROGRAM, ENTER
;
; VIDCON OFF TO TURN THE VIDEO CONTROL PORT OFF
; VIDCON TO TURN THE VIDEO CONTROL PORT ON
;
; BY P. SWAYNE, HUG SOFTWARE ENGINEER
SLAVE EQU 0F0H ;SLAVE PIC PORT
VCIM EQU 01000000B ;VIDEO CONTROL INT. MASK
CODE SEGMENT
ASSUME CS:CODE,DS:CODE,ES:CODE,SS:CODE
ORG 5CH
FCB LABEL BYTE ;DEFINE FCB
ORG 100H
START: MOV DX,OFFSET OPMSG
MOV AH,9
INT 21H ;PRINT OPENING MSG.
CMP FCB+1,'0' ;TURN VIDCON OFF?
MOV DX,OFFSET OFFMSG ;ASSUME OFF
IN AL,SLAVE+1 ;GET SLAVE MASK
JZ OFF
MOV DX,OFFSET ONMSG ;GET ON MSG.
AND AL,0FFH-VCIM ;TURN ON VIDEO CONTROL PORT
JMP SHORT FINISH ;FINISH UP
OFF: OR AL,VCIM ;MASK OFF VIDEO CON. PORT
FINISH: OUT SLAVE+1,AL ;UPDATE MASK
MOV AH,9

```



```

INT      21H          ;PRINT OFF/ON MSG.
INT      20H          ;AND EXIT

OPMSG    DB      13,10,'Video control port is $'
OFFMSG   DB      'off.',13,10,'$'
ONMSG    DB      'on.',13,10,'$'

CODE     ENDS
END      START

PAGE     ,132
FIXWI   -- FIX SCREEN WIDTH IN ZPC

;
;
; THIS PROGRAM FIXES THE SCREEN WIDTH AS STORED
; IN MEMORY BY ZPC, SO THAT PROGRAMS THAT ACCESS
; THE WIDTH AS A WORD WILL WORK. CAUTION: DO NOT
; USE A PSC PROGRAM AFTER RUNNING THIS PROGRAM.
;
; BY P. SWAYNE, HUG SOFTWARE ENGINEER 17-DEC-86

BIOSSEG  EQU      40H          ;BIOS RAM SEGMENT

CODE     SEGMENT
ASSUME   CS:CODE,DS:CODE,ES:CODE,SS:CODE
ORG      0

TESTB   LABEL    BYTE          ;BIOS MEMORY TEST BYTE
ORG      4BH

WIDTH2  LABEL    BYTE          ;WIDTH SECOND BYTE
ORG      100H

START:   MOV      AX,BIOSSEG
MOV      DS,AX          ;POINT TO BIOS RAM
CMP      TESTB,0E9H     ;TEST FOR Z-100 MODE
JNZ      PCMODE        ;IT'S PC MODE
MOV      DX,OFFSET ZMODE ;ELSE, INDICATE ERROR
JMP      PMSG

PCMODE:  MOV      WIDTH2,0     ;ZERO WIDTH SECOND BYTE
MOV      DX,OFFSET FIXMSG

PMSG:   PUSH     CS
POP      DS          ;FIX DS
MOV      AH,9
INT      21H          ;PRINT EXIT MSG.
INT      20H          ;AND EXIT

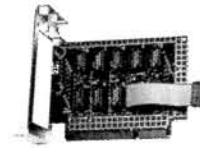
ZMODE   DB      13,10,'Switch to PC mode, then run
                FIXWI.',13,10,'$'
FIXMSG  DB      13,10,'BIOS screen width is fixed.',13,10,'$'

CODE     ENDS
END      START

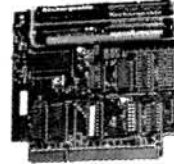
```

HEATH/ZENITH 88, 89, 90 PERIPHERALS

16K RAM EXPANSION CARD



Only \$65.00
Shipping &
Handling \$5.00



REAL TIME CLOCK

Price \$130.00
with
Batteries
Shipping &
Handling \$5.00
\$114.00
w/o Batteries

2 PORT SERIAL/3 PORT PARALLEL I/O CARD

Price \$199.00
2nd Oper. System Driver \$25.00
Ship. & Hdlg \$10



PRICES ARE LESS SHIPPING & TAX IF RES. OF CALIFORNIA.

MAIL ORDER: 12011 ACLARE ST. CERRITOS, CA 90701 (213) 924-6741

TECHNICAL INFO / HELP: 8575 KNOTT AVENUE, SUITE D BUENA PARK, CA 90620 (714) 952-3930

TERMS & SPECIFICATIONS SUBJECT TO CHANGE WITHOUT NOTICE VISA & MASTER CARD GLADLY ACCEPTED

NEW WHISK

FOR IBM AND Z-100

A file management utility that loads much faster than XTREE and uses color far more effectively than VFILER. Capable of displaying up to 54 files at one time, it leaves programs like SWEEP and WASH in the dust!

- Variable file sorting methods
- Copy, Move, Rename, or Delete files
- Display and set file attributes
- View or Print ASCII and Binary files
- Pagination of large directories
- Archive files
- Easy movement through directories
- Runs programs from within WHISK
- Includes both color and monochrome versions
- Available for Z-100 or IBM compatibles (PC, XT, AT)

Only \$29⁹⁵

KALLTRONICS

31316 Via Colinas, Suite #116, Westlake Village, CA 91362 **818-707-2921**
We Accept Money Order, COD, Check
Calif. Res. add 6.5%Tax Please add \$3.00 with order for Shipping and Handling

Z151-52 OWNERS

VIDEO ADAPTER MODULE FOR 309/309A VIDEO BOARDS

Many color video monitors (CGA) will not work with your Zenith PC because they require a shorter vertical sync pulse. The SYNC-MOD plugs directly into your video board 'piggyback' style and allows you to use AMDEK™ and PGS™ monitors that were previously off limits.

If the image on your CGA compatible monitor tears or is smeared across the screen in a riot of disorganized dots and lines, then the SYNC-MOD may solve your problem.

PRICE: \$22 visa and MC accepted

FLR

DIGITAL

(406) 586-0740

815 N. 7th Suite 145, Bozeman, MT 59715

*** 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. \$277.00. Dual Drive Unit\$424.00

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 alone or \$148.50 with 27 256K RAM chips included.

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\$44.00

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 \$699.000, Payload\$477.00

UCI EASY 87. Add an 8087 Numeric Coprocessor. \$69.00 for the board without an 8087 Chip. With 5 MEG 8087 \$197.00 or with 8 MEG 8087 installed\$234.00

UCI MEMORY UPGRADE CARD. We recommend this one highly. The board has sockets for up to 2 MEG of RAM. With no RAM installed \$328.00. With RAM installed and fully tested, 512K \$387.00, One MEG \$446.00, Two MEG \$564.00 Add \$35.00 for EasyDrive RAM Drive Software if desired.

UCI RAMSAVER. Maintains power on UCI MEMORY CARD RAM when computer is off. Save your programs in RAM with your computer off. Payload\$177.00

UCI EASY-I/O. S-100 board that provides IBM PC communications port compatibility with your EasyPC. Easy I/O-1, One Serial Port \$91.00. Easy I/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 \$632.00, 31 MEG \$727.00 System without Drive and Controller\$277.00

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\$48.00

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

Zenith packages with software, manuals and registration cards for the original Z-100 computer series (not for the IBM compatibles).

| PART NUMBER | DESCRIPTION | LIST PRICE | SALE PRICE |
|-------------|--------------------------------|------------|------------|
| MS-463-1 | Z-Basic (16 bit) interpreter | \$175.00 | \$24.00 |
| MS-463-7 | Multiplan | \$195.00 | \$24.00 |
| CB-463-11 | Z-Chart | \$150.00 | \$15.00 |
| CD-463-2 | Condor File Manager | \$299.00 | \$24.00 |
| PK-100-4 | All 4 listed above | \$819.00 | \$62.00 |
| MS-253-1 | Microsoft BASIC-80 (8-bit) | \$175.00 | \$24.00 |
| OS-53-2 | CP/M-85 (8 bit) | \$150.00 | \$24.00 |
| OS-63-4 | Z-DOS | \$150.00 | \$35.00 |
| CB-463-9 | PECON Peachtree to Condor | \$99.00 | \$15.00 |
| RS-463-5 | Peachtree Inventory Management | \$499.00 | \$38.00 |
| RS-463-75 | PeachText 5000 | \$395.00 | \$77.00 |
| WI-463-1 | Remote Batch Terminal Emulator | \$899.00 | \$28.00 |

*** CHIP SPECIALS ***

The finest RAM available and at PAYLOAD prices. Order one to one thousand chips and add only \$2.00 for shipping.

64K Dynamic RAM, 150 ns\$1.35 each
256K Dynamic RAM, 150 ns\$2.94 each
256K Dynamic RAM, 120 ns\$3.40 each

V-20 CHIPS. High Speed NEC V-20-8 8088 replacement. These run at up to 8 MEG and are said to increase CPU speed 10-30%. Payload\$14.75

8087 MATH COPROCESSOR CHIPS. Speeds and improves numeric processing. 5 MEG 8087-3.....\$129.00, 8 MEG 8087-2\$165.00

*** UPGRADE ACCESSORIES FOR Z-150/160 ***
SERIES COMPUTERS

SmartWatch from FBE Research. Installs in ROM Socket on CPU Board in Zenith computer series Z-100/150/158/160. This tiny jewel of a product contains a ten year battery and keeps your computer informed of both time and date at each boot-up. Complete instructions and software included\$44.00

MEMORY KIT #150-256-18. Includes a ZPAL chip which allows use of 256K RAM chips included (18 pieces 256K 150 ns RAM chips). Kit increases 128k memory to 640K or 256K memory to 704K. All chips plug into your existing Zenith Memory Board. Unbelievable but true\$87.00

Winchester Hard Disk Drive Internal Set-up. Includes Winchester drive, controller/interface card, cables and all hardware. With 20 MEG (formatted) drive \$448.00. May be installed in Z-148 using an Expansion Card sold below.

PTZ-148 Expansion Card for Z-148. Includes 2 expansion slots plus a clock/calendar. \$129.00

EVERCOM INTERNAL MODEM. Fully Hayes compatible 1200/300 baud with powerful BitCom software included\$128.00

*** ZENITH SOFTWARE FOR THE ***
Z-150/160 SERIES COMPUTERS

| PART NUMBER | DESCRIPTION | LIST PRICE | SALE PRICE |
|-------------|--------------------------|------------|------------|
| RS-463-75 | PeachText 5000 | \$395.00 | \$77.00 |
| BP-5063-7 | BPI Series Self-Training | \$69.00 | \$25.00 |
| BP-5063-8 | BPI Personal Accounting | \$195.00 | \$55.00 |

*** UPGRADE ACCESSORIES FOR H/Z-89 ***
COMPUTERS

Magnolia Microsystems Double Density Controller. Soft-sectored disk controller. Supports up to four each 5.25" and 8" disk drives. Complete with cables, installation instructions and CP/M ...\$294.00

INTERNAL DUAL DRIVE SETUPS. Includes two half height double sided disk drives and all hardware and connectors required to mount inside your H-89. Steel mounting shield/case included. MITSUBISHI MF501 Setup, 48 TPI, 6 MS seek, 320K\$279.00
MITSUBISHI M4853 Setup, 96 TPI, 3 MS seek, 640K disks ..\$284.00

*** HALF HEIGHT FLOPPY ***
DISK DRIVES

MITSUBISHI M501 5.25" 48 TPI DS/DD 320K/360K\$105.00
MITSUBISHI M504 5.25" 96 TPI DS/DD 360K/1.2 MEG ..\$177.00
MITSUBISHI M4853 5.25" 96 TPI DS/DD 640K\$109.00

*** LAPINE HARD DISK DRIVES ***
2 YEAR WARRANTY CALL FOR PRICES

*** SEAGATE HARD DISK DRIVES ***

ST-225 20 MEG Winchester Hard Disk\$355.00
With Western Digital Controller & Cables\$448.00
ST-238 30 MEG, Requires RLL type controller\$405.00
With Adaptec RLL Controller & Cables\$499.00
ST-4026 20 MEG High Speed for Z-200\$508.00
ST-4038 30 MEG High Speed for Z-200\$619.00
ST-4051 40 MEG High Speed for Z-200\$739.00
ST-4096 80 MEG High Speed with Software\$1425.00

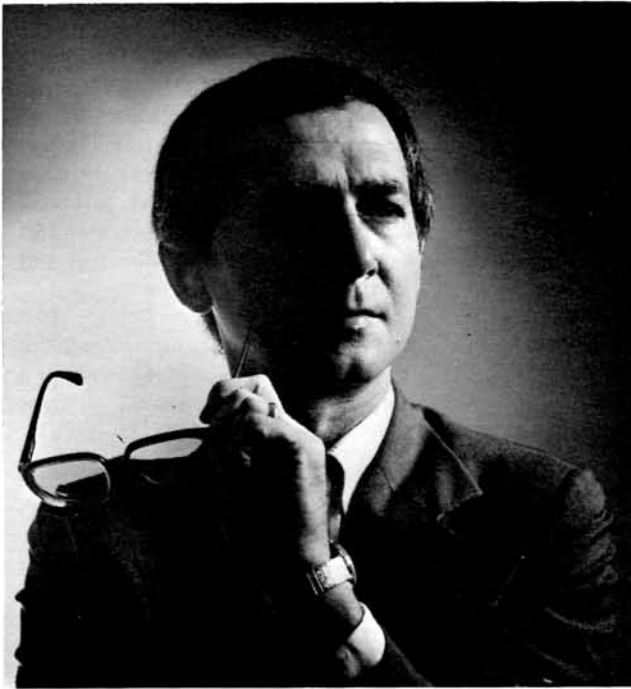


PAYLOAD
COMPUTER SERVICES



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

Please MAIL or PHONE your order today and expect prompt service. MASTERCARD and VISA gladly accepted with no additional charge. All hardware carries a 90 day warranty. Add \$5.00 to all prepaid orders for handling and shipping, we pay the balance. Texas Residents please add 6.125% sales tax. We accept purchase orders from schools, government and approved accounts.



Mainstream Computing

Joseph Katz

103 South Edisto Avenue
Columbia, SC 29205

Copyright (C) 1987, by Joseph Katz. All Rights Reserved.

What a month it has been. You'll recall that our Apple LaserWriter came a few days before the end of 1986. Since then I've spent nearly all my disposable time exploring it. There's still more to do before I'll feel entirely comfortable with the LaserWriter, because I'm doing pioneering work. Oh, I'm sure there must be numerous others who have found the promised land before me. I'm not egotistical enough to think I'm really the very first. But so far as I can determine, no one else has yet published a real and solid map of how to use an Apple LaserWriter — or any PostScript printer — on an IBM compatible computer. The most I've been able to find are a few things that say "Take care. There be monsters here." Since I have little superstition and less caution, I've prodded the monsters to test their sense of humor. They have none. They bite, sometimes pretty hard. I'll keep you posted on PostScript, the LaserWriter and maybe some other sophisticated printers, software for them, and related things from time to time.

Right now, I'm having a ball producing the best printed output I have ever seen come directly from my very own microcomputers. Don't get scared off by my allusions to difficulties. I'm long past the hurdles a normal user would face in using the LaserWriter on an IBM compatible and I know how to instruct you on making the combination sing. Oddly enough, you're better off using a Heath or Zenith computer with an Apple LaserWriter than you would be using any other IBM-compatible or an IBM machine itself. The key is Zenith's MS-DOS. So we're not talking about a combination too esoteric to be practical. It's most certainly a practical combination provided you're interested in near-typeset-quality printing or knockyoursocksoff black and white graphics. The key is PostScript. If all you want from a laser printer is reasonably-fast, reasonably-quiet, high-quality typewriting, you'd be silly to spend the money it takes for this kind of printer. You can get a good laser printer of the other kind for much less money. I want this kind. You'll be seeing more and more of them during the next few months. There's a good chance that PostScript — the page description language in

the Apple LaserWriter and several other printers now — will become the standard page description language. It is at least a standard at this time.

Your reward this month for staying with me on my adventures is more RAM in your computer and simplified access to your hard disk. The extra RAM is yours free, by the way. You own it. You paid for it. You might as well get it. The simplified access to your hard disk is free too, thanks to some cheap tricks I developed a few years ago. Sure, I'll do the free RAM first. I understand your impatience.

Free RAM In Your IBM Compatible

For the past few years I've been wondering what has been stealing RAM from my computers. For example, I started out with what was supposed to be 540KB of RAM (the standard amount) in my '241. I counted the RAM chips and they were where they were supposed to be. But I couldn't always run programs supposed to run on machines with 540KB of RAM. I had to install Boca Research's *TophAT* to get 640KB of RAM just so I could run programs supposed to run in 540KB of RAM. That's crazy.

The thing is that every time I booted the computer, I wound up with a whole lot less usable RAM than it should have had. An indication of what I mean is that CHKDSK would report my 655,360 bytes (640KB) of base RAM gave me only 452,304 (442KB) of usable RAM. I myself had spent part of the 198KB difference on system overhead: on an expanded number of files and buffers for performance, on device drivers for added boards, on my data pathfinder, and on an initial Mark so that I could clear out any TSR programs I use from time to time. (You'll remember my talking about Turbopower Software's *Mark* and *Release* programs in the February 1987, "Mainstream Computing.") I figured all that stuff together to consume 60KB-61KB. Where in the world was the other 135KB or so of memory going? Don't be silly: of course I looked under my disk to see if there was a memory pool on the car-

pet. No leak there. Don't write off this problem if you own another IBM compatible either: it hits all of them.

I knew if I waited long enough, the grapevine would solve my problem for me. The indolent, after all, inherit what's left by the enterprising. Sure enough, the vine produced some tender grapes. It turns out, as any fool can plainly see (the quotation is from Al Capp's *Lil Abner*), the TSRs were stealing my RAM. They're stealing yours too.

Hold on. You'll be embarrassed if you say you don't use any TSR programs at all. Of course you do: COMMAND.COM, the MS-DOS command interpreter is one. If you set a PATH to areas of your hard disk, there's a TSR command you're using. If you use a supplementary data pathfinder, there's another one. You're beginning to see what I mean now, I think: if you boot your computer with an AUTOEXEC.BAT file that does just about anything except clear the screen, chances are you're working with some kind of TSR. If so, chances are you're getting at least some RAM stolen.

The reason is that every time you run a TSR of any kind, DOS accompanies it with a copy of the MS-DOS environment as it existed at the time you ran the TSR. (If you're not interested in explanations, you can forget that. But don't skip. We're doing business now.) So my RAM — more than 135KB of it — was going to needless duplication of the same information time after time after time, with each TSR loaded.

Here's how to reclaim that wasted RAM by revising your very own AUTOEXEC.BAT file.

First, make sure to put the TSRs you load automatically, each time you boot the computer, into the root directory of Drive C (assuming you have a hard disk) so you don't have to supply a pathname. DOS keeps track of that pathname throughout your computing session, even after it's no longer useful. Don't give it a pathname to track.

Second, postpone setting the PATH (and data path, if you use a data pathfinder) in your AUTOEXEC.BAT file until the last possible line in that file. If you can possibly work things out so you can set it in the next-to-last line (I'll explain in the next paragraph why it's "next-to-last line" and not the "last line") do so: it's worth spending time and ingenuity on this job.

Third, lie to MS-DOS. Start your AUTOEXEC.BAT file with a line that says you have no command interpreter at all: MS-DOS is dumb enough to believe it. The line — **SET COMSPEC=** — should be typed just that way, with nothing (not even a space) after the equals sign. Then continue with the rest of the junk in your AUTOEXEC.BAT. Set things right in its very last line with **SET COMSPEC=C:\COMMAND.COM** (assuming you have a normally-organized hard disk system). You'll recall I said *all* TSR programs get stored with a copy of the environment. The environment includes COMMAND.COM. And since COMMAND.COM is itself a TSR, it is stored with a copy of the environment when it is run. Postpone its execution and you'll save bytes.

Right after I did all this, CHKDSK reported I had 593,280 bytes of RAM available for my programs. Where else can you get so much RAM for free and without even the bother of installing it?

Cheap Tricks With Subdirectories

All right, they're lowdown tricks. Maybe they're even sort of lazy or sleazy tricks. But whatever you call them, they're awfully useful tricks if you have a hard disk with several subdirectories and you find that doing things by the book takes too long for you. I call

them "cheap tricks" because they don't cost anything but would be worth money even if they did. Here's what I mean.

Trick 1. What do you do if you want a list of files — a **DIR** — in a specific subdirectory? What you're told to do is make it your default directory and call for a directory listing. So, assuming you're in the root directory and want to know what's in its WP subdirectory, you type **CD \WP [RETURN]** and then type **DIR [RETURN]**. (From now on we'll assume you press the RETURN key after all commands.) You then get a listing of the WP subdirectory. The faster way to do the same thing is stay in the root directory and type **DIR WP**. (Don't argue and explain why it won't work. Just type what I tell you to type and look at what scrolls down the screen.) So to get a subdirectory listing, you can stay in the root directory and type **DIR [SUBDIRECTORY NAME]**.

Trick 2. What if you're in a subdirectory and want a list of files in the root directory? You're told to back up to the root and call for a directory listing. I just type **DIR ** and get what I need with less sweat. (The \ is MS-DOS's pseudonym for the root directory.)

Trick 3. What if you're in a secondary or tertiary subdirectory (the subdirectory of a subdirectory, or the subdirectory of a subdirectory of a subdirectory) and want to know what's in the "parent," the subdirectory immediately above? I won't bother with what you're told to do, because this stuff is tedious to explain and you get the point anyway. What I do is type **DIR ..** (because the .. is MS-DOS's pseudonym for the parent of the current directory).

Trick 4. What if you want to remove a subdirectory? You're told to first change to that directory, clear out its contents (because you can't remove a subdirectory so long as it contains anything at all), then move to its parent directory (because you can't remove the current subdirectory), and at last issue the command to remove the subdirectory. People who do things that way tend to have white hair and twitches by age 20, I think. I stay youthful by conserving my fingers. While I'm in the parent directory, I type **DEL [SUBDIRECTORY NAME]**, answer the "Are you sure?" question in the affirmative, then type **RD [SUBDIRECTORY NAME]**. Right: the **DEL** used that way deletes all files in the subdirectory.

What all these tricks have in common is that they take advantage of the properties inherent in MS-DOS' treatment of a subdirectory as a list of filenames maintained in what is itself really a file. A subdirectory is a special kind of file, identified as such by a special attribute byte, but it's a file nonetheless. Within certain severe limitations, therefore, you can do the same kinds of things to a subdirectory you are permitted to do with other files.

Feel free to experiment with this basic concept. I haven't seen any of these tricks mentioned in the DOS books or articles I've encountered, so I guess they're an original gift from me to you. You'll be able to work out on your own several variations that I haven't mentioned because I don't want to run this subject into the ground. Have fun. Don't get into trouble. Experiment with subdirectories on floppy diskettes (make sure they're backups, please) before you try fiddling with those on your hard disk.

COM Ports And Communications Programs

As I've said, the communications programs I use now are *Hyper-ACCESS*, *Crosstalk XVI*, *Mirror*, and *HUGMCP*. I understand that there are several really good shareware programs with loyal followings, but what I use now does what I need now and so I plan to continue using it until I have other needs or find things that meet my existing needs better. I spend no less than an hour a day, every day of the week, working telecommunications. Sometimes I'm on

the air eight hours at a stretch, sometimes writing with one computer system while another maintains a connection to a remote system. I dug through other stuff to find what I use now, and I'm happy with it. I don't have the time to work with stuff I don't need, no matter how nifty it is. I have to practice my professions for a living. Matthew likes to eat. Skipper, Janet's mild mannered poodle, makes threatening noises when he's hungry. I get scared of fast, hungry little dogs with sharp teeth.

I said a while back I didn't know about whether programs other than the first three I use support anything but COM1 or COM2 on IBM compatibles. Jim Buszkiewicz has just told me that he has just finished a version of *HUGMCP* that allows you to specify your own ports in addition to COM1 and COM2. Now you can get into as much trouble as you like by specifying combinations of port addresses and interrupts. Jim says there are other nifty new features too, all of them generally useful. Owners of the current *HUGMCP* can update to the new version for \$5 the usual way. Call or write Nancy Strunk at HUG to arrange things. Don't call or write me, please, because I'm using up paper at such an incredible rate watching grand things come out of my LaserWriter that I might not be able to resist the temptation to swap your Mr. Lincoln for a ream of paper. In any case, I'll be happy to have that version of *HUGMCP* because I like the streamlined approach Jim takes: when I'm in a situation in which I don't need the features of the first three programs, *HUGMCP* is just right for me.

Don M. Deck of Line Pine, California, is a nice guy I've never met and had not heard from before. In addition to being a nice guy, he's thoughtful too. He didn't merely tell me that *PRCOMM 2.4.2* and *QMODEM 2.3* both support additional ports: he sent me copies of the two programs. Such courtesy deserves immediate attention, so I stayed up later than usual that night of the day when the disks came. Those programs do indeed support COM3 and COM4, and nicely too. Add them to the list.

Que Books

I want to meet the art director at Que Corporation. Que publishes first-rate computer books and sends copies to me, which I think is a very nice thing to do. Some time ago, I had a telephone chat with Chris DeVoney of Que (he's an editorial big wig at Que, a sharp C programmer, the author of *Using PC DOS*, which I think the best introduction of its kind I've seen, and an attractive talker), and I'd like to meet him too. I'm a publishing junkie.

But the person I really want to meet is the art director, or whoever it is designs the covers for the Que books I've seen. I had the recent ones spread out on a table ready for placement on my "to examine" shelf and burst out laughing. Whoever it is has a splendid sense of visual humor. I didn't catch on at first because it's all so unexpected in connection with computer books.

Run right down to your favorite, full-featured bookstore and look at the photographs on the front of these titles, even if you don't use the software they explain.

Turbo Pascal Program Library by Rugg and Feldman shows an IBM monitor in front of an old fashioned wooden index-card filebox. On the screen is a *Turbo Pascal* procedure. Next to it is a stack of index cards flipping at breakneck speed into the filebox. Consider the connection between the procedure on the screen and the only readable index card in flight.

Using Q&A by Ewing and Langenes has one bust in profile facing another shown full face. Between and behind them is a computer screen on which you can read "Shall I do the following?" That

boldfaced sentence seems to come from the profile's lips to the fullface's ear.

Using Smart by Schwartz has the software packages encased in their plastic boxes, a graduate's mortarboard on top and a rolled diploma bound in red tape at the side. Not bad.

1-2-3 Tips, Tricks, and Traps shows the Lotus disks — instead of un-copyprotected rabbits? — emerging from a magician's hat. To the left are white gloves and wand. To the right is a . . . I won't insult your intelligence by naming it. Go look.

My favorite is the cover of *1-2-3 Command Language* by Fenn. It shows part of a commanding officer's regalia arranged with wonderful casualness.

I don't know the software so I can't get all the humor on *Excel Macro Library* by Campbell and I don't get at all *Using Javelin* by Ewing and Carrabis. *Excel Macro Library* shows a Macintosh with Hyperdrive and mouse, resting. At the right is an hour glass and *The Dictionary of American History*. In the foreground is a stop watch.

Enough. You really must take a look at these books. I want to see more. I also want to meet the art director.

I'll let you know what's in some of the books later, when I have time. A Que book on anything usually repays its cost quickly with good, solid information of a "how to" kind. I own quite a few already. It hadn't hit me before that they're worth collecting for their covers. I can't wait to see the next group.

Maybe my real favorite is *Excel Macro Library*. It's funny even though I don't know the software.

I'll see you again next month. Right now I have a PostScript printer waiting patiently by my side until I can resume this session of our fun and games. I won't promise, but I certainly will try to have a few specimens of its incredible output for you to see then.

Products Discussed

Using Q&A. (ISBN 0-88022-262-X.) \$19.95
By David P. Ewing and Bill Langenes.

Turbo Pascal Program Library. (ISBN 0-88022-244-1.) \$19.95.
By Tom Rugg and Phil Feldman.

Using Smart. (ISBN 0-88022-229-8.) \$22.95.
By Andrew N. Schwartz.

1-2-3 Tips, Tricks, and Traps. 2nd Edition.
(ISBN 0-88022-263-8.) \$19.95.
By Dick Andersen and Douglas Ford Cobb.
Revised by Dick Andersen, Bill Weil, and Janet McBeen.

1-2-3 Command Language. (ISBN 0-88022-268-9.) \$19.95.
By Darien Fenn.

Excel Macro Library. (ISBN 0-88022-255-5.) \$19.95.
By Mary V. Campbell.

Using Javelin. (ISBN 0-88022-199-2.) \$19.95.
By David Paul Ewing and Joseph-David Carrabis.

Que Corporation
7999 Knue Road
Indianapolis, IN 46250
(317) 842-7162



A Winchester For The '89

Part Eleven

Peter Ruber
P.O. Box 502
Oakdale, NY 11769

Time Out

In place of the usual product dissection this month, I thought I would take time out to reflect and make some observations that have been prompted by the bulk of the correspondence that I have received over the course of this series.

Eighteen months have elapsed since I first contemplated writing this series, and fourteen months since the first installment appeared in REMark. This is a long time to devote to one category of computer hardware for a single computer, but worth the time and effort because it helped to fill a void in the published material available for the '89. The series was launched at a fortuitous time, because a number of systems appeared suddenly and provided a lot of additional material that enabled me to cover all Winchester systems and related products one after another.

I also mentioned that my familiarity with hard disk systems was practically nil and that researching and writing the series would be a learning experience for me, as it would for others who were in the same boat.

One of the articles I planned to do in the projected scope of the series, was a do-it-yourself Winchester system using public domain CP/M utilities and some close out bargains from various electronic wholesalers. While this appeared, at first, to be a sound project, I discovered very quickly that close out bargains don't remain on the market for any extended period of time. So, considering the delays incurred by working with a "home-brew" system, and testing it, writing the article, and then waiting for it to be scheduled for publication, most of the hardware would have disappeared from the market.

Hard disk controller boards were another problem. Each has its own protocols for accessing a hard disk drive, which require a driver and a hard disk formatter. While these are not difficult to customize (providing you could get the source code to a public

domain driver and are experienced in assembly language programming), access to controller and disk drive technical manual is a problem. Most suppliers I contacted informed me that they don't always get the necessary documentation from the manufacturers.

The final topic I had to consider was system reliability. What if it failed? How could you determine which element was at fault — the drive or the controller — short of having duplicate hardware to substitute? Could you get the component serviced? Would the cost be reasonable?

Unfortunately, the answers weren't always encouraging. Some close-out drives were reconditioned or tested units taken out of service from leased equipment. Others were manufacturers' overstocks. The drives that had a reliable reputation disappeared quickly. The oddballs remained on the catalog pages like relics from the stone age. For the most part, they were all specimens of an older technology that had a higher failure rate than drives in current production. Worse still, the cost of servicing a hard disk drive is costly. It's not an object you dump on your workbench and then take apart. The plated media is highly sensitive to contamination and must be serviced in a near-sterile environment. That costs money — often as much as twice what you paid for your bargain drive. A lot of drives (even though still in manufacturers' original packing) were sold without warranty, or on a limited 30-day exchange basis. The odds of getting all your components together during the same 30-day period, and then getting the system up and running, were rather slim.

You could also have problems even with a brand new hard disk system. Two of the systems I worked with developed problems. One apparently was sensitive to power surges and damage occurred to the drive's electrical system. Another unit worked like a charm one day, and wouldn't boot the next, for no apparent reason. I did not mention these problems because I felt they were

isolated. Furthermore, as the systems had been supplied by reliable vendors, the defective products were replaced immediately.

I therefore reached the conclusion that it was safer to spend a little more and get a guaranteed turn-key system that you could get serviced or replaced if the worst fears come true. This is especially recommended if you are going to store data that has work related importance.

The Mail Bag

A few days before I sat down to write this column, I received the following letter from a HUG member in Pittsburgh:

"I am a fellow HUGGIE and I'm impressed by your ambitious and thoroughgoing series of H-89 articles in REMark. I own an H-89 and love it, although I do still just use floppies. Nevertheless, the H-89 is our common ground in these days when your REMark articles alone represent a substantial portion of total user-generated information currently being published. The magazines and newsletters are swinging over to the PC-compatibles. We don't need PCs, and we are being left behind. Where they make their mistake is in overlooking the fact that we don't want to be left behind . . ."

Flattering letters are always nice to read — but a lot of the correspondence I received had a prevailing mood that I feel deserves some commentary. Most H-89 owners have a fanatical loyalty to their machine, a kind of beneficence some people bestow on their first born. My own fanaticism for the '89 has been on the downslide lately.

This is due in large part to my involvement with the Winchester series. My system ceased to be a functional working tool except for writing the related articles. Equipment moved in and out regularly and my '89 never had the same configuration for more than a few weeks at a time. At one point, I had several different hard disk systems, co-processing units, graphic cards and more RAM installed than I could use in a month of Sundays, that my '89 grew into an obsessive monstrosity. These interfacing projects were fun but not productive in terms of my regular work.

I picked up a Zenith PC along the way, and a Visual PC portable I could lug to and from my office. And it was inevitable that my tastes would alter in the process. I started making comparisons, but this was like comparing a Chevy to a Corvette. The analogy wasn't really fair, because the '89 and the PCs are products of radically different technologies. IC technology has progressed to such a point where it takes half the number of components to produce a computer that is ten times more powerful at half the price.

The only consumer product that can offer this kind of cost effectiveness is a computer. A short four to five years ago a basic 48K '89 went for nearly \$2,000 in kit form. Today, for the same money, you can buy two H-148 PCs. But you can't buy two Corvettes for the price of a four-year old Chevy, even though auto manufacturing techniques have the same kind of high volume, automated assembly-line production as computers now do.

Don't be misled into thinking that I have turned my back on the '89, or consider it less important than my PCs. I'm fond of all my children, even though their personalities and abilities differ. And so it goes with my computers. When the opportunity presented itself recently to pick a low-priced used '89 and an H-19 terminal, I snapped them up, and my office is cozier now that I have six computers cranking up the monthly electric bill.

The only real benefit a PC has over an '89 (aside from a lower cost ratio) is the software. It has become more user friendly, and the functions of many sophisticated programs can be mastered without having to digest reams of unintelligible documentation. Too bad that software manual writers haven't achieved haven't and mastered the art of simplicity. It's easier to learn Microsoft's WORD by loading the disk and playing with the program than by trying to understand the manual.

At the primitive level, all computers function in much the same fashion, and the only difference between a 16-bit PC computer with lots of available RAM and an 8-bit '89, is that there is generally less disk I/O traffic with a PC, because larger portions of a program can remain in memory. An '89, with a hard disk overcomes some of the speed advantage a PC has in this respect. But the PC's cursor movement is pathetic when you compare to the '89. The '89 has a separate CPU for the terminal board, so it zips along more than twice as fast as the cursor on my PC's. This is a benefit when there are numerous editing changes to be made within a manuscript. You can easily become spoiled by the bells and whistles found in PC software; and, by the same token, you can ignore most of them as you'll never have need for them. So, for most general business and personal productivity and programming functions, an 8-bit '89 is not really so backward as some pundits will tell you. The one area in which the PC does shine is in CAE/CAD applications. These types of programs were never fully developed for 8-bit computers because they required massive chunks of free memory.

To get back on the track, the general undercurrent of the letters I've received indicate that there is still a sizable installed base of '89 users who want (and demand) support. But what some people seem to ignore is that the '89 has been around since 1979 and a lot of articles have been written about the computer. Some people may not be aware of the fact that all the early issues of REMark are available from the Heath Users' Group in bound volumes, and that Sextant still has a supply of their old magazines.

They're worth collecting if you plan to keep your '89 as your main computing tool in the years ahead. If it continues to function without problems, and the software you have collected is suitable for your needs, then by all means enjoy your machine. But you must realize that sooner or later people are going to run out of worthwhile things to write about, but an important contribution on a hardware enhancement will not be overlooked by the editors.

I think the fears of some '89 owners who believe that support will dissipate in the next few years is unfounded. Heath/Zenith has always provided a minimum of seven years of service and technical support for any product they discontinue. There is also support at the local HUG level, and through correspondence with others. I honestly feel that the kind of user participation and help available within the H/Z community is far better than what is available for other computer systems. Where can you get support for your Osborne, Eagle, Columbia, OSI, Exidy Sorcerer, or a dozen others?

But there is the other side of the coin when you discuss support. How many '89 owners have given support to the enhancement products offered by Heath and by independent vendors?

I have gotten to know some H/Z support vendors rather well in the last few years, and most of them feel that '89 owners always had a "Look but don't buy" attitude, and that only a relatively small percentage of these people ever enhanced their systems beyond the basic configuration.

I also know for a fact that most vendors of '89 enhancement products made very little profit from their ventures. In some cases it was a hobby for an engineer or a programmer, or a sideline for a dealer of a broad range of computers and peripheral products. No more than a half-dozen products out of the 70 or so that were produced over the years ever returned a half-way decent profit. A powerful expansion chassis suitable also for controlling external devices sold less than two dozen units.

It costs money to advertise, which must come out of the profit of the product. If the vendor of a first-rate product cannot sell enough units to pay for his time and efforts, then obviously he's going to turn his talents toward machines that offer more of a potential for him.

Not everyone is going to need a graphics card, or a RAM card, or what have you. But some exciting products that offer speed and power to rival the PC's have been largely ignored. I have heard the argument that some of these products were very expensive. This is true. Custom products assembled in limited quantities have a higher unit cost.

If, for the sake of argument, a vendor's profit on a \$200 board is \$60, and he has a monthly advertising expense of \$300, then he must sell 5 boards just to meet his advertising bill, not counting his telephone, office expenses, printing and mailing costs, etc. If he only sells 4 units that month, he's lost money, and too many losing months will put him out of business. The same analogy holds true whether you must sell 5 boards or 50 boards every month. The more you sell, the higher your costs go up in proportion.

You can't survive in an atmosphere of diminishing returns. So, support is a two-way street. If you don't support your vendors, they will ultimately abandon you. You don't have to rush out and buy everything in sight, because you'll never use it, let alone afford it. And if you do contemplate a purchase, check it out at a dealer or with someone who has the product. Learn if it will do what it claims it will. It might turn out that it's just what you needed to make your computing tasks a little easier and productive.

A few readers have chided me for not being more opinionated about recommending this product over that one. Well, I didn't feel it was appropriate to be too subjective, because I worked with one system at a time and really had no idea what the next one would be like. So I attempted to approach each system with a fresh mind and from the viewpoint of an average user. I have no exceptional hardware or programming skills that would enable me to nit-pick here and there. And I tried to arrange the format of each article the same way. I listed the features and explored the software and hardware installation. I didn't concern myself too much with the price spreads because a user would have to make up his own mind of which system was suitable to his or her budget. Some systems also had expansion options that others didn't, and those costs would have to be equated by the user in terms of need.

Occasionally some of the documentation irritated me, especially since I expected a little more. Other systems had superb documentation. Where a system might generate power supply problems, I went to some lengths to secure solutions and compiled a list of the power requirements of most of the major '89 enhancement cards so that users had a means for analyzing their own system. In another instance where a system exhibited possible problems by its close proximity to the flyback transformer, I recommended shielding and additional power supply modifications.

These suggestions were by no means arbitrary solutions. They were tested and retested, checked out by some willing victims I

recruited, and then double-checked by the manufacturer. I tried, in a sense, to cover all contingencies where there was doubt in my mind — or where a problem was brought to my attention.

Since hindsight is a great teacher, it is, of course, possible that I overlooked a feature or a minor problem here or there. If I have, no one has brought them to my attention, so I assume that, despite deadlines and other writing obligations, every system got a fair shake, and readers had a reference source by which to make comparisons.

What's Next?

Working with hard disk systems has been very interesting. There is a new technology emerging that is not only lowering costs dramatically, but making drives more reliable. There is also the SCSI interface, and a universal SCSI/PLUS I/O bus being promoted by Ampro Computers that will open up the architecture of closed computer systems.

I spent several days in Silicon Valley toward the end of last August nosing around and talking with people. I also visited the management at Ampro to get some first-hand information on SCSI/PLUS. I will therefore spin off a limited series devoted to the SCSI and SCSI/PLUS interfaces and discuss some of the new hard disk technologies for PCs. There will be more than a few surprises.

There will be two more articles in the present series. The next one will deal with some product updates, and the final piece will discuss a new 16-bit co-processor that can be used with the '89 and Heath/Zenith terminals. This will be a blockbuster to end the Winchester/'89 series, because there is nothing like it on the market.



TaxAide

Income tax preparation worksheets for use with your Lotus 1-2-3.

● PRODUCES IRS APPROVED PRINTOUTS—NO SPECIAL FORMS TO BUY ●

- Easy to Use — Menu Driven — Integrated ●
- Screen Displays Resemble Actual IRS forms ●
- Automatically Calculates Income Tax ●
- Full Technical Support — Satisfaction Guaranteed ●

TaxAide Plus — Provides worksheets for Forms 1040, 1040A, 2106, 2119, 2210, 2441, 3468, 3800, 3903, 4562, and 6251; Schedules 1, A, B, C, D, E, F, G, R, SE, and W; plus IRS worksheets. Multiple Forms 2106 and 4562 and Schedules C and SE. TAXAIDE PLUS PRODUCES IRS APPROVED PRINTOUTS FOR ALL FORMS AND SCHEDULES, INCLUDING 1040 AND 1040A. Ideal for income tax preparers. **\$59.95**

TaxAide Personal — Provides worksheets for Forms 1040, 1040A, 2106, 2441, and Schedules 1, A, B, D, G, R, and W, plus IRS worksheets. Two Forms 2106 are provided. Produces IRS approved printouts (except Forms 1040 and 1040A). **\$29.95**

REQUIRES ONLY 192K of RAM — If you have more memory, you may still run your background program such as SideKick or Perks.

Requires Lotus 1-2-3 Release 1A or later, and DOS 2.0 or later.

To order, send check to:
Kansas residents add
5% sales tax

Software Applications of Wichita
2204 Winstead Circle
Wichita, KS 67226

For VISA or MasterCard orders, call (316) 684-0304.

TaxAide is a trademark of Software Applications of Wichita.
Lotus and 1-2-3 are registered trademarks of Lotus Development Corp.

SideKick is a trademark of Borland International, Inc.
Perks is a trademark of Barry A. Watzman.

H-150 Speed-up Modification Update

Dante Bencivengo

P.O. Box 234

Wyandotte, MI 48192

I was pleased to see my article on speeding up the H-150 published in the June 1986 Issue of REMark. Since that time I have gathered information on the modification from HUG members who have tried it, as well as from my own experimentation. I want to thank all the HUG members who wrote me with feedback both good and bad. Much of the information is presented in this update article.

Benchmarks

Often I am asked about the Norton SI speed index. The design of the speed-up modification presented in the June REMark is similar to many commercial designs currently available. When operating at 7.37 MHz, it is as fast as any of the commercial mods that increase the CPU clock speed (not to be confused with the plug in cards using an 80186 or 80286 which are significantly faster and more expensive). Although I did not report the numbers obtained from the Norton SI utility in the original article, I will report them here purely for comparison.

At this point in time, it is pretty much known that the NEC V-20 fools the SI utility into calculating a meaninglessly large speed index which bears no resemblance to the actual performance increase. Simply replacing the 8088 with an NEC V-20 results in a Norton SI rating of 1.8 (at the standard 4.77 MHz clock speed), while the actual noticeable increase is around 6-10%.

When running at a higher clock speed, the number becomes even more meaningless with values of 2.8 being commonly quoted. With a standard Intel 8088-2, the SI utility returns a speed index of 1.6 at 7.38 MHz and 1.5 at 6.67 MHz. These numbers are a realistic indication of the increase in computer throughput. When an NEC V-20 is substituted, the numbers become 2.8 and 2.5 for 7.37 and 6.67 MHz, respectively. The Norton SI speed index is a useful tool for comparing performance under similar hardware conditions, but is not so useful for making absolute measurements.

For a CPU intensive task (all things being equal), the increase in performance over the standard clock speed can be predicted fairly reliably by dividing the higher clock speed by 4.77. The resulting ratio will be the increase in performance. If there is a significant amount of I/O or disk access occurring, the actual increase in performance will be less than the predicted increase.

Parts Replacement

Most of the feedback concerning the modification indicates that most readers have gotten their 150 going at 6.67 MHz with some at 7.37 MHz. The information contained within this article should provide some answers for those very stubborn cases out there.

The previous article mentioned that the bus controller and DMA controllers need to be of a certain manufacturer. To run at 7.38 MHz in an H-150, the 8288 bus controller must be a ceramic device manufactured by Advanced Micro Devices. Unfortunately, I have found that not all AMD devices are capable of running at 7.37 MHz. This problem is peculiar to the 150, since a bus controller that does not work at 7.37 MHz in a 150 (say one manufactured by Intel) will operate just fine in an IBM PC at 7.37 MHz. There would seem to be a fairly subtle timing problem with the bus controller when running at higher speeds, a problem I have as yet not isolated.

However, just about every AMD D8288 that I have sampled will run at 6.67 MHz. Another 8288 supplied with some 150s, the Harris CP82C88, will usually operate properly at 6.67 MHz, but again not at 7.38 MHz. It is interesting to note that neither the 148 or 158 uses the 8288 bus controller, but rather its functions are performed by PAL devices in both computers.

The bus controller is the most critical factor in getting your computer to run reliably at high speed. Usually the computer will run for a while and then just hang up, usually not allowing anymore keyboard input.

The use of a heat sink on the 8288 is very helpful, often allowing marginal ICs to operate properly while increasing the reliability. The heat sink recommended in the parts list does not fit the ceramic package very well at all. Unfortunately, there is no other clip-on heat sink commercially available. My solution has been to epoxy onto the top of the 8288 a small shim of aluminum. The strip of aluminum, measuring 7/8" x 1/4" x .03" thick helps to prevent the heat sink from moving around on the IC and to keep it properly centered.

The other critical IC is the 8237A-5 DMA controller which should be manufactured by either Intel or NEC. Feedback has indicated that the more readily available NEC D8237AC-5 works just fine. There does not appear to be a need for a heat sink on the DMA controller.

An additional problem that may prevent many older 150 CPU boards from working at high speed has surfaced. Apparently, Heath had upgraded some of the 74LSXX parts to 74ALSXX parts on the newer CPU boards. The ALS (advanced low power Schottky) series ICs have about 1/2 the propagation delay of the comparable LS (low power Schottky) series and actually consume less power to boot. I recommend that anyone with one of these older CPU boards replace the ICs listed in Table 1 with the ALS version. Since Heath found it necessary to do so in the stock 4.77 MHz machine trying to get away with these ICs at the higher speeds is almost impossible. Fortunately, the ICs are commonly available and not at all expensive. This cleared up some mysterious problems I have had in getting some computers to operate even at 6.67 MHz.

Table 1

| | |
|------|---------|
| U204 | 74ALS32 |
| U210 | 74ALS00 |
| U217 | 74ALS74 |
| U219 | 74ALS08 |
| U224 | 74ALS10 |
| U226 | 74ALS74 |
| U234 | 74ALS04 |
| U235 | 74ALS00 |

8 MHz Operation

A good deal of mail and a lot of interest concerning operation at 8 MHz. I have experimented with it and found several problems. On first approximation, the floppy drives in my computer would not operate at all (although this could probably be cured). The major problem is (again) with the bus controller. The AMD D8288s that operate properly at 7.37 MHz will often fail at 8 MHz. From my point of view, this last little bit of speed is not worth the grief. If some of you have been successful in running your machine at 7.37 MHz and wish to try 8 MHz operation, then all that is necessary is using a 24 MHz crystal for Y1 and replacing the 100 ns delay line on the Heath memory board (U456) with the 75 ns part (part# HE-41-21).

Using The 8087 Numeric Coprocessor

The 8087 numeric coprocessor can be used successfully provided it has the proper speed rating. As with microprocessors, the 8087 is available in a variety of speed ratings for the job at hand. The normally available 8087-3 is rated for 5 MHz operation and generally the speed rating is not particularly conservative. In other words, if it's a 5 MHz part that is as fast as it will go. I have had a few reports of the 5 MHz 8087 operating at 6.67 MHz but never at 7.37 MHz. Of course, an 8 MHz version of the 8087 is available for a few dollars

more and recommended if you want to run at high speed. It is best to remove the 8087 at first until you have everything running properly in your computer, then reinstall the 8087 and see what the effects are.

The 8087 even at the slow clock speed generates an amazing amount of heat. Instead of putting a heat sink on the DMA controller, I suggest you put one on the 8087. One HUG member has mentioned that he moved his CPU card over to the slot next to the power supply so that the air movement will help keep it cool. The circulation of air inside the 150 is pretty poor and the CPU card generates considerably more heat than the floppy controller which enjoys the best circulation. To move the card, the back panel needs to be notched out so that the keyboard connector will fit properly.

(HUG Engineer's Note: You'll get better cooling in your H-150 if you leave the boards where they are and reverse the fan in your power supply. You probably will not need heat sinks on any chips. See my article "Cheap Speed" in the December 1986 Issue of REMark for more details. -Pat Swayne)

RAM Speed

I have found that in general, a computer with 200 nanosecond RAM will run at 7.37 MHz. The computer I am preparing this article on has two banks of 200 ns 64K chips and two banks of 150 ns 256K chips. If you have any 300 ns RAM in your computer get rid of it because it has severe problems running at high speed. The price of 64K ICs has become very reasonable and upgrading to 150 ns is recommended. The above is true for the Heath memory card and may not be true for some of the memory expansion cards available from other manufacturers.

I have also had reports that during boot up at high speed, some computers report the message "+ + + RAM Failure ... " with the mention of some IC number. In at least two cases, this problem was cured by replacing the 100 ns delay line (U456) on the Heath memory card with the 75 ns part (HE-41-21). If you have a Heath 150, the delay line is socketed. However, if you have a Zenith, it is usually soldered to the PC board.

There have been problems reported with the parity circuits on some of these third party expansion cards. Generally, the problem is that the parity circuit is not fast enough and errors are generated which bombs the program into the MFM-150 monitor. The parity circuit on the Heath memory card can be disabled by writing the value of two to port 100 hex. This is easily accomplished in the monitor by typing "o 100,2" at the prompt. Typing "g" should then start execution of your program again. If the memory board will not operate in an H/Z-158, then you can be fairly confident that it will not work in a 150 with a 7.37 MHz clock rate. Again, the problem is less severe when running at 6.67 MHz and may be the CPU speed of choice in this instance.

Exec-PC, which sells a speed-up product specific to the IBM PC, has made available a program that disables the parity generation on these cards. The program, NOPRTYCK.COM, can be downloaded from the Exec-PC bulletin board as part of the file called "PC-SPRINT.ARC" (Exec-PC, P.O. Box 11268, Shorewood, WI 53211, BBS (414) 964-5160). It is in a public area of the bulletin board and you will need ARC to decompress it.

High Speed Verify

Further experience has also shown that the verify switch for COPY does not work with the floppy drives at high speed. This problem does not affect a hard drive in any way. The same problem extends

to FORMAT, DISKCOPY and the MS-DOS command "VERIFY ON". Also in the public area of the Exec-PC BBS is a program called HSPDVER.COM which cures the problem. The program is memory resident, and once loaded, traps the verify flag allowing these programs to work properly. Although the program works properly and can be recommended if you use your floppies a good deal of the time at high speed, the disk verify is disabled.

Accessory Cards

There are becoming more and more clones and other computers (like the 158) which run at increased clock speeds. It is also becoming more apparent that many of the add-on accessory cards now available will not operate properly at the increased CPU speed. As mentioned above, memory cards can be bad actors. For instance, the Heath catalog warns that the Intel Above Board memory expansion card is not suitable for use in the 158. Another troublesome family of cards are internal modems. These problems are not restricted to this modification, but will be a problem whenever the CPU clock speed is greater than stock. In my case, I can report that the Everex internal 1200 baud modem has no trouble running at 7.37 MHz.

At least one multifunction card (the Paradise 5 Pak) uses the 14.13 MHz OSC signal on the bus for its memory timing functions. Even a slight change in CPU clock speed will not be tolerated by the board, since it expects the OSC signal to be three times the CPU clock frequency.

There are plenty more examples of what will and won't work, and it is probably best to inquire as to the suitability of a particular card to high speed operation before purchasing it.

Modifications To The Modification

One of the better suggestions came from a reader, Robert Masky, who suggested that the diagnostic LEDs present on the CPU board be used to communicate with the speed-up board. In this way, infamous "+++ Timer Interrupt Error +++" can be avoided when CTRL-ALT-DEL is used. The modification requires that a wire from the speed-up board be soldered to the CPU board.

The diagnostic LEDs on the CPU board reside at port 0C0 (192 decimal). On boot up or after the CTRL-ALT-DEL key sequence, the ROM diagnostics extinguish the LEDs as each test is passed. We can, therefore, use the LED to switch the PCLOCK frequency instead of the RC delay network on the speed-up board.

Since the status of these LEDs can be controlled by software, it is possible to switch the clock speed by turning an LED on or off. This requires a rather extensive modification to the speed-up board and I won't go into it here. But for you experimenters, writing a value of 37H to port 0C0H will turn on the INT LED and FFH will turn it off.

To connect the speed-up board for automatic PCLOCK switching, the following modification needs to be performed (please refer to the schematic in the June issue for the circuit detail). Cut the trace connected to pin 2 of U3d, and solder a wire to the pin 2 pad long enough to reach the back side of the board behind the INT LED. Alternatively, R5, CR1, and C3 can be removed from the board and the wire soldered to the hole nearest pin 1 of U3 (this pad is connected to pin 2).

Next, locate the CPU board so the solder side is up and the edge card connector is on your left-hand side. Find the row of LEDs and from the left end count to the sixth solder pad. Connect your wire

here using only a small amount of solder. If there is enough room on the component side between the LED body and the PC board, you may be able to use one of those ultra-miniature spring loaded component grabbers, so that you don't have to solder (soldering, however, is much more positive).

Now whenever the computer is reset, the PCLOCK frequency will not switch until the ROM diagnostic is satisfied that the interrupt timer is OK and the LED is extinguished. This switching will operate properly for both a hardware or software reset.

Troubleshooting Techniques

If you have had a problem in operating your computer at high speed (or for that matter any other type of problem), an organized approach to troubleshooting is necessary. I will usually keep a written log of the problem, how I tried to alleviate it, and the result. If not, it can become very confusing. Another very important item to remember is to only change one thing at a time. Make a change, and note the results. This helps if you need to retrace your steps when something (either positive or negative) occurs.

The basic idea is to isolate the problem as much as possible and to simplify the situation. For instance, if you are experiencing RAM errors, and there is a multifunction or memory expansion card in the computer, remove it and note the results (don't forget to reset the DIP switches on the CPU board). Or if the computer will boot at low speed but not at high, try removing any accessory cards until the problem is located (modems, I/O cards, etc.). Once the problem is isolated, you can decide what you want to do to cure it. If the problem is with an accessory board essential to the operation of your computer, then you are pretty much stuck with tracking down the problem with the board.

The diagnostic LEDs on the CPU board can also be of considerable help in isolating a problem. For example, if all the LEDs remain on, then there is an excellent chance that there is something wrong on the speed-up board. Again, if the computer fails to boot at the normal speed, then there is something wrong on the speed-up board itself. Common problems include a short on the board or the 8284A on the left side of the board is not acting properly. Try switching the 8284As around and see if the problem changes or goes away.

Certain solder rosins can also have a deleterious effect on the operation of an electronic circuit involving high frequency signals, and defluxing the board has cured several mysterious problems.

ERRATA

Several observant readers pointed out differences between the schematic and the PC board layout. The leads for the crystal on the CPU board were flip-flopped and the RESET line on U2 is tied to +5V in the schematic, but is connected to the RESET line of the header in the PC board layout. In the case of the crystal, the leads to it are equivalent. Connection of the RESET line of U2 to the RESET line of the header is functionally the same as tying it to +5V, but was easier to layout. A considerable effort was made to keep the PC board single-sided so that it could be easily reproduced. Also the crystal (Y1) frequency was specified as 22.18 MHz when it should have been 22.118 MHz.

And Finally,

At this point, there is no reason that with a little effort there should be any H/Z-150/160s out there that aren't running at 6.67 MHz. By the way, an H/Z-150 running at 6.67 MHz with a V-20 is at least as fast as the H/Z-158 running at 8 MHz. If any advances are made in curing the bus controller blues, the information will be submitted to REMark. ✱

HUG Price List

The following HUG Price List contains a list of all products in the HUG Software Catalog. For a detailed abstract of these products, refer to the issue of REMark specified.

| Part Number | Description of Product | Selling Price | Vol. Issue | Part Number | Description of Product | Selling Price | Vol. Issue | Part Number | Description of Product | Selling Price | Vol. Issue |
|-------------------------------|-----------------------------------|---------------|------------|--|--------------------------------|---------------|------------|--|-------------------------------|---------------|------------|
| HDOS HARDCOPY SOFTWARE | | | | | | | | BUSINESS, FINANCE AND EDUCATION | | | |
| 885-1008 | Volume I Documentation | 9.00 | | 885-5001-37 | CP/M-86 KEYMAP | 20.00 | 51 | 885-1070 | Disk XIV Home Fin H8/H89 | 18.00 | |
| 885-1013 | Volume II Documentation | 12.00 | | 885-5003-37 | CP/M-86 Utilities by PS: | 20.00 | 54 | 885-1071-[37] | MBASIC SmBusPkg H8/H19/H89 | 75.00 | 17 |
| 885-1015 | Volume III Documentation | 9.00 | | 885-5008-37 | CP/M 8080 To 8088 Trans. & HFM | 20.00 | 64 | 885-1131-[37] | HDOS CheapCalc | 20.00 | 47 |
| 885-1037 | Volume IV Documentation | 12.00 | 8 | 885-5009-37 | CP/M-86 HUG Bkgrd Print Spool | 20.00 | 66 | 885-8010 | HDOS Checkoff | 25.00 | 32 |
| 885-1058 | Volume V Documentation | 12.00 | | 885-8018-[37] | CP/M Fast Eddy & Big Eddy | 20.00 | 43 | 885-8021 | HDOS Student's Statistics Pkg | 20.00 | 44 |
| | | | | 885-8019-[37] | DOCUMAT and DOCULIST | 20.00 | 43 | 885-8027 | HDOS SciCalc | 20.00 | 50 |
| | | | | 885-8025-37 | CP/M-85/86 Fast Eddy | 20.00 | 49 | | | | |
| GAMES | | | | ZDOS/MSDOS | | | | | | | |
| HDOS | | | | 885-3005-37 | ZDOS Etchdump | 20.00 | 39 | CP/M | | | |
| 885-1010 | Adventure Disk H8/H89 | 10.00 | 4 | 885-3007-37 | ZDOS CP/Emulator | 20.00 | 47 | 885-1218-[37] | CP/M MBASIC Payroll | 60.00 | 31 |
| 885-1029-[37] | Disk II Games 1 H8/H89 | 18.00 | 8 | 885-3008-37 | ZDOS Utilities | 20.00 | 47 | 885-1233-[37] | CP/M CheapCalc | 20.00 | 47 |
| 885-1093-[37] | D&D H8/H89 Disk | 20.00 | 16 | 885-3010-37 | ZDOS Keymap | 20.00 | 51 | 885-8011-[37] | CP/M Checkoff | 25.00 | 32 |
| 885-1124 | HUGMAN & Movie Animation Pkg | 20.00 | 41 | 885-3022-37 | ZDOS/MSDOS Useful Programs I | 30.00 | 63 | 885-8036-[37] | CP/M Grade | 20.00 | 70 |
| 885-8009-[37] | HDOS & CP/M Galactic Warrior | 20.00 | 32 | 885-3023-37 | ZDOS/MSDOS EZPLOT | 20.00 | 63 | 885-8047-37 | CP/M Accounting System | 20.00 | 85 |
| 885-8022 | HDOS SHAPES | 16.00 | 45 | 885-3031-37 | ZDOS/MSDOS Graphics | 20.00 | 69 | ZDOS/MSDOS H/Z100 ONLY | | | |
| 885-8032-[37] | HDOS Castle | 20.00 | 59 | 885-3037-37 | MSDOS Z-100 PC Emulator II | 60.00 | 76 | 885-3006-37 | ZDOS CheapCalc | 20.00 | 47 |
| | | | | 885-3039-37 | ZDOS/MSDOS HelpScreen | 20.00 | 82 | 885-3013-37 | ZDOS Checkbook Manager | 20.00 | 54 |
| CP/M | | | | 885-3042-37 | MSDOS ZPC Upgrade Disk | 20.00 | 83 | 885-3018-37 | ZDOS Contest Spreadsheet Disk | 25.00 | 58 |
| 885-1206-[37] | CP/M Games Disk | 20.00 | 11 | 885-8029-37 | ZDOS Fast Eddy | 20.00 | 53 | 885-8028-37 | ZDOS SciCalc | 20.00 | 50 |
| 885-1209-[37] | CP/M MBASIC D&D | 20.00 | 19 | 885-8035-37 | MSDOS DOCUMAT and DOCULIST | 20.00 | 70 | 885-8030-37 | ZDOS Mathflash | 20.00 | 55 |
| 885-1211-[37] | CP/M Sea Battle | 20.00 | 20 | 885-8035-37 | MSDOS DOCUMAT and DOCULIST | 20.00 | 70 | 885-8043-37 | MSDOS Calc | 20.00 | 80 |
| 885-1220-[37] | CP/M Action Games | 20.00 | 32 | 885-8041-37 | ZDOS/MSDOS Orbits | 25.00 | 75 | 885-8048-37 | ZDOS/MSDOS Accounting System | 20.00 | 85 |
| 885-1222-[37] | CP/M Adventure | 10.00 | 35 | H/Z100 ZDOS/MSDOS - H/Z100 PC MSDOS | | | | | | | |
| 885-1227-[37] | CP/M Casino Games | 20.00 | 38 | 885-3012-37§§ | ZDOS HUG Editor | 20.00 | 52 | | | | |
| 885-1228-[37] | CP/M Fast Action Games | 20.00 | 39 | 885-3014-37§§ | ZDOS/MSDOS Utilities II | 20.00 | 54 | | | | |
| 885-1236-[37] | CP/M Fun Disk I | 20.00 | 55 | 885-3016-37§ | ZDOS/MSDOS Adventure | 10.00 | 57 | | | | |
| 885-1248-[37] | CP/M Fun Disk II | 35.00 | 69 | 885-3020-37§ | MSDOS HUG Menu System | 20.00 | 62 | | | | |
| ZDOS | | | | 885-3021-37§§ | ZDOS/MSDOS Cardcat | 20.00 | 63 | | | | |
| 885-3004-37 | ZDOS ZBASIC Graphic Games | 20.00 | 37 | 885-3024-37§ | ZDOS/MSDOS 8080 To 8088 Trans. | 20.00 | 64 | | | | |
| 885-3009-37 | ZDOS ZBASIC D&D | 20.00 | 50 | 885-3025-37§§ | ZDOS/MSDOS Misc. Utilities | 20.00 | 64 | | | | |
| 885-3011-37 | ZDOS ZBASIC Games Disk | 20.00 | 52 | 885-3029-37§§ | ZDOS/MSDOS HUG Bg. Print Spool | 20.00 | 66 | | | | |
| 885-3017-37 | ZDOS Contest Games Disk | 25.00 | 58 | 885-3035-37§§ | MSDOS SPELLS & SPELL5F | 20.00 | 72 | | | | |
| 885-8042-37 | ZDOS/MSDOS Poker Party | 20.00 | 77 | 885-3038-37§ | ZDOS/MSDOS DEBUG Support Util | 20.00 | 77 | | | | |
| UTILITIES | | | | 885-3040-37§ | MSDOS HADES | 40.00 | 83 | | | | |
| HDOS | | | | 885-3041-37§ | MSDOS ScreenDump | 20.00 | 83 | | | | |
| 885-1025 | Runoff Disk H8/H89 | 35.00 | | 885-8039-37§§ | MSDOS DPATH | 20.00 | 74 | | | | |
| 885-1063 | Floating Point Disk H8/H89 | 18.00 | | 885-8040-37§§ | MSDOS HELP Programs | 20.00 | 74 | | | | |
| 885-1079-[37] | HDOS Page Editor | 25.00 | 15 | 885-8045-37§§ | MSDOS MATT | 20.00 | 80 | | | | |
| 885-1082 | Programs for Printers H8/H89 | 20.00 | | 885-8046-37§ | MSDOS ASM Language Utilities | 20.00 | 82 | | | | |
| 885-1089-[37] | Disk XVIII Misc H8/H89 | 20.00 | 20 | § All program files run on both | | | | | | | |
| 885-1105 | HDOS Device Drivers H8/H89 | 20.00 | 24 | §§ Program files run partially on both | | | | | | | |
| 885-1116 | HDOS Z80 Debugging Tool | 20.00 | 27 | PC/IBM COMPATIBLE | | | | | | | |
| 885-1119-[37] | BHBASIC Support | 20.00 | 29 | 885-6001-37 | MSDOS Keymapper | 20.00 | 59 | | | | |
| 885-1120-[37] | HDOS 'WHEW' Utilities | 20.00 | 33 | 885-6002-37 | CP/Emulator II & ZEmulator | 20.00 | 59 | | | | |
| 885-1121 | HDOS Hard Sec Sup Pkg 2 Disks | 30.00 | 37 | 885-6003-37 | MSDOS EZPLOT | 20.00 | 65 | | | | |
| 885-1126 | HDOS Utilities by PS: | 20.00 | 42 | 885-6004-37 | MSDOS CheapCalc | 20.00 | 67 | | | | |
| 885-1127-[37] | HDOS Soft Sector Support Pkg | 30.00 | 45 | 885-6005-37 | MSDOS Skyviews | 20.00 | 67 | | | | |
| 885-1135-[37] | HDOS Variety Pkg | 20.00 | 76 | 885-6006-37 | MSDOS Cardcat | 20.00 | 69 | | | | |
| 885-8001 | SE (Screen Editor) | 25.00 | 28 | 885-6007-37 | MSDOS DND (Dung. & Dragons) | 20.00 | 70 | | | | |
| 885-8004 | UDUMP | 35.00 | 28 | 885-6009-37 | MSDOS Screen Saver Plus | 20.00 | 76 | | | | |
| 885-8006 | HDOS SUBMIT | 20.00 | 31 | 885-6010-37 | MSDOS HAM HELP | 20.00 | 86 | | | | |
| 885-8007 | EZITRANS. | 30.00 | 30 | 885-8033-37 | MSDOS Fast Edit | 20.00 | 62 | | | | |
| 885-8017 | HDOS Programmers Helper | 16.00 | 42 | 885-8037-37 | MSDOS Grade | 20.00 | 70 | | | | |
| 885-8024 | HDOS BHBASIC Utilities Disk | 16.00 | 46 | 885-8044-37 | MSDOS TCSPELL | 20.00 | 79 | | | | |
| CP/M | | | | 885-8049-37 | MSDOS Accounting System | 20.00 | 85 | | | | |
| 885-1212-[37] | CP/M Utilities H8/H89 | 20.00 | 21 | PROGRAMMING LANGUAGES | | | | | | | |
| 885-1213-[37] | CP/M Disk Utilities H8/H89 | 20.00 | 22 | HDOS | | | | | | | |
| 885-1217-[37] | HUG Disk Duplication Utilities | 20.00 | 26 | 885-1078-[37] | HDOS Z80 Assembler | 25.00 | 21 | | | | |
| 885-1223-[37] | HRUN HDOS Emulator 3 Disks | 40.00 | 37 | 885-1085 | PILOT Documentation | 9.00 | | | | | |
| 885-1225-[37] | CP/M Disk Dump & Edit Utility | 30.00 | 40 | 885-1086-[37] | Tiny HDOS PASCAL H8/H89 | 20.00 | 13 | | | | |
| 885-1226-[37] | CP/M Utilities by PS: | 20.00 | 40 | 885-1132-[37] | HDOS Tiny BASIC Compiler | 25.00 | 59 | | | | |
| 885-1229-[37] | XMET Robot Cross Assembler | 20.00 | 40 | 885-1134 | HDOS SMALL-C Compiler | 30.00 | 63 | | | | |
| 885-1230-[37] | CP/M Function Key Mapper | 20.00 | 42 | CP/M | | | | | | | |
| 885-1231-[37] | Cross Ref Utilities for MBASIC | 20.00 | 43 | 885-1215-[37] | CP/M BASIC-E | 20.00 | 26 | | | | |
| 885-1235-37 | CP/M COPYDOS | 20.00 | 54 | MSDOS | | | | | | | |
| 885-1237-[37] | CP/M Utilities | 20.00 | 55 | 885-3026-37 | MSDOS SMALL C Compiler | 30.00 | 65 | | | | |
| 885-1245-37 | CP/M-85 KEYMAP | 20.00 | 63 | | | | | | | | |
| 885-1246-[37] | CP/M HUG File Manager & Utilities | 20.00 | 64 | | | | | | | | |
| 885-1247-37 | CP/M-85 HUG Bkgrd Print Spooler | 20.00 | 67 | | | | | | | | |

Continued on Page 55



HUG NEW PRODUCTS

- HEWLETT PACKARD LASERJET+/500+
- DEC LN03/LN03+ LASER
- DEC LA100/LA210
- TOSHIBA 3-IN-ONE SERIES (P1341/P341/P351)
- EPSON LQ SERIES (800/1000/1500/2500)
- C. ITOH M24LQ/1570

Also included is a skeletal version of the source code for those wishing to adapt SCREENDUMP to their own particular printer model.

Original owners of SCREENDUMP, P/N 885-3041-37, can update their software to this latest version by returning their original disk set along with a check or money order (made out to HUG), to Nancy Strunk, Heath Users' Group, Hilltop Road, Saint Joseph, MI 49085. *

HUG P/N 885-3043-37 SCREENDUMP (Version 3.52) \$30.00

This product is a re-release of HUG P/N 884-3041-37. It has been updated to support many of the newer printers, including laser types. For a complete description, refer to page 43 in the December 1986 issue of REMark. The printers which are currently supported by this new version are as follows:

- C. ITOH 8510/1550
- NEC 8023A
- EPSON MX, RX, EX, and FX series
- IBM PROPRINTER
- STAR MICRONICS GEMINI
- STAR MICRONICS GEMINI 10X
- OKIDATA MICROLINE
- ZENITH/MPI 99/150
- ANADEX SILENT SCRIBE
- IDS PAPER TIGER

Continued from Page 54

| | |
|---------------------------------|---|
| MSDOS H/Z100 - H/Z150 PC | |
| 885-3027-37 | MSDOS HUG PBBS 40.00 66 |
| 885-3028-37 | MSDOS HUG PBBS Source Listing 60.00 66 |
| 885-3033-37 | MSDOS HUG MCP 40.00 71 |

| | |
|----------------------|--|
| MISCELLANEOUS | |
| 885-0004 | HUG Binder 5.75 |
| 885-1221-[37] | Watzman ROM Source Code/Doc 30.00 33 |
| 885-4001 | REMark Vol. I Issues 1-13 20.00 |
| 885-4002 | REMark Vol. II Issues 14-23 20.00 |
| 885-4003 | REMark Vol. III Issues 24-35 20.00 |
| 885-4004 | REMark Vol. IV Issues 36-47 20.00 |
| 885-4005 | REMark Vol. V Issues 48-59 25.00 |
| 885-4006 | REMark Vol. VI Issues 60-71 25.00 |
| 885-4007 | REMark Vol. VII Issues 72-83 25.00 |
| 885-4500 | HUG Software Catalog 9.75 |
| 885-4501 | HUG Software Catalog Update #1 9.75 |
| 885-4600 | Watzman/HUG ROM 45.00 41 |
| 885-3015-37 | ZDOS Skyviews 20.00 55 |
| 885-3036-37 | MSDOS TREE-ID 20.00 77 |

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

TABLE C Product Rating

- 10 - Very Good
- 9 - Good
- 8 - Average

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

- 7 - Has 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/Z89) 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. p/n 885-1103 or 885-1211[-37] SEA BATTLE.

ORDERING INFORMATION

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

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

NOTE

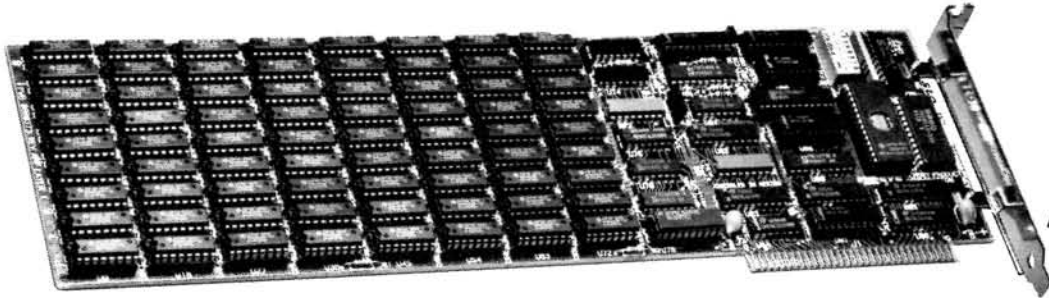
The [-37] means the product is available in hard-sector or soft-sector. Remember, when ordering the soft-sector format, you must include the "-37" after the part number; e.g. 885-1223-37.

Note: 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.** ZPC II contains only one disk. It is a combination of ZPC I and the ZPC Support disk plus added improvements. Thank you.

Expand your **ZENITH** PC Memory with

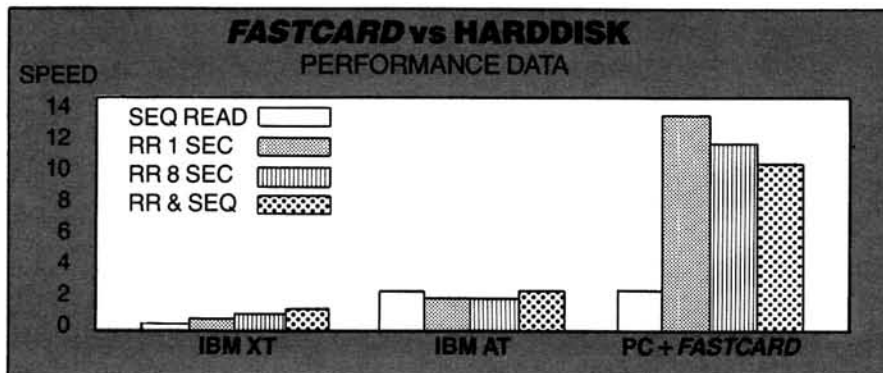
FASTCARD[®]

fully loaded with 2 *MBYTES* for just \$395*



**10 DAY
FREE
TRIAL!**

**FASTCARD speeds disk I/O
by as much as tenfold,
as shown in the following chart.**



This graph shows benchmark test results of a PC equipped with *FASTCARD* versus a PC-XT and PC-AT. These tests were run by a major independent testing laboratory using their standard disk performance benchmark tests.

Specifically designed for compatibility with the 8 MHz Zenith bus, *FASTCARD* provides both extended and expanded memory plus the following standard features:

- Portable between all Zenith models
- Up to 2MB with Split Memory Mapping to
 - Fill memory to 640K
 - Provide Expanded memory over 640K
- Unique Disk Caching
- Ram Disks (up to 8MB)
- Custom Password Security
- Print Buffering
- Built-in Diagnostics and Automatic Fault Tolerance

*Each *FASTCARD III* comes with 2 MBytes of memory for \$395. *FASTCARD IV*, available with 1 MByte, includes serial/parallel ports and a clock/calendar for \$299. *FASTCARD III* without memory is \$140 and *FASTCARD IV* without memory is \$169.

For additional information, call: **PMI** PERIPHERAL MARKETING, INC. (602) 483-7983
7825 E. EVANS RD., #600, SCOTTSDALE, AZ 85260

Printing The Contents Of The Screen On Demand

Bill Rothman

*Charles Rothman Inc.
72 Clifford Street
Providence, RI 02903*

Saving what appears on the screen to a printer is a useful ability provided by the Z-100 (Shift-F12) and the PCs (Shift-PSC). In this article, I will discuss programs which can save the contents of the console's screen on a printer or in a disk file. While the Z-100 (operating in "native mode") needs a program to display the screen's contents (PSC.COM), PCs (referring to IBM clones) have the software built into their BIOS ROM. The techniques used to implement the programs for both machines illustrate several useful bits of knowledge to have when interfacing with MSDOS: TSR routine structure, interrupt-time processing, and screen data accessing, which can be carried over to other programs. Listings for the three versions of the screen printing utilities are provided here. The source code is available from the HUG SIG on CompuServe and the HUG bulletin board in Michigan. Instructions on assembling the programs are at the end of this article.

Print Screen Program For The Z-100 — PSC.COM

The program for the Z-100 is the simplest program of the three presented here. It is essentially a rewritten version of the original provided with ZDOS version 1. The source for the program, called PSC.ASM, is shown in Listing 1.

PSC.COM does the following when it is loaded into memory by typing "PSC" on the command line:

1. It gets the current contents of the PSC interrupt vector, #5, from MSDOS (refer to lines 256-268 in Listing 1). The vector is returned in the ES:BX register pair. If the segment address portion points to the monitor ROM data segment (40h), then I assume that there is no other version of the program around, and that it is all right to load this copy of the program: I am trying to prevent memory from becoming cluttered with versions of this program because the operator kept on typing in "PSC". I don't consider this method to be foolproof or applicable to every TSR program, but it works in this instance for me.
2. If the program is considered ok to load, I have MSDOS cram interrupt vector #5 with the address of this program via its interrupt-vector-set request (lines 269-272). I then load in the offset to the end of this program, round it to the next "para-

graph" boundary (multiple of 16 bytes), and finally convert this value into paragraphs by dividing it by 16 (lines 274-282). Note that if the address doesn't start off on a paragraph boundary and it is not rounded up before it is converted into paragraphs, you will lose the end of your program after the terminate-and-stay-resident request has been made to MSDOS, because MSDOS doesn't round it up either. Further, please note that this algorithm works only for programs (including data and stack areas, if any) which are less than 64K in length. If they aren't, then the number of paragraphs will have to be increased as necessary (adding 4096 for every 64K of memory). I don't really understand why Microsoft considered this method better than just specifying the ending address in a register pair, but who knows? There might even be a good reason for all this hassle.

When a print screen is requested (Shift-F12):

1. The program saves the old stack segment and pointer registers, and then switches to my local stack (lines 82-104). This is done because I cannot be guaranteed either how much stack space the caller is providing (and remember, I don't know who the caller is), or how much stack space I will need during the execution of this program, since I have to call the BIOS directly to have the printing done.
2. Next I set a counter to either 24 or 25, depending upon whether or not the 25th line is considered enabled by the ROM H-19 Emulator (lines 132-141). This bit of information is found by accessing the monitor ROM's data segment and checking a byte in the H-19's data structure. The location of this byte (as well as others which can be of use) is described in the ROM listings Heath provides.
3. Next I get the current setting of the video mapping port and save this value. I then enable access to all video memory, even though I will access only the green plane (lines 142-145).

Now the heart of the program:

4. I fetch the contents of a line on the screen and send it to the line printer (the PRN device, as configured by MSDOS) via the BIOS

call to the PRNFUNC entry point. The characters are fetched from the screen memory's green plane (lines 160-161). (They are stored in the 10th scan line and the attributes are stored in the 11th. Neither are displayed on the screen with the normal CRT controller settings. But, for example, if you are in the interlace mode, they will appear as the garbage, just below each character.) Because my printer, a TI810, does not have graphics (it barely has lower case descenders), I can only print the displayable ASCII characters: all others are changed into spaces (lines 163-168). Note that I get the characters from right to left. This is done to allow me to figure out where the first non-space character is on the line. If I don't do this, but simply print the entire line even if it is all blank, then my printer spends lots of time moving the print head across the page, "printing" all the spaces, wasting lots of time for a typical screen (lines 197-200). Printing the line is accomplished by sending each character, one at a time, to the BIOS (lines 229-243). Note that the program will hang if the printer is offline, out of paper, or out of action, for whatever reason.

5. When I have completed the screen image, I print a form feed to eject the current page (lines 222-223), and then restore the original video mapping port state (lines 225-226).
6. All that is left now is to restore the stack registers to the way I found them and return to MSDOS (lines 108-124).

Listing 1

```

1  page      ,132
2  title    New Version of Print Screen Utility
3
4  ; psc.asm
5  ;
6  ; PSC is a print screen utility. It intercepts INT 5
7  ; from the BIOS (generated when <shift f12> is struck),
8  ; and prints the contents of the screen onto the PRN
   device.
9  ;
10 ; This program is an extensively rewritten version of
11 ; the original one supplied by Heath/Zenith. The pro-
12 ; gram is for my TI810 dot matrix printer, which has
13 ; no graphic mode or graphic characters: all graphics
   and special characters are changed to spaces.
14 ;
15 ;
16 ; PSC
17 ;
18 ; To activate:
19 ;
20 ; Press SHIFT and F12 simultaneously.
21 ;
22 ; Once PSC has been executed, it remains permanently
23 ; in RAM and does not need to be executed again.
24 ;
25 ;
26 ; Written by:      William B. Rothman,
27 ;                  Heath Cliff Software
28 ;
29 ; Date last updated: 12-Sep-86
30 ;
31
32 ; include e:/asm/defmtr.asm
33 ; include e:/asm/defbios.asm
34 ; include e:/asm/defchr.asm
35 .xlist
36 include e:/asm/defmtr.asm
37 include e:/asm/defbios.asm
38 include e:/asm/defchr.asm
39 .list
40 page
41 COLUMNS = 80          ; Characters/line
42

```

```

43 VID_CMD = 0d8h        ; Video memory control port.
44
45 PAR_CRN = 0e000h     ; Green plane.
46
47 LF      = 0ah        ; Line feed.
48 CR      = 0dh        ; Carriage return.
49 FF      = 0ch        ; Form feed.
50
51 code segment byte public 'code'
52     assume cs:code,ss:code,ds:code,es:nothing
53
54     org 100h
55
56 ;
57 ; Entry point (required to be at this point).
58 ;
59
60 begin:
61     jmp  start        ; Go to real initializa-
                       ; tion code.
62
63 ;
64 ; Variables.
65 ;
66
67 int_ss  dw  0         ; Saved stack values.
68 int_sp  dw  0
69
70 int_cy  dw  0         ; Vertical index.
71 line    db  (COLUMNS+3) dup (0) ; Line buffer.
72 lines   dw  0         ; Number of lines valid on
                       ; screen.
73 state   db  0         ; Video ram port status
                       ; at invocation.
74
75         dw  64 dup (0) ; Internal stack.
76 int_stack label near
77
78 ;
79 ; Int 5 handler:
80 ;
81
82 int_5:
83     push  ds         ; Save DS and AX.
84     push  ax
85
86     mov  ax,cs
87     mov  ds,ax      ; Set up my data segment.
88
89     cli             ; Disable interrupts.
90
91     mov  int_sp,sp  ; Save the stack pointer
                       ; and segment.
92
93     mov  int_ss,ss
94
95     mov  ss,ax      ; Use my larger, local stack
96     mov  sp,offset int_stack
97
98     sti             ; Now allow interrupts.
99
100    push  es         ; Save everything.
101    push  bx
102    push  cx
103    push  dx
104    push  si
105    push  di
106
107    call int_scrn    ; Process the screen.
108
109    pop  di          ; Restore the world.
110    pop  si
111    pop  dx
112    pop  cx
113    pop  bx
114    pop  es
115
116    cli             ; Turn interrupts off
                       ; for a bit.

```



```

116
117     mov  sp,int_sp      ; Restore the old stack.
118     mov  ax,int_ss
119     mov  ss,ax
120
121     pop  ax             ; Restore AX & DS.
122     pop  ds
123
124     iret                ; Back to the BIOS.
125
126 ;
127 ; Int_scrn - Interrupt time Screen Processor.
128 ;
129
130 int_scrn:
131     cld                ; Clear direction
                        ; indicator.
132     mov  lines,24      ; Assume 24 lines.
133     xor  ax,ax         ; Zero extra segment.
134     mov  es,ax        ; Fetch the variable
135     mov  es,es:[MTR_DS] ; containing the status
136     cmp  byte ptr es:[2dbh],0 ; of the 25th lines:
                        ; 00= disabled, FF= enabled.
137     jz   int0
138
139     inc  lines         ; Set lines to 25.
140
141 int0:
142     in   al,VID_CMD    ; Get the video status.
143     mov  state,al     ; Save the current state.
144     mov  al,78h      ; Enable access to the
                        ; memory.
145     out  VID_CMD,al
146
147     mov  ax,PAR_GRN   ; Address the green plane.
148     mov  es,ax
149
150     mov  int_cy,0     ; Start at line 0.
151     xor  bh,bh
152
153 int1:                ; Do a screen line:
154     mov  bl,COLUMNS-1 ; Offset of the last char-
                        ; acter.
155     mov  di,COLUMNS-1 ; Offset to the end of the
                        ; line.
156
157     xor  dx,dx        ; Show no non-spaces found yet.
158
159 int2:
160     mov  al,es:[bx+9*128] ; Get the character's
                        ; font index.
161     add  al,' '       ; Then make it into the char-
                        ; acter.
162
163     test al,80h      ; H19 graphics are numbered
                        ; 80h - Ah.
164     jz   int3        ; If printable, go.
                        ; (My printer can't.)
165
166     mov  al,' '       ; Not printable: make it
                        ; a space.
167
168 int3:
169     cmp  al,' '       ; Is the character a space?
170     jne  int4        ; If not, go.
171
172     and  dx,dx        ; Any non-space found yet?
173     jz   int5        ; If no, go.
174
175 int4:
176     mov  line[di],al  ; Save the character.
177
178     and  dx,dx        ; Non-space found yet?
179     jnz  int5        ; If yes, just go.
180
181     mov  dx,di        ; Else keep the offset to
                        ; the non-space.
182
183 int5:
184     dec  bl           ; Go to next character.
185     dec  di           ; Decrement the offset to
                        ; the line.
186     jge  int2        ; Repeat this sequence for
                        ; all chars.
187
188 ; Reached the start of character line:
189
190     and  dx,dx        ; See if there is anything
                        ; to print.
191     jz   int8        ; If not, go.
192
193     mov  cx,dx        ; Set up the up the counter
194     inc  cx           ; by putting offset into CX
                        ; and adding 1 to it.
195     mov  si,offset line ; Get address of first
                        ; character.
196
197 int7:
198     lodsb              ; al= ds:[si++]. Get the
                        ; character.
199     call print         ; Print it.
200     loop int7         ; Loop until CX= 0, when
                        ; all is done.
201
202 int8:
203     mov  al,LF        ; Print LF.
204     call print
205
206     mov  al,CR        ; Print CR for end of line.
207     call print
208
209     mov  ax,int_cy    ; Get vertical index.
210     inc  ax           ; Do next line.
211     cmp  ax,lines     ; End of screen?
212     je   int9        ; Yes, go.
213
214     mov  int_cy,ax    ; Else store new value and
215     shl  al,1         ; fix up pointer to screen
                        ; data.
216
217     shl  al,1
218     mov  bh,al
219     jmp  short int1   ; Now loop back.
220
221 int9:
222     mov  al,FF        ; Print FF for end of page.
223     call print
224
225     mov  al,state     ; Restore the previous state.
226     out  VID_CMD,al
227     ret              ; Done with screen: return.
228
229 print:
230     mov  ah,CHR_WRITE ; Have write done.
231
232 prn0:
233     push ax           ; Save the registers.
234     push cx
235     push si
236     call BIOS_PRNFUNC ; Call the bios to print char.
237     pop  si          ; Restore the registers.
238     pop  cx
239     pop  ax
240
241     jc   prn0        ; If error, try again.
242
243     ret              ; Return.
244
245 ; Mark end of resident portion
246
247 int_end label  near
248
249 ;
250 ; This routine loads in the PSC routine if it is not
251 ; already loaded into memory.
252 ;

```

```

253 start:
254     mov     sp,offset int_stack ; Get offset to
                                local stack.

255
256     mov     ah,35h    ; See if PSC already loaded.
257     mov     al,5     ; Returns vector in ES:BX.
258     int     21h
259
260     mov     ax,es     ; If the interrupt is no
261     cmp     ax,0040h ; longer vectored into
                                monitor, exit now.
262     je      st0      ; Else go load this routine.
263
264     mov     ah,4ch    ; Return to MSDOS now.
265     mov     al,1     ; Show error.
266     int     21h
267
268 st0:
269     mov     ah,25h    ; Function code type int.
270     mov     al,5     ; Set interrupt #5.
271     mov     dx,offset int_5 ; Put offset into DX;
272     int     21h     ; address is in DS:DX.
273
274     mov     ah,31h    ; Keep this process.
275     mov     al,0     ; Return zero as completion
                                code.
276     mov     dx,offset int_end ; Last byte of this
                                program.
277     add     dx,15    ; Round up to next paragraph
                                boundary.
278     shr     dx,1     ; Make into "paragraphs".
279     shr     dx,1
280     shr     dx,1
281     shr     dx,1
282     int     21h     ; Keep the process.
283
284 code  ends
285     end     begin

```

"fpsc" by itself on the command line will act as PSC, sending its output to the PRN device as configured by MSDOS.

"fpsc fn" will send the output to the file "fn", rewriting the file upon each invocation (that is, each Shift F-12). Note that unless you specify the drive and path, it will put the "fn" at the currently logged in disk and directory. Since you might not know what the default disk and directory you are going to be in while running a particular program whose screen you wish to store, I suggest that you fully qualify the name of the file (device, path, and name) when you set "fn".

"fpsc /a fn" will send the output to the file "fn" also, but will append the contents to the previous contents of the file. If the file doesn't exist, it will automatically create it. Each screen image will be separated from the other by a form feed.

"fpsc /h" will display a help message on the screen. Note that requesting the help text does NOT cause the program to be made resident in memory, and does not affect any of the settings of a copy of the program already in memory.

The program works a little differently than PSC.COM:

1. It starts off by getting the system's switch character, which is probably a "/", but could be, and is for my system, a "-" (refer to lines 438-442 in Listing 2). I save the value and modify the help text accordingly, which is a common practice of mine. The options presented above assume the more common "/" switch. Microsoft may have abandoned the "-" as a switch character, but I haven't.
2. As with PSC.COM before, I get the segment address for the Int 5 vector to see if the program has already been loaded into memory (lines 444-451). If it has not, I set the ES register back to this copy of the program and set a flag (lines 453-455). I then do a compare with a constant in this copy of the program to whatever the ES register points (lines 458-463). (Remember that both the ES and DS registers might be the same.) In any event, if the comparison fails, I set a flag and set the ES register back to this copy of the program (lines 465-467).
3. Now I turn off the append-mode flag (line 470). Because I am using the ES register to access this flag, it will be altered in either this copy of the program or in the already resident version of this program, if there is one. This will also be true for all the other variables I will change, which I consider neat.
4. I then look at the command line passed to this copy of the program (lines 471-515). If help is requested, I display the help text and end the program (lines 554-559). I set or leave reset the append-mode flag, depending upon whether or not the append-mode switch is present (lines 523-525). If a filename was specified, I save it (lines 530-532). Finally, I set or reset the output-flag type (file or printer) (line 533 or line 537). If an invalid switch is specified, I give an error message and exit (lines 517-520).
5. If the program detected some other program using Int 5, I will not allow the program to become resident, and will give an error message to that effect (lines 548-552). Otherwise, if the program found another copy of itself in memory, all of the parameters in the resident copy have now been updated, and

File Oriented PSCs

A serious flaw in MSDOS is that it is not reentrant. This means that it cannot call itself while it is in the middle of one of its own routines. This also means that YOU cannot call it during the middle of some routine which is itself in the middle of an MSDOS routine. This situation occurs with device drivers during a function request and with routines which are called by an interrupt, like the print screen function. Even though, as in this case, the print screen function is actuated by the BIOS, the actual interrupt could come at any time, which means that MSDOS might be in the midst of doing something. This is why I have to call the BIOS directly to have the characters printed in PSC.COM, since the call to the BIOS bypasses MSDOS completely. MSDOS does have an obscure system call which returns the "MSDOS Critical Flag". Getting this flag is supposed to be able to let a program know if it is safe to issue a call to MSDOS. Making this call is in itself, however, enough to kill the system, somewhat reducing the value of the function call. (Like to zip.)

There is a solution of sorts to this vexatious problem in the form of an undocumented function in MSDOS, Int 28h. It is apparently executed by MSDOS after it processes a software interrupt, but before it returns to the caller, signaling that it is all right to do a system call. The print despooler (PRINT.COM) uses the interrupt, which is why it is able to do the necessary I/O operations to read a file from the disk and print it in "background".

File Oriented PSC For The Z-100 — FPSC.COM

FPSC.COM, assembled from FPSC.ASM as shown in Listing 2, can be loaded into memory by typing one of the following three variations on the command line:

the program just returns to MSDOS, tossing away this copy (lines 561-563).

6. If this copy of the program is to be kept, I link into the Int 28h chain (lines 565-576) and then cram the routine's address into the Int 5 vector (lines 578-581). Finally, I calculate the number of paragraphs this program is in length and request that MSDOS keep it here after I return (lines 583-591).

The ability of the program to be able to alter variables in the resident copy of the program allows you to change the way the program executes (sending a screen image to the printer, to a file, and to a file in append mode) as many times as you like after the copy of the program has been made permanently resident. (Sort of like the HDOS SET facility at its best.)

Operation of this program upon a Shift-F12 is a little different than PSC:

1. In PSC.COM, I executed the print screen function whenever the Int 5 was executed by the BIOS. In this program, all that happens upon an Int 5 is that a flag is set (refer to lines 130-132 in Listing 2). For every Int 28h, my program gets control long enough to check the state of that flag (lines 150-151). If it is set, I do the print screen function at that time (line 154). I also have a lock flag, there more for prudence than demonstrated necessity, which prevents the print screen function from being executed while I am in the middle of a previous one (lines 147-148 and line 153). When the print screen is done, I reset both flags before returning (line 155-159).

While I am aware that the Heath BIOS also has its own flag to prevent it from executing the Int 5 print screen function before a previous one is completed, I don't know how often during the file I/O, which my program will do the Int 28h, will be executed. I do know that I do not want my Int 28h code to try to execute another print screen function before the current one ends.

2. If the output is to go to a file, the file is opened. There are two versions of the open: If append mode is in effect, the file is opened for write (lines 226-228). If the call succeeds, a seek is done to the end of the file and the program proceeds to step 5 (lines 234-241). If the call fails, then the program goes to the next step.
3. If the output is to a file, and either append mode is off or the file does not already exist, a call to create is made (lines 246-250). If this fails, an error message is issued and the program returns to MSDOS (lines 374-376). If the call succeeds, it goes to step 5.
4. If the output is to go to the printer, then the PRN device is opened for write (lines 257-260).

Note that from this point on, whether the output is going to a file or the printer, everything is done in exactly the same fashion, since the file handle mechanism is being used:

5. The screen data is gathered and dispatched (again with trailing spaces eliminated) in a single write via the handle, a line at a time (lines 345-348). At the end of the screen, a form feed is also written to separate the screens from each other (line 340).
6. At the conclusion of the print screen function, the file handle is closed (lines 361-367), the original video state is restored (lines 369-371), the registers are restored, the stack context is switched, and the program returns to MSDOS (lines 192-208 and line 159).

Listing 2

```
1  page      .l32
2  title    New Version of Print Screen Utility
3
4  ; fpsc.asm
5  ;
6  ; FPSC is a print screen utility. It intercepts INT 5
7  ; from the BIOS (generated when <shift f12> is struck),
8  ; and prints the contents of the screen onto the PRN
   device or to a file.
9  ;
10 ; This program is an extensively rewritten version of
11 ; the original one supplied by Heath/Zenith. The program
12 ; is for my TI810 dot matrix printer, which has no
13 ; graphic mode or graphic characters: all graphics and
   special characters are changed to spaces.
14 ;
15 ; The program now allows screens to be printed to a file.
16 ; There are two modes for files: create a new file each
17 ; time and append the screen data to the end of file.
18 ; Note that the screen images are ended with a form feed.
19 ;
20 ; NOTE: This program uses Int 28h to figure out if it
21 ; is all right ***** to do file I/O. Because this
22 ; interrupt is undocumented, there are NO guarantees,
23 ; expressed or implied, about the use of this program.
   YOU HAVE BEEN WARNED!
24 ;
25 ;
26 ; Written by:           William B. Rothman,
27 ;                   Heath Cliff Software
28 ;
29 ; Date last updated:   21-Oct-86
30 ;
31
32 ; include e:/asm/defmtr.asm
33 ; .xlist
34 ; include e:/asm/defmtr.asm
35 ; .list
36 page
37 COLUMNS = 80          ; Characters per screen line.
38
39 VID_CMD = 0d8h         ; Video-memory control port.
40
41 PAR_GRN = 0e000h       ; Green video plane segment address.
42
43 TAB      = 09h         ; Horizontal tab.
44 LF       = 0ah         ; Line feed.
45 FF       = 0ch         ; Form feed.
46 CR       = 0dh         ; Carriage return.
47 CRLF     = 0a0dh       ; CR-LF.
48
49 STDOUT   = 1           ; Handle of standard out.
50 STDERR   = 2           ; Handle of standard error.
51
52 code     segment byte public 'code'
53         assume cs:code,ss:code,ds:code,es:nothing
54
55         org      100h
56
57 ;
58 ;         Entry point (required to be at this point).
59 ;
60
61 begin:
62         jmp     start      ; Go to the one time initiali-
                           zation code.
63
64 ;
65 ;         Variables.
66 ;
67
68 flag     db      0        ; PSC wanted? 0= No, 1= Yes.
69
70 working  db      0        ; In PSC function? 0= No, 1= Yes.
71
```

```

72 io_dir db 0 ; Whence goeth I/O? 0= printer,
; l= file.
73
74 fname db 64 dup (0) ; Filename, terminated by
; zero byte.
75
76 prn_nm db '/dev/prn',0 ; Standard printer name.
77 PRN_L equ $ - prn_nm ; Length for ID comparison.
78
79 handle dw 0 ; I/O handle.
80
81 dispt dw 0 ; Displacement & segment for
82 segmt dw 0 ; int 28 return address.
83
84 int_ss dw 0 ; Save area for the old stack's
85 int_sp dw 0 ; segment and offset registers.
86
87 int_cy dw 0 ; Vertical screen index
; (line number).
88
89 line db (COLUMNS+3) dup (0) ; Screen's line
; buffer.
90
91 lines dw 0 ; Number of lines valid on
; screen.
92
93 eos db 0 ; flag: 0= not end of screen,
; l= yes.
94
95 append db 0 ; flag: 0= not append mode,
; l= yes.
96
97 state db 0 ; Video ram port-status at
; invocation.
98
99 err_msg dw 0 ; Offset to error message.
100 err_l dw 0 ; Length of message.
101
102 ; Error messages:
103
104 msg0 db CR,LF,'Fpsc: error in '
105 MSG0L equ $ - msg0
106
107 msg1 db 'seeking to end of the file!',CR,LF
108 MSG1L equ $ - msg1
109
110 msg2 db 'creating the file!',CR,LF
111 MSG2L equ $ - msg2
112
113 msg3 db 'opening the printer!',CR,LF
114 MSG3L equ $ - msg3
115
116 msg4 db 'writing to the file!',CR,LF
117 MSG4L equ $ - msg4
118
119 dw 64 dup (0) ; Internal stack.
120 int_stack label near
121 page
122 ;
123 ; Int 5 handler:
124 ;
125 ; This function merely sets a flag when the Shift-Fcn12
126 ; key combination is pressed and then returns. It has
127 ; to do this since it is probably not safe to do file
; I/O at this time.
128 ;
129
130 int_5:
131 mov cs:flag,1 ; Set the print-screen-
; request flag on.
132 iret ; Back to the BIOS.
133
134 ;
135 ; Int 28 handler:
136 ;
137 ; This routine is called when MSDOS wants input to
138 ; the terminal and it is all right to do disk I/O.
; At least that is the theory.

```

```

139 ;
140 ; In any event, this routine checks the flag which is
141 ; set when the print screen "hot key" is pressed. If
142 ; the flag is set, it does the print screen function.
143 ; If the print screen function is in progress already,
; or the flag is not set, it simply returns.
144 ;
145
146 int_28:
147 cmp cs:working,1 ; In this function?
; (I don't know if
148 je in0 ; this is needed.)
; If yes, go.
149
150 cmp cs:flag,0 ; Print screen func-
; tion wanted?
151 je in0 ; If not, go.
152
153 mov cs:working,1 ; Set flag.
154 call psc_fcn ; Else do it.
155 mov cs:flag,0 ; Turn off flag now.
156 mov cs:working,0 ; Also turn off this
; flag, too.
157
158 in0:
159 jmp dword ptr cs:dispt ; Return back to the
; system now.
160 page
161 ;
162 ; This is the print screen function which is called
163 ; if wanted during an int 28.
164 ;
165
166 psc_fcn:
167 push ds ; Save DS and AX.
168 push ax
169
170 mov ax,cs
171 mov ds,ax ; Set up my data segment.
172
173 cli ; Disable interrupts.
174
175 mov int_sp,sp ; Save the stack pointer
; and segment.
176 mov int_ss,ss
177
178 mov ss,ax ; Use my larger, local stack.
179 mov sp,offset int_stack
180
181 sti ; Now allow interrupts.
182
183 push es ; Save everything.
184 push bx
185 push cx
186 push dx
187 push si
188 push di
189
190 call int_scrn ; Process the screen.
191
192 pop di ; Restore the world.
193 pop si
194 pop dx
195 pop cx
196 pop bx
197 pop es
198
199 cli ; Turn interrupts off
; for a bit.
200
201 mov sp,int_sp ; Restore the old stack.
202 mov ax,int_ss
203 mov ss,ax
204
205 pop ax ; Restore AX & DS.
206 pop ds
207
208 ret ; Return.

```


| | | | | | |
|-----|-----------|--|----------------------------|--|----------------|
| 209 | page | | | | |
| 210 | ; | | | | |
| 211 | ; | Gather the screen's characters and display them on | | | |
| 212 | ; | a printer or write them to a file. | | | |
| 213 | ; | | | | |
| 214 | | | | | |
| 215 | int_scrn: | | | | |
| 216 | cld | ; | Clear direction indicator. | | |
| 217 | mov | eos,0 | ; | Initialize end of screen flag to NO | |
| 218 | mov | handle,0 | ; | and the file handle, as well. | |
| 219 | | | | | |
| 220 | cmp | io_dir,0 | ; | Is the I/O to the printer? | |
| 221 | je | int01 | ; | If yes, go. | |
| 222 | | | | | |
| 223 | cmp | append,0 | ; | Want the append mode? | |
| 224 | je | int00 | ; | If not, go. | |
| 225 | | | | | |
| 226 | mov | ax,3d01h | ; | Open the file for write. | |
| 227 | mov | dx,offset fname | ; | Use the specified file's name. | |
| 228 | int | 21h | ; | Open the file. If it doesn't exist, | |
| 229 | jc | int00 | ; | then go create it. | |
| 230 | | | | | |
| 231 | mov | handle,ax | ; | Else save the handle. | |
| 232 | mov | err_msg,offset msg1 | ; | Set error message address. | |
| 233 | mov | err_1,MSG1L | ; | Also set length. | |
| 234 | mov | bx,ax | ; | BX= file's handle. | |
| 235 | mov | ax,4202h | ; | Move to the end of the file. | |
| 236 | xor | cx,cx | ; | CX:DX= 0... want the end of the file. | |
| 237 | xor | dx,dx | | | |
| 238 | int | 21h | | | |
| 239 | jc | int02 | ; | Go if trouble. | |
| 240 | | | | | |
| 241 | jmp | short int04 | ; | Else continue. | |
| 242 | | | | | |
| 243 | int00: | | ; | Create the file: | |
| 244 | mov | err_msg,offset msg2 | ; | Set error message address. | |
| 245 | mov | err_1,MSG2L | ; | Also set length. | |
| 246 | mov | ah,3ch | ; | Create the file (or kill the old one). | |
| 247 | xor | cx,cx | ; | Use no special attributes. | |
| 248 | mov | dx,offset fname | ; | Use the specified file's name. | |
| 249 | int | 21h | ; | name. | |
| 250 | jnc | int03 | ; | If ok, go. | |
| 251 | | | | | |
| 252 | jmp | short int02 | ; | If trouble, go. | |
| 253 | | | | | |
| 254 | int01: | | ; | Use the printer: | |
| 255 | mov | err_msg,offset msg3 | ; | Set error message address. | |
| 256 | mov | err_1,MSG3L | ; | Also set length. | |
| 257 | mov | ax,3d01h | ; | Open printer for write. | |
| 258 | mov | dx,offset prn_nm | ; | DS:DX is the address of the printer | |
| 259 | int | 21h | ; | name. | |
| 260 | jnc | int03 | ; | If ok, continue. | |
| 261 | | | | | |
| 262 | int02: | | | | |
| 263 | jmp | int14 | ; | If trouble, go. | |
| 264 | | | | | |
| 265 | int03: | | | | |
| 266 | mov | handle,ax | ; | Else save the handle. | |
| 267 | | | | | |
| 268 | int04: | | | | |
| 269 | mov | err_msg,offset msg4 | ; | Set error message address. | |
| 270 | mov | err_1,MSG4L | ; | Also set length. | |
| 271 | mov | lines,24 | ; | Assume 24 lines on the screen. | |
| 272 | xor | ax,ax | ; | Zero extra segment. | |
| 273 | mov | es,ax | ; | Fetch the variable | |
| 274 | mov | es,es:[MTR_DS] | ; | status of the 25th line: | containing the |
| 275 | cmp | byte ptr es:[2dbh],0 | ; | 00= disabled, FF= enabled. | |
| 276 | je | int05 | | | |
| 277 | | | | | |
| 278 | inc | lines | ; | Set number of lines on screen to 25. | |
| 279 | | | | | |
| 280 | int05: | | | | |
| 281 | in | al,VID_CMD | ; | Get the current video status. | |
| 282 | mov | state,al | ; | Save it. | |
| 283 | mov | al,78h | ; | Enable access to the memory. | |
| 284 | out | VID_CMD,al | | | |
| 285 | | | | | |
| 286 | mov | ax,PAR_GRN | ; | Address the green plane, which is | |
| 287 | mov | es,ax | ; | where the characters are stored. | |
| 288 | | | | | |
| 289 | mov | int_cy,0 | ; | Start at line 0 on the screen. | |
| 290 | xor | bh,bh | | | |
| 291 | | | | | |
| 292 | int06: | | ; | Do a screen line: | |
| 293 | mov | bl,COLUMNS-1 | ; | Offset of the last character. | |
| 294 | mov | di,COLUMNS-1 | ; | Offset to the end of the line. | |
| 295 | | | | | |
| 296 | xor | si,si | ; | Show no non-spaces found yet. | |
| 297 | | | | | |
| 298 | int07: | | | | |
| 299 | mov | al,es:[bx+9*128] | ; | Get the character's font index. | |
| 300 | add | al,' ' | ; | Then make it into the character. | |
| 301 | | | | | |
| 302 | cmp | io_dir,1 | ; | If IO to a file, show all. | |
| 303 | je | int08 | | | |
| 304 | | | | | |
| 305 | test | al,80h | ; | H19 graphics are numbered 80h - 1Ah. | |
| 306 | jz | int08 | ; | If printable, go. (My printer can't.) | |
| 307 | | | | | |
| 308 | mov | al,' ' | ; | Not printable: make it a space. | |
| 309 | | | | | |
| 310 | int08: | | | | |
| 311 | cmp | al,' ' | ; | Is the character a space? | |
| 312 | jne | int09 | ; | If not, go. | |
| 313 | | | | | |
| 314 | and | si,si | ; | Any non-space found yet? | |
| 315 | jz | int10 | ; | If no, go. | |
| 316 | | | | | |
| 317 | int09: | | | | |
| 318 | mov | line[di],al | ; | Save the character. | |
| 319 | | | | | |
| 320 | and | si,si | ; | Non-space found yet? | |
| 321 | jnz | int10 | ; | If yes, just go. | |
| 322 | | | | | |
| 323 | lea | si,[di+1] | ; | Else keep the offset to the non-space. | |
| 324 | | | | | |
| 325 | int10: | | | | |
| 326 | dec | bl | ; | Go to next character. | |
| 327 | dec | di | ; | Decrement the offset to the line. | |
| 328 | jge | int07 | ; | Repeat this sequence for all chars. | |
| 329 | | | | | |
| 330 | ; | Reached the start of character line: | | | |
| 331 | | | | | |
| 332 | mov | word ptr line[si],CRLF | ; | Put in a CR-LF. | |

```

333
334 lea cx,[si+2] ; Set up the counter.
335 inc int_cy ; Up index to the next
; line and then
336 mov ax,int_cy ; get it.
337 cmp ax,lines ; Are we at the end of
; screen?
338 jne int11 ; If not, go.
339
340 mov byte ptr line[si+2],FF ; Else store
; form feed for
; end of page.
341 inc cx ; Up length to print.
342 mov eos,1 ; Turn on the end-of-
; screen flag.
343
344 int11: ; Write out the line:
345 mov dx,offset line ; Get address of first
; character.
346 mov ah,40h ; Do the write: CX=
; length, DS:DX=
347 mov bx,handle ; buffer, BX= handle,
; AH= op code.
348 int 21h
349 jc int14 ; Go if trouble.
350
351 cmp eos,1 ; End of screen?
352 je int12 ; If yes, go.
353
354 mov ax,int_cy ; Else get the line
; index and set the
355 shl al,1 ; pointer to the next
; line of screen
356 shl al,1 ; data.
357 shl al,1
358 mov bh,al
359 jmp short int06 ; Now loop back.
360
361 int12:
362 mov bx,handle ; See if there is a
; handle to close.
363 and bx,bx
364 jz int13 ; If not, go.
365
366 mov ah,3eh ; Else close whatever
; it is.
367 int 21h
368
369 int13:
370 mov al,state ; Restore the original
; video port
371 out VID_CMD,al ; state.
372 ret ; Done with screen: return.
373
374 int14: ; Error found:
375 call int15 ; Display the appropriate
; error message.
376 jmp short int12 ; Go close files, etc.
377
378 int15: ; Error message display
; routine:
379 mov ah,40h ; Write out the error
; message beginning.
380 mov cx,MSG0L ; CX= length.
381 mov dx,offset msg0 ; DS:DX= error message
; address.
382 mov bx,STDERR ; BX= handle.
383 int 21h
384
385 mov ah,40h ; Write out the error
; message text.
386 mov cx,err_1 ; CX= length.
387 mov dx,err_msg ; DS:DX= error message
; address.
388 mov bx,STDERR ; BX= handle.
389 int 21h
390 ret ; Return.
391
392 int_end label near ; This is the end of
; resident code.
393
394 ;
395 ; This routine loads in the FPSC routine if it is
396 ; not already loaded into memory. It also has the
397 ; ability to update the in-memory copy of the program,
398 ; if there is one, with new parameters. The routine can
; also provide a help screen.
399 ;
400 ; Note that the program makes an effort to verify that,
401 ; if it finds that something pointed to by Int #5 is
402 ; loaded into memory, it is a copy of this program.
403 ; This may not be foolproof. [ Another warning. ]
404 ;
405
406 msg5 db CR,LF,'fpsc.com',TAB,'Version 1.03 by
; Heath Cliff Software'
407 db CR,LF,LF,'This TSR program prints the
; contents of the screen'
408 db CR,LF,'when Shift-F12 is pressed. It
; has the ability to print'
409 db CR,LF,'either to a file or to the PRN
; device.'
410 db CR,LF,LF,'Options:'
411 db CR,LF,TAB,'fpso',TAB,TAB,'Prints the
; screen to the PRN device.'
412 db CR,LF,TAB,'fpso fn',TAB,TAB,'Prints
; the screen to the file fn.'
413 db CR,LF,TAB,'fpso '
414 msg5s0 db 0
415 db 'a fn',TAB,'Prints the screen to the
; file fn,'
416 db CR,LF,TAB,TAB,TAB,'appending each screen
; to the end of the file.'
417 db CR,LF,TAB,'fpso ? ',TAB,TAB,'Displays
; this message.'
418 db CR,LF,TAB,' '
419 msg5s1 db 0
420 db 'h'
421 MSG5L equ $ - msg5
422
423 msg6 db 'invalid switch specified!',CR,LF
424 MSG6L equ $ - msg6
425
426 msg7 db 'some other program is already using
; the Print Screen '
427 db 'interrupt',CR,LF,TAB,' vector...
; can not load this '
428 db 'program!',CR,LF
429 MSG7L equ $ - msg7
430
431 switch db 0 ; MSDOS switch character.
432
433 ok db 0 ; Flag: 0= ok to go, 1= alien
; in memory.
434
435 start:
436 mov sp,offset int_stack ; Set the offset
; to my local stack.
437
438 mov ax,3700h ; Get the switch character.
439 int 21h
440 mov switch,d1 ; Save it.
441 mov msg5s0,d1
442 mov msg5s1,d1
443
444 mov ah,35h ; See if this program is
; already loaded.
445 mov al,5 ; This function returns vector
; in ES:BX.
446 int 21h
447
448 xor dx,dx ; Assume old copy of the
; program.
449 mov ax,es ; If the interrupt is no
; longer
450 cmp ax,0040h ; vectored into monitor,

```

```

451         jne  st00      ; this program
                    ; has been loaded. Fix the
                    ; parms only.
452
453         mov  ax,cs     ; Else reset the ES register
                    ; back to
454         mov  es,ax    ; this copy of the program.
455         inc  dx       ; Show new copy of the program.
456
457 st00:
458         mov  ok,0     ; Assume ok to proceed.
459         mov  si,offset prn_nm ; Make sure that we
                    ; actually have this
460         mov  di,si    ; program in memory.
461         mov  cx,PRN_L ; DS:SI : ES:DI for CX bytes:
462         repz cmpsb    ; Do the compare.
463         je   st01     ; If ok, go.
464
465         mov  ax,cs     ; Else reset the ES register
                    ; back to
466         mov  es,ax    ; this copy of the program
                    ; and show
467         mov  ok,1     ; alien in memory.
468
469 st01:
470         mov  es:append,0 ; Turn off append mode.
471         mov  si,80h   ; Set SI to offset of the
                    ; command line.
472         xor  ch,ch    ; Set CX to the length of
                    ; the command
473         mov  cl,[si]  ; line.
474         and  cl,cl    ; Is it zero?
475         jz   st07     ; If yes, go.
476
477 st02:
478         inc  si       ; Up the pointer.
479         cmp  byte ptr [si],' ' ; Is it a space?
480         je   st03     ; If yes, skip it.
481
482         cmp  byte ptr [si],TAB ; Is it a tab?
483         jne  st04     ; If not, go.
484
485 st03:
486         dec  cl       ; Skip over space or tab:
                    ; Decrement length so far.
487         jnz  st02    ; If not zero, keep on looking.
488
489         jmp  short st07 ; Else go.
490
491 st04:
492         and  cl,cl    ; Got anything?
493         jz   st07     ; If not, just go.
494
495         mov  al,byte ptr [si] ; Get the character.
496         cmp  al,'?'   ; Help wanted?
497         je   st10     ; If yes, go.
498
499         cmp  al,switch ; Is it the switch character?
500         jne  st06     ; If not, go.
501
502         inc  si       ; Up the pointer. Get next
                    ; character.
503         lodsb        ; al= ds:[si++]. Up the
                    ; ptr again.
504
505         cmp  al,'h'   ; Help wanted?
506         je   st10     ; If yes, go.
507
508         cmp  al,'H'   ; Help wanted?
509         je   st10     ; If yes, go.
510
511         cmp  al,'a'   ; Append mode wanted?
512         je   st05     ; If yes, go.
513
514         cmp  al,'A'   ; Append mode wanted?
515         je   st05     ; If yes, go.
516
517         mov  err_msg,offset msg6 ; Set error message
                    ; address.
518
518         mov  err_1,MSG6L ; Also set length.
519         call int15     ; Display error message.
520         jmp  short st11 ; Go now.
521
522 st05:
523         mov  es:append,1 ; Set flag to yes.
524         sub  cl,3      ; Decrement length accordingly.
525         jg  st02      ; If more left, go back
                    ; for it.
526
527         jmp  short st07 ; Else just go clear I/O
                    ; dir flag.
528
529 st06:
530         mov  di,offset fname ; DI= address of the
                    ; filename.
531         rep  movsb    ; es:[di++] = ds:[si++],
                    ; while cx-- > 0.
532         mov  byte ptr es:[di],0 ; End string with 0.
533         mov  es:io_dir,1 ; Indicate that I/O will
                    ; be to a file.
534         jmp  short st08 ; Go.
535
536 st07:
537         mov  es:io_dir,0 ; Indicate the I/O is to
                    ; the printer.
538
539 st08:
540         cmp  ok,1     ; Ok to either load this
                    ; copy or change
541         je   st09     ; the copy in memory?
                    ; If not, go.
542
543         and  dx,dx    ; Now see if loading this
                    ; copy of the
544         jz   st11     ; program. If not, return
                    ; to MSDOS.
545
546         jmp  short st12 ; Else go install this program.
547
548 st09:
                    ; Can't set options if
                    ; alien present:
549         mov  err_msg,offset msg7 ; Set error message
                    ; address.
550         mov  err_1,MSG7L ; Also set length.
551         call int15     ; Display error message.
552         jmp  short st11 ; Go now.
553
554 st10:
                    ; Help function:
555         mov  ah,40h   ; Write out help text.
556         mov  cx,MSG5L ; CX= length.
557         mov  dx,offset msg5 ; DS:DX= text address.
558         mov  bx,STDOUT ; BX= handle.
559         int  21h
560
561 st11:
562         mov  ax,4c00h ; Return to MSDOS now.
563         int  21h
564
565 st12:
                    ; Install this copy of the program:
566         mov  ah,35h   ; Function code type.
567         mov  al,28h   ; Get interrupt vector #28.
568         int  21h     ; Returned in ES:BX.
569
570         mov  dispt,bx ; Save the ES:BX vector.
571         mov  segmt,es
572
573         mov  ah,25h   ; Function code type.
574         mov  al,28h   ; Set interrupt vector #28.
575         mov  dx,offset int_28 ; Address in DS:DX.
576         int  21h
577
578         mov  ah,25h   ; Function code type int.
579         mov  al,5     ; Set interrupt #5.
580         mov  dx,offset int_5 ; Put offset into DX;
581         int  21h     ; address is in DS:DX.
582
583         mov  ah,31h   ; Keep this process.

```

```

584     mov     al,0           ; Return zero as completion
                          ; code.
585     mov     dx,offset int_end ; Last byte of this
                          ; program.
586     add     dx,15         ; Round up to next paragraph
                          ; boundary.
587     shr     dx,1         ; Make into "paragraphs"
                          ; by dividing
588     shr     dx,1         ; the ending address by 16
589     shr     dx,1         ; (16 bytes per paragraph).
590     shr     dx,1
591     int     21h          ; Keep the process.
592
593 code ends
594     end     begin

```

File Oriented PSC For The PC — PCFPSC.COM

The version of FPSC.COM for the PC is called PCFPSC.COM. Listing 3 shows the source code, called PCFPSC.ASM, for this program. It is quite similar to the version for the Z-100, except:

1. The check to see if another print screen utility has already been loaded has been deleted (refer to initial program loading code, lines 438-576, in Listing 3). I found that the default address seemed to be different for a Z-148, a Z-158, and a Z-241. I simply gave up trying to protect the user from wiping out a previous print screen utility and will always either load in this program or update the copy of it which is already in place (unless the user has only requested that the help information be displayed).

Operation of the program (Shift-PSC) differs from the Z-100 version as follows:

1. The PC already has a flag byte defined to prevent a print screen function from being called before a previous one is finished. This byte is zero if the function may be executed, one if the function is being executed at that time, and minus one if an error was detected during the execution of the print screen function. So upon entry of Int 28h, if the print screen function was wanted (as indicated by a flag I set in the Int 5 code (lines 122-124)), the byte in the PC's memory is checked (lines 144-147). Since I am not a PC expert, I have no idea who, if anyone, cares to what this byte is set. In any event, I dutifully set this byte (line 149), like a good DoBee. (I bet THAT dates me.)
2. The current screen mode is requested from the console using Int 10h (lines 267-268). This must be done to obtain the video page number in use and the number of columns per line.
3. The current cursor location is requested and is stored for restoration at the conclusion of the function (lines 272-275).
4. Now to actually fetch the characters, it is necessary to have the cursor positioned at the character wanted and another call made to return the character (lines 286-290). (During the execution of this function, you will see the cursor winging its way across the screen.)
5. The characters, from this point in the program on, are handled in the same way as with FPSC.COM (lines 292-347).
6. When all of the lines have been written, the file handle is closed (lines 349-355), the cursor is returned to its original position (lines 357-360), and the machine's state is restored before the program returns to MSDOS (lines 189-203 and lines 151-158).

Listing 3

```

1 page      ,132
2 title     New Version of Print Screen Utility

```

```

3
4 ; pcfpsc.asm
5 ;
6 ; PCFPSC is a print screen utility. It intercepts
7 ; INT 5 from the BIOS (generated when <shift PSC>
8 ; is struck), and prints the contents of the screen
9 ; onto the PRN device or to a file.
10 ;
11 ; This program is an extensively rewritten version of
12 ; the original one supplied by Heath/Zenith for the
13 ; Z-100. The program is for my TIB10 dot matrix printer,
14 ; which has no graphic mode or graphic characters: all
15 ; graphics and special characters are changed to spaces.
16 ;
17 ; The program now allows screens to be printed to a file
18 ; There are two modes for files: create a new file each
19 ; time and append the screen data to the end of file.
20 ; Note that the screen images are ended with a form feed
21 ;
22 ; NOTE: This program uses Int 28h to figure out if it
23 ; is all right ***** to do file I/O. Because this
24 ; interrupt is undocumented, there are NO guarantees,
25 ; expressed or implied, about the use of this program.
26 ; YOU HAVE BEEN WARNED!
27 ;
28 ;
29 ; Written by:      William B. Rothman,
30 ;                Heath Cliff Software
31 ;
32 ; Date Written:   07-Sep-86
33 ; Last Updated:  21-Oct-86
34 ;
35 ;
36 ;
37 ;
38 ;
39 ;
40 ;
41 ;
42 ;
43 ;
44 ;
45 ;
46 ;
47 ;
48 ;
49 ;
50 ;
51 ;
52 ;
53 ;
54 ;
55 ;
56 ;
57 ;
58 ;
59 ;
60 ;
61 ;
62 ;
63 ;
64 ;
65 ;
66 ;
67 ;
68 ;
69 ;
70 ;
71 ;
72 ;
73 ;

```



```

74
75 cursor dw 0 ; Original cursor location.
76
77 cols dw 0 ; Number of columns on the
screen.
78
79 dispt dw 0 ; Displacement & segment for
80 segmt dw 0 ; int 28 return address.
81
82 int_ss dw 0 ; Save area for the old stack's
83 int_sp dw 0 ; segment and offset registers.
84
85 line db 83 dup (0) ; Screen's line buffer.
86
87 eos db 0 ; flag: 0= not end of screen,
l= yes.
88
89 append db 0 ; flag: 0= not append mode,
l= yes.
90
91 err_msg dw 0 ; Offset to error message.
92 err_l dw 0 ; Length of message.
93
94 ; Error messages:
95
96 msg0 db CR,LF,'Pcfpsc: error in '
97 MSG0L equ $ - msg0
98
99 msg1 db 'seeking to end of the file!',CR,LF
100 MSG1L equ $ - msg1
101
102 msg2 db 'creating the file!',CR,LF
103 MSG2L equ $ - msg2
104
105 msg3 db 'opening the printer!',CR,LF
106 MSG3L equ $ - msg3
107
108 msg4 db 'writing to the file!',CR,LF
109 MSG4L equ $ - msg4
110
111 dw 64 dup (0) ; Internal stack.
112 int_stack label near
113 page
114 ;
115 ; Int 5 handler:
116 ;
117 ; This function merely sets a flag when the Shift-PSC
118 ; key combination is pressed and then returns. It has
119 ; to do this since it is probably not safe to do file
I/O at this time.
120 ;
121
122 int_5:
123 mov cs:flag,l ; Set the print-screen-request
flag on.
124 iret ; Back to the BIOS.
125
126 ;
127 ; Int 28 handler:
128 ;
129 ; This routine is called when MSDOS wants input to
130 ; the terminal and it is all right to do disk I/O.
At least that is the theory.
131 ;
132 ; In any event, this routine checks the flag which is
133 ; set when the print screen "hot key" is pressed. If
134 ; the flag is set, it does the print screen function.
135 ; If the print screen function is in progress already,
or the flag is not set, it simply returns.
136 ;
137
138 int_28:
139 cmp cs:flag,0 ; Print screen function wanted?
140 je inl ; If not, go.
141
142 push es ; See if we are already in this
143 push ax ; function.
144 mov ax,xxdata ; Set ES to PC flag byte segment
145
146 mov es,ax ; address. Flag is at offset 0.
147 cmp es:status,l ; PSC in progress?
148 je in0 ; If yes, go.
149
150 mov es:status,l ; Set flag to PSC in progress.
151 call psc_fcn ; Do it.
152 mov cs:flag,0 ; Turn off flag now.
153 in0:
154 pop ax ; Restore AX & ES.
155 pop es
156
157 inl:
158 jmp dword ptr cs:dispt ; Return back to the
system now.
159 page
160 ;
161 ; This is the print screen function which is called
162 ; if wanted during an int 28.
163 ;
164
165 psc_fcn:
166 push ds ; Save DS.
167
168 mov ax,cs
169 mov ds,ax ; Set up my data segment.
170
171 cli ; Disable interrupts.
172
173 mov int_sp,sp ; Save the stack pointer and
segment.
174 mov int_ss,ss
175
176 mov ss,ax ; Use my larger, local stack.
177 mov sp,offset int_stack
178
179 sti ; Now allow interrupts.
180
181 push bx ; Save everything.
182 push cx
183 push dx
184 push si
185 push di
186
187 call int_scrn ; Process the screen.
188
189 pop di ; Restore the world.
190 pop si
191 pop dx
192 pop cx
193 pop bx
194
195 cli ; Turn interrupts off for
a bit.
196
197 mov sp,int_sp ; Restore the old stack.
198 mov ax,int_ss
199 mov ss,ax
200
201 pop ds ; Restore DS.
202
203 ret ; Return.
204 page
205 ;
206 ; Gather the screen's characters and display them on
207 ; a printer or write them to a file.
208 ;
209
210 int_scrn:
211 cld ; Clear direction indicator.
212 mov eos,0 ; Initialize end of screen
flag to NO
213 mov handle,0 ; and the file handle, as well.
214
215 cmp io_dir,0 ; Is the I/O to the printer?
216 je int01 ; If yes, go.
217
218 cmp append,0 ; Want the append mode?

```

```

219     je    int00      ; If not, go.
220
221     mov   ax,3d01h    ; Open the file for write.
222     mov   dx,offset fname ; Use the specified file's
                        ; name.
223     int   21h        ; Open the file. If it doesn't
224     jc    int00      ; exist, then go create it.
225
226     mov   handle,ax   ; Else save the handle.
227     mov   err_msg,offset msg1 ; Set error message
                        ; address.
228     mov   err_1,MSG1L ; Also set length.
229     mov   bx,ax       ; BX= file's handle.
230     mov   ax,4202h    ; Move to the end of the file.
231     xor   cx,cx       ; CX:DX= 0... want the end
                        ; of the file.
232     xor   dx,dx
233     int   21h
234     jc    int02      ; Go if trouble.
235
236     jmp   short int04 ; Else continue.
237
238 int00: ; Create the file:
239     mov   err_msg,offset msg2 ; Set error message
                        ; address.
240     mov   err_1,MSG2L ; Also set length.
241     mov   ah,3ch      ; Create the file (or kill
                        ; the old one).
242     xor   cx,cx       ; Use no special attributes.
243     mov   dx,offset fname ; Use the specified file's
                        ; name.
244     int   21h
245     jnc   int03      ; If ok, go.
246
247     jmp   short int02 ; If trouble, go.
248
249 int01: ; Use the printer:
250     mov   err_msg,offset msg3 ; Set error message
                        ; address.
251     mov   err_1,MSG3L ; Also set length.
252     mov   ax,3d01h    ; Open printer for write.
253     mov   dx,offset prn_nm ; DS:DX is the address of
                        ; the printer
                        ; name.
254     int   21h
255     jnc   int03      ; If ok, continue.
256
257 int02:
258     jmp   int14      ; If trouble, go.
259
260 int03:
261     mov   handle,ax   ; Else save the handle.
262
263 int04:
264     mov   err_msg,offset msg4 ; Set error message
                        ; address.
265     mov   err_1,MSG4L ; Also set length.
266
267     mov   ah,15       ; Return current screen mode
268     int   10h         ; in ah= mode, al= cols/line,
                        ; bh= page #.
269     dec   ah
270     mov   byte ptr cols,ah
271
272     mov   ah,3        ; Return current cursor location
273     int   10h         ; for page # (in BH), in DX.
274
275     mov   cursor,dx   ; Save it.
276
277     xor   dh,dh       ; Start with row 0.
278
279 int05: ; Do a screen line:
280     mov   di,byte ptr cols ; Start with (x,end of
                        ; line) on screen.
281     mov   di,cols     ; Offset to the end of the line.
282
283     xor   si,si       ; Show no non-spaces found yet.
284
285 int06:
286     mov   ah,2        ; Set cursor to row,col in DX.
287     int   10h         ; Page # in BH.
288
289     mov   ah,8        ; Get character at that location.
290     int   10h         ; which is returned in AL.
                        ; BH= page #.
291
292     cmp   io_dir,1    ; If IO to a file, show all.
293     je    int08
294
295     cmp   al,' '      ; Make sure only printable
296     jl    int07       ; characters are retained.
297
298     cmp   al,'~'
299     jl    int08
300
301 int07:
302     mov   al,' '      ; Not printable: make it a
                        ; space.
303
304 int08:
305     cmp   al,' '      ; Is the character a space?
306     jne   int09      ; If not, go.
307
308     and   si,si       ; Any non-space found yet?
309     jz    int10      ; If no, go.
310
311 int09:
312     mov   line[di],al ; Save the character.
313
314     and   si,si       ; Non-space found yet?
315     jnz   int10      ; If yes, just go.
316
317     lea   si,[di+1]   ; Else keep the offset to
                        ; the non-space.
318
319 int10:
320     dec   di          ; Decrement the offset to
                        ; the line.
321     dec   dl          ; Go to next character.
322     jge   int06      ; Repeat this sequence for
                        ; all chars.
323
324 ; Reached the start of character line:
325
326     mov   word ptr line[si],CRLF ; Put in a CR-LF.
327
328     lea   cx,[si+2]   ; Set up the counter.
329     inc   dh          ; Up index to the next line
330     cmp   dh,25       ; and then see if we are at
                        ; the end of the screen.
331     jne   int11      ; If not, go.
332
333     mov   byte ptr line[si+2],FF ; Else store form
                        ; feed for end of
                        ; page.
334     inc   cx          ; Up length to print.
335     mov   eos,1       ; Turn on the end-of-screen
                        ; flag.
336
337 int11: ; Write out the line:
338     push  dx          ; Save the current cursor
                        ; position.
339     mov   dx,offset line ; Get address of first
                        ; character.
340     mov   ah,40h      ; Do the write: CX= length,
                        ; DS:DX=
341     mov   bx,handle   ; buffer, BX= handle, AH= op
                        ; code.
342     int   21h
343     pop   dx          ; Recall the current cursor
                        ; position.
344     jc    int14      ; Go if trouble.
345
346     cmp   eos,1       ; End of screen?
347     jne   int05      ; If not, go back.
348
349 int12:

```

```

350     mov     bx,handle    ; See if there is a handle
                        ; to close.
351     and     bx,bx
352     jz      int13      ; If not, go.
353
354     mov     ah,3eh      ; Else close whatever it is.
355     int     21h
356
357 int13:
358     mov     ah,2        ; Restore the original cursor
                        ; location.
359     mov     dx,cursor
360     int     10h
361     mov     es:status,0 ; Show all ok.
362     ret
363
364 int14:
365     call    int15      ; Error found:
                        ; Display the appropriate
                        ; error message.
366     mov     es:status,-1 ; Show error.
367     jmp     short int12 ; Go close files, etc.
368
369 int15:
370     mov     ah,40h      ; Error message display routine:
                        ; Write out the error message
                        ; beginning.
371     mov     cx,MSG0L    ; CX= length.
372     mov     dx,offset msg0 ; DS:DX= error message
                        ; address.
373     mov     bx,STDERR   ; BX= handle.
374     int     21h
375
376     mov     ah,40h      ; Write out the error message
                        ; text.
377     mov     cx,err_1    ; CX= length.
378     mov     dx,err_msg  ; DS:DX= error message address.
379     mov     bx,STDERR   ; BX= handle.
380     int     21h
381     ret
382
383 int_end label near    ; This is the end of
                        ; resident code.
384
385 ;
386 ; This routine loads in the PCFPSC routine if it is
387 ; not already loaded into memory. It also has the
388 ; ability to update the in-memory copy of the program,
389 ; if there is one, with new parameters. The routine can
390 ; also provide a help screen.
391 ;
392 ; Note that the program makes an effort to verify that,
393 ; if it finds that something pointed to by Int #5 is
394 ; loaded into memory, it is a copy of this program.
395 ; This may not be foolproof. [ Another warning. ]
396 ;
397 msg5 db CR,LF
398     db 'pcfpssc.com',TAB,'Version 1.03 by Heath
399     db CR,LF,LF
400     db 'This TSR program prints the contents
401     db CR,LF
402     db 'when Shift-PSC is pressed. It has the
403     db CR,LF
404     db 'either to a file or to the PRN device.'
405     db CR,LF,LF
406     db 'Options:'
407     db CR,LF,TAB
408     db 'pcfpssc'
409     db TAB,TAB
410     db 'Prints the screen to the PRN device.'
411     db CR,LF,TAB
412     db 'pcfpssc fn'
413     db TAB
414     db 'Prints the screen to the file fn.'
415     db CR,LF,TAB
416     db 'pcfpssc '
417 msg5s0 db 0
418     db 'a fn'
419     db TAB
420     db 'Prints the screen to the file fn,'
421     db CR,LF,TAB,TAB,TAB
422     db 'appending each screen to the end of the
423     db CR,LF,TAB
424     db 'pcfpssc ?'
425     db TAB
426     db 'Displays this message.'
427     db CR,LF,TAB
428     db ' '
429 msg5s1 db 0
430     db 'h'
431 MSG5L equ $ - msg5
432
433 msg6 db 'invalid switch specified!',CR,LF
434 MSG6L equ $ - msg6
435
436 switch db 0 ; MSDOS switch character.
437
438 start:
439     mov     sp,offset int_stack ; Set the offset to
                        ; my local stack.
440
441     mov     ax,3700h    ; Get the switch character.
442     int     21h
443     mov     switch,d1  ; Save it.
444     mov     msg5s0,d1
445     mov     msg5s1,d1
446
447     mov     ah,35h      ; See if this program is
                        ; already loaded.
448     mov     al,5        ; This function returns
                        ; vector in ES:BX.
449     int     21h
450
451     xor     dx,dx      ; Assume program is already
                        ; resident.
452
453     mov     si,offset prn_nm ; Make sure that we
454     mov     di,si      ; actually have this program
                        ; in memory.
455     mov     cx,PRN_L   ; DS:SI ; ES:DI for CX bytes:
456     repz   cmpsb      ; Do the compare.
457     je     st00        ; If ok, go.
458
459     mov     ax,cs      ; Else reset the ES register
460     mov     es,ax      ; to this copy of the program
461     inc     dx         ; and set a flag to have this
                        ; copy retained in memory.
462
463 st00:
464     mov     es:append,0 ; Turn off append mode.
465     mov     si,80h     ; Set SI to offset of the
                        ; command line.
466     xor     ch,ch      ; Set CX to the length of
467     mov     cl,[si]    ; the command line.
468     and     cl,cl      ; Is it zero?
469     jz     st06        ; If yes, go.
470
471 st01:
472     inc     si         ; Up the pointer.
473     cmp     byte ptr [si], ' ' ; Is it a space?
474     je     st02        ; If yes, skip it.
475
476     cmp     byte ptr [si],TAB ; Is it a tab?
477     jne     st03        ; If not, go.
478
479 st02:
480     ; Skip over space or tab:
481     dec     cl         ; Decrement length so far.
482     jnz     st01        ; If not zero, keep on looking.
483     jmp     short st06 ; Else go.
484
485 st03:
486     and     cl,cl      ; Got anything?

```

```

487     jz     st06      ; If not, just go.
488
489     mov    al,byte ptr [si] ; Get the character.
490     cmp    al,'?'     ; Help wanted?
491     je     st08      ; If yes, go.
492
493     cmp    al,switch  ; Is it the switch character?
494     jne    st05      ; If not, go.
495
496     inc    si        ; Up the pointer. Get next
                        ; character.
497     lodsb         ; al= ds:[si++]. Up the ptr
                        ; again.
498
499     cmp    al,'h'     ; Help wanted?
500     je     st08      ; If yes, go.
501
502     cmp    al,'H'     ; Help wanted?
503     je     st08      ; If yes, go.
504
505     cmp    al,'a'     ; Append mode wanted?
506     je     st04      ; If yes, go.
507
508     cmp    al,'A'     ; Append mode wanted?
509     je     st04      ; If yes, go.
510
511     mov    err_msg,offset msg6 ; Set error message
                        ; address.
512     mov    err_l,MSG6L ; Also set length.
513     call  int15      ; Display error message.
514     jmp    short st09 ; Go now.
515
516 st04:
517     mov    es:append,1 ; Set flag to yes.
518     sub    cl,3      ; Decrement length accordingly.
519     jg     st01      ; If more left, go back for it.
520
521     jmp    short st06 ; Else just go clear I/O
                        ; dir flag.
522
523 st05:
524     mov    di,offset fname ; DI= address of the
                        ; filename.
525     rep    movsb     ; es:[di++] = ds:[si++],
                        ; while cx-- > 0.
526     mov    byte ptr es:[di],0 ; End string with 0.
527     mov    es:io_dir,1 ; Indicate that I/O will be
                        ; to a file.
528     jmp    short st07 ; Go.
529
530 st06:
531     mov    es:io_dir,0 ; Indicate the I/O is to
                        ; the printer.
532
533 st07:
534     and    dx,dx     ; Now see if loading this copy
535     jz     st09      ; of the program. If not,
                        ; return to MSDOS.
536
537     jmp    short st10 ; Else go install this program.
538
539 st08:
540     mov    ah,40h    ; Help function:
541     mov    cx,MSG5L  ; Write out help text.
542     mov    dx,offset msg5 ; DS:DX= text address.
543     mov    bx,STDOUT ; BX= handle.
544     int    21h
545
546 st09:
547     mov    ax,4c00h  ; Return to MSDOS now.
548     int    21h
549
550 st10:
551     mov    ah,35h    ; Install this copy of the
                        ; program:
552     mov    al,28h    ; Function code type.
553     int    21h      ; Get interrupt vector #28.
554     ; Returned in ES:BX.

```

```

555     mov    dispt,bx  ; Save the ES:BX vector.
556     mov    segmt,es
557
558     mov    ah,25h    ; Function code type.
559     mov    al,28h    ; Set interrupt vector #28.
560     mov    dx,offset int_28 ; Address in DS:DX.
561     int    21h
562
563     mov    ah,25h    ; Function code type int.
564     mov    al,5      ; Set interrupt #5.
565     mov    dx,offset int_5 ; Put offset into DX;
566     int    21h      ; address is in DS:DX.
567
568     mov    ah,31h    ; Keep this process.
569     mov    al,0      ; Return zero as completion
                        ; code.
570     mov    dx,offset int_end ; Last byte of this
                        ; program.
571     add    dx,15     ; Round up to next paragraph
                        ; boundary.
572     shr    dx,1      ; Make into "paragraphs" by
573     shr    dx,1      ; dividing the ending offset
574     shr    dx,1      ; by 16 (16 bytes/paragraph).
575     shr    dx,1
576     int    21h      ; Keep the process.
577
578 code  ends
579     end    begin

```

Assembly Instructions

All of the programs are assembled in the same way, per the following batch file:

```

masm %1;
if errorlevel 1 goto done
link %1;
if errorlevel 1 goto done
exe2bin %1 %1.com
if errorlevel 1 goto done
del %1.obj
del %1.exe
:done

```

While the program for the PC is self-contained, it is necessary to have the following ".ASM" files from the Programmer's Utility Pack (Version 2 or 3) for the two Z-100 programs:

```

For PSC.ASM:  defmtr.asm
              defbios.asm
              defchr.asm

```

```

For FPSC.ASM: defmtr.asm

```

You may have to change the drive and path designation in my source files to be able to reach the include files on your system. If you have any errors reported (other than the usual "NO STACK SEGMENT" annoyance, standard with COM files), then do not try to run the resulting program! All three programs will all work with Versions 2 or 3 of MSDOS.

Warning

Please Note: The use of Int 28h is entirely at your own risk! This function is not documented by Microsoft. All of its effects and side-effects are not known. The interrupt could break in an upgrade to our current release of MSDOS or in the next version of MSDOS. While we Z-100 and PC users with lowly 8088s (and equivalents) might never see MSDOS Version 5 (Version 4, according to rumor, is not going to be released in this country), since it is supposed to be for the 80286 and 80386 based machines, it is important to keep in mind that Microsoft is actively hostile to TSR routines (who will buy an expensive operating system upgrade, or better yet, XENIX, if everything you want to do can be done

Continued on Page 83

C__Power

Part 2

John P. Lewis

*6 Sexton Cove Road
Key Largo, FL 33037*

This article will expand on the theme of "CPOWER". It will continue the building of functions in "C" with the eventual result being a data storage program which will lead the user through it's routines using menu selections. The resulting program will feature random access storage with the ability to edit any field of any record, as well as record retrieval using substrings (part of a record) and sorting on one or more fields.

This is a rather ambitious project and will obviously require several issues of this magazine for it's completion. This issue will serve to present the skeleton of the resulting program and give the reader an idea where we are headed. Once again, this article is directed at those Huggies who are using an H-89 or H/Z-100 series computer under CP/M and Software Toolworks "C/80". Other configurations will need some changes in the coding before the program will be usable.

I'm going to format this program as the old familiar "Mail List". With appropriate changes, the reader will be able to reformat it to whatever his needs are for a random access program. I have used this data storage program, with some changes, for reloading data (rifle, pistol, shotgun), a card catalog (magazine library), inventory records and the aforementioned mailing list/directory.

Those of you who just joined us will need to get a copy of "REMARK" containing the article "CPOWER — Part 1", since we will build this program step-by-step using, in some cases, functions described in a previous issue.

As an added bonus to those who follow this series to fruition, I have obtained permission from Howard W. Sams & Co. to include a program from their book, "Soul of CP/M", which I made some changes to and altered slightly for use under "C". The original program was written in assembly language, but as the reader knows by now, the "C" compiler "likes" assembly language. In fact, the compiler uses the "C" code to generate it's own assembly language version before final compilation into machine language

which is actually accomplished by an "assembler", a part of the package you get when buying "C/80". The program which we will include is called "HEXDUMP" by it's authors, and allows the user to access any address in memory and read the instructions at that address. The instructions will be in hex and an ASCII representation will also be displayed (my contribution to this program). You are probably saying, "no big deal, DDT will do that". The big deal is that you can do this without exiting your program and it is much easier to use than DDT. This program will take the form of another function and will be called using a single keystroke.

Are you sold yet? If so, read on, things are just beginning to get interesting.

Here is the skeleton that we will build on:

```

#include "printf.c"
#include "funclib.c" /* routines cls, locate, fetchc and
    gofor from "CPOWER" */
#define DEL 127 /* this char used for an "end of records"
    delimiter */

/* --- by John P. Lewis --- */
main ()
{
    unsigned fd, offset;
    char men[2], compny[32], name[32], street[32], city[32];
    /* above variables defined for use in expanded program */
    do
    {
        cls ();putchar(27);putchar(70);locate (4,5); /* enter
            graphics mode */
        printf("pppppppppppppppppppppppppppppppppppppppp");
        printf("pppppppppppppppppppppppppppppppppppppppp");
        putchar(27);putchar(71); /* exit graphics mode */
        printf("\n\n\t          Please ....");
        printf("\n\n\t          1. Search for record.");
        printf("\n\n\t          2. Review existing record
            names.");
        printf("\n\n\t          3. Create new record(s).");
        printf("\n\n\t          4. Print entire mailing list

```

```

    (to printer).");
printf("\n\t          5. Sort list (on specified
field).");
printf("\n\t          6. Exit to operating system.");
printf("\n\n\t      Enter the number corresponding to
your choice ");
gofor(men,1);
switch (men[0])
{
case '1':
    cls();locate (4,1);
    printf("You have selected the 'Search' option which
has not\n");
    printf("yet been implemented. Press RETURN for menu ");
    gofor(men,1);
    break;
case '2':
    cls ();locate (4,1);
    printf("You have selected the 'Review' option which
has not\n");
    printf("yet been implemented. Press RETURN for menu ");
    gofor(men,1);
    break;
case '3':
    cls();locate(4,1);
    printf("You have selected the 'Create' option which
has not\n");
    printf("yet been implemented. Press RETURN for menu ");
    gofor(men,1);
    break;
case '4':
    cls();locate(4,1);
    printf("You have selected the 'Print' option which
has not\n");
    printf("yet been implemented. Press RETURN for menu ");
    gofor(men,1);
    break;
case '5':
    cls();locate(4,1);
    printf("You have selected the 'Sort' option which has
not yet\n");
    printf("been implemented. Press RETURN for menu ");
    gofor(men,1);
    break;
} /* end of switch loop */
} /* end of menu do - while loop */
while ( men[0] != '6');
} /* end of main */

```

Notice the `#include "funclib.c"` near the top of the listing. This is our function library from the article "CPOWER". Your filename is probably different so should be changed to correspond since we will be referring to and adding functions to this file in future issues. This program does nothing spectacular, but it is the vehicle for our random access program which we are going to build, one routine at a time. Be sure that this skeleton works as it should, since we are about to replace one of the "switch-case" routines with code which will allow us to write "random" records.

The best thing to do at this point, if you have not already done so, is to spend some time reading the documentation that is furnished with "C/80". Concentrate on the section dealing with opening a file, and reading and writing to a file. A review of your CP/M documentation on disk files might be of some help here. Are you back yet? Thoroughly confused? Don't feel bad, we will sort all of this out and you will soon be an expert on CP/M file handling.

The first thing the computer must do when dealing with a file is determine whether or not the file already exists, since the program must tell the computer to read, write or update the file in question. If we ignore the fact that the file already exists and simply tell the computer to write to the file, it will certainly do so, by destroying the existing file and creating a new one! This is hardly conducive to records integrity. We must, therefore, determine not only if the file

already exists, but its size since we may want to tack on some records to its end. The best way to accomplish this is to tell the computer to read "FILENAME". We do this with a code fragment that resembles this:

```
fd=fopen("data","r")
```

Where "fd" is the file descriptor and the computer will assign a number (non-zero if file exists, zero if file is not found). We then test "fd" for its value and take appropriate action. Please notice the rest of the code fragment before we leave to do other things. We opened the file (if a file named "data" exists) and we told the computer to "read" this file. Now the program will test "fd" for zero. We will use a code fragment like this:

```
if ( fd == 0 )
{
fd = fopen("data","w");flag = 1;
}

```

We are going to have a file named "data" when we get past this line of code whether one existed before or not. If we need to create one, we put the computer in the "write" mode. We also give a value of 1 to "flag".

Now we have our file and we know whether it is a new one (if flag is equal to one) or if it is to be appended. How can we tell where the end of the file is if we intend to append it? I was hoping you would ask that! When we add a new record to this file we must also add a marker to the file so we can come back and find the last record. The best way I have found to do this is to write a unique character to the record after the one we are creating. This unique character is then written over and a new one written each time we append the file.

We tell the computer to search the file for this unique character and when it is found we have also found the next vacant record. We tell the computer to search the file using an offset. The offset will be the number of records past the start of the file times the number of characters in a record. To elaborate: if each record is 64 bytes in length and we wish to read the third record, we would tell the computer to read a record using an offset of 128. Remember, we start with zero so byte number 128 would be the first byte of the third record. Each time we wish to look at a succeeding record, we increment the offset by 64. Be sure you understand this concept before proceeding because the rest of this series will use this technique to retrieve and read (or edit) or append disk files.

OK hang on, we are going to examine the method used to write a record so the format will fit our parameters. Then we will be able to identify the last record, or any field within a record. Assuming a record length of 128 characters, and assuming we have already opened the file in question for writing, the coding should look something like this:

```
fprintf(fd,"%-32s%-32s%-32s%-32s%c",compny,name,street,
city,DEL);
```

Here we are telling the computer to allocate 32 bytes each to four individual fields, one byte to the 129th byte of this record (actually the first byte of the next record) and write this record to the disk with the information it has been given. Here I'm assuming that data has already been assigned to company name, street and city through previous code. I'm also assuming that this is either the initial record of this file or that we are appending a new record. This technique will not work if we are attempting to edit an existing record, since we would write over the first character of the succeeding record with a delete (127) character (delete defined earlier at top of program [`#define DEL 127`]). This delete character will be used in identifying the last record in a file. Also please note the

format of the `fprintf` statement. Here we are telling the computer to write (`fprintf`) to a file descriptor (`fd`) using a field that is 32 characters in length and is left justified (`%-32s`).

We just covered a lot of ground and omitted some rather important concepts which was done in the interests of clarity. We are going to do this random record job one step at a time so don't become impatient.

One aspect of random records which needs more clarification at this point is that of file update. When a file is opened in the write mode, the computer will first search the disk for a file name which matches the one it is looking for. If it finds a match, it will erase that file and then create a new one with the name given. This means that we can only open a file in the write mode when we wish to create a file. When we are updating a file, we must use the update mode. This is the mode used when editing a file or appending (adding on). We can tell the computer that we wish to update by opening the file with the following code:

```
fd=fopen("filename","u")
```

Now you can see why we gave a value of 1 to "flag" in the code fragment above.

The "C" language is a bit tedious to use when dealing with random records, but that is one reason why "C" is so powerful. You are not confined by the structure of the language. This also means that you, the programmer, will have to take care of some of the tasks that are done for you by a language such as BASIC. We have already touched on one of those tasks and we are about to enter an area that involves a great deal of "housekeeping" by the program. What this really means is that YOU are going to have to provide a way for the program to keep track of where on the disk the data is located. We already mentioned "offset", the next new term will be "seek". This will be a familiar term to those of you who did their homework, but strange to some of you. What we are going to accomplish with these two commands is the positioning of the read-write head of the disk drive to read or write in the exact location we wish. We will soon be able to identify one field within thousands of bytes of data, and then manipulate the data and change any field (if we wish) within that file at will. That, of course, is what random records are all about.

The following code will cause the computer to insert (write over) the specified field within an existing record in a file named "data":

```
fd=fopen("data","u");offset=32;
seek(fd,offset,0);fprintf(fd,"%-32s",name);
seek(fd,0,2);fclose(fd);
```

With the above code we could change the name field within an existing record, advance to the end of the file, and then close the file assuming that we had already defined "name" with a "gofor" or "scanf" statement, and that we were dealing with a record where the name field started with the 32nd byte in the record (as in the 128 byte record described earlier). Notice the code following the `fprintf` statement. We must, if we wish to avoid unpredictable results, seek to the end of the file being altered and then close it unless we are going to alter more records within the same file. The command:

```
seek(fd,0,2)
```

will cause the read-write head to seek to the end of the file designated "fd".

Now is a good time to get your C/80 manual and review some of the material we have covered. Then, take a deep breath and begin

altering the program displayed in this article. Insert your own code into the "case" corresponding to "Create a new record". Hint: Code this routine so the computer checks for an existing file, searches for the last record, and appends a record if file exists or creates a new file if "filename" is not present on disk. The next article on "CPOWER" will include code to accomplish this in the manner that I would do it but your way might be better. Give it a try anyway since that is the best way to learn.

You are on your way to becoming proficient in the technique of using random records in your programming efforts. Arm yourself with a good measure of patience and fortitude and try to keep "Murphy" out of your work area (he lives at my house so that should be no problem)! When you have finished this series on "CPOWER" you should have a good comprehension on the subject of random records.

One of the most helpful books I have been able to find on the subject of "C" is: "PROGRAMMING IN C" by Stephen G. Kochan (a HAYDEN publication). This refers to "C" in a "UNIX" environment but the treatment is quite similar to that for CP/M. Another book that I have found very helpful in the CP/M environment is: "Soul of CP/M", by Mitchell Waite and Robert Lafore. This is a Howard W. Sams & Co. publication (previously mentioned). Good luck, "C" you soon.

```
cls ()
{
  putchar(27);putchar(69);/* escape, "E" sequence for H89's */
}
```

```
locate (row,col)
int row, col;
{
  row+=31; col+=31;
  putchar(27);putchar(89); /* escape, "Y" sequence for
  direct */
  putchar(row);putchar(col); /* cursor control */
}
```

```
fetchc ()
{
  #asm
  DIR:MVI E,0FFH /* "direct" call to CP/M Dos */
  MVI C,006H/* for I/O (input) */
  CALL 5
  CPI 0
  JZ DIR
  #endasm
}
```

```
gofor (c,s)
int s;
char c[];
{
  int i;
  for ( i=0 ; i <= s && c[i-1] != 13 ; ++i )
  {
    c[i]=fetchc();/* c[i]=toupper(c[i]) */ putchar(c[i]);
    if ( c[i]==8 ) /* this subroutine "calls" */
    { /* fetchc () for character I/O */
      i-=2;putchar(32);putchar(8);
    }
  }
  if ( i-1 == s ) /* a return or s+1 characters will */
  c[i-1]='\0';/* cause this routine to return */
  if ( c[i-1]==13 )/* and will replace the return with a
  null */
  c[i-1]='\0';/* ('\0'). This is a "C" requirement */
}
```



FBE Products

For the H/Z-150, 160 Series

MegaRAM-150 — Modification kit allows memory board to be filled with 256K RAM chips (1.2 MByte). No soldering. Supplied with RAM disk software. **\$49.95**

ZP640 PLUS — Replacement PAL for standard memory board allows up to 2 banks of 256K and 2 or 3 banks of RAM chips to be installed for 640K or 704K maximum memory. **\$24.95**

COM3 — Replacement PAL allows installation of three serial ports (one an internal modem). Supplied with printer driver software for 3rd port. **\$39.95**

FBE Smartwatch

Calendar/Clock using Dallas Semiconductor's DS1216E SmartWatch module. Works with H/Z-110/120, 138/148, 150/158. Package includes SmartWatch with our software and documentation. Spacer kit (\$2) required for Z-100. **\$44.95**

For the H/Z-100 Series

ZMF100a — Modification package allows installation of 256K RAM chips in older Z-100 without soldering. Works only with old-style motherboard. **\$65**

ZRAM-205 — Kit allows 256K RAM chips to be put on Z-205 memory board to make 256K memory plus 768K RAM disk. Requires soldering. PAL (\$8) required for new motherboard. **\$49**

For the H/Z-89, 90 Series

SPOOLDISK 89 — 128K byte electronic disk and printer interface/spooler card. **\$195**

H89PIP — Dual port parallel interface card. Use as printer interface. Driver software included. **\$50 Cable \$24**

SLOT4 — Extender card adds 4th I/O expansion slot to right side bus. **\$47.50**

FBE

FBE Research Company, Inc.

P.O. Box 68234, Seattle, WA 98168

(206) 246-9815, M-F 9-5

UPS/APO/FPO Shipping Included.
VISA or MasterCard Accepted.



TO ORDER
CALL 714-540-1174
or WRITE

M/C, VISA, CHECK or MONEY ORDER
CALIF. RESIDENTS ADD 6% Sales Tax

SHIPPING-ADD
UPS Ground \$1.50
2nd Day Air \$4.00
Next Day Air \$12.00

Zenith is a registered trademark of Zenith Data Systems Corporation.

ZP-150 RAM EXPANSION 32K as low as \$45

The ZP-150 is a great laptop. It lacks only one thing—sufficient on-line memory. Our 32K low-power CMOS modules plug into existing sockets in the computer to provide up to 416K of on-line, non-volatile memory. Upgrading your machine with from one to twelve modules takes a matter of minutes.

We are in our third year of providing quality memory upgrades similar to the 32K to thousands of satisfied portable computer users. Our memory modules are 100% factory tested and carry a one year materials and workmanship warranty. Because your satisfaction is our goal we offer a full refund 30 day return policy.

Priced at: \$59 for 1 or 2
\$49 for 3 to 9
\$45 for 10 or more

Illustrated step-by-step instructions included.

**AI AMERICAN
CRYPTRONICS INC.**

(Formerly Cryptronics, Inc.)
1580 Corporate Drive, Suite 123
Costa Mesa, California 92626
(714) 540-1174

VAX & PC Users

ZSTEMpc™-VT220 Emulator for the PC Series
High performance COLOR VT220. Double high/wide, 132 mode, smooth scrolling, downloadable fonts, user defined keys, 8-bit and full national/ multi-national modes. XMODEM and KERMIT, softkey/MACROS, DOS access. **\$150.00**

EGAmate™ — option for true 132 column mode **\$39.00**

ZSTEMpc™-VT100 Fastest, most complete emulation **\$99.00**

ZSTEMpc™-4014 Emulator for the PC Series
Use with ZSTEMpc-VT100, VT220, or stand-alone. Interactive zoom and pan. Save/recall images to/from disk. Keypad, mouse, printer, and plotter support. **\$99.00**

ZSTEM™-VT100 Emulator with XMODEM FOR THE Z-100 **\$148.95**

DECKHAND™ Utilities for the Z-100 and PC Series
MS-DOS Utilities with VAX/PDP-11 switch processing, DIR, COPY, DELETE, RENAME, TYPE with wildcards, full DATE processing, attribute processing, query, backup and more. **\$49.00**



KEA Systems Ltd.

2150 West Broadway, Suite 412
Vancouver, B.C. Canada V6K 4L9
Technical Support (604) 732-7411

Order desk (800) 663-8702 Toll Free

Telex 04-352848 VCR
VISA and Mastercard accepted.
30 day money back guarantee.
Quantity and dealer discounts available.

Trademarks
VT100, VT220, DEC, VAX — Digital Equipment Corp. ZSTEMpc
ZPAL, COMPAL, EGAmate, DECKHAND — KEA Systems Ltd.

Z-100 PC Expansion with ZPAL™

Z-151/Z-161 Users
Use 256K bit memory chips on the original memory board. Extend the memory to 640K or 704K bytes with ZPAL-2 Decoder **\$29.00**

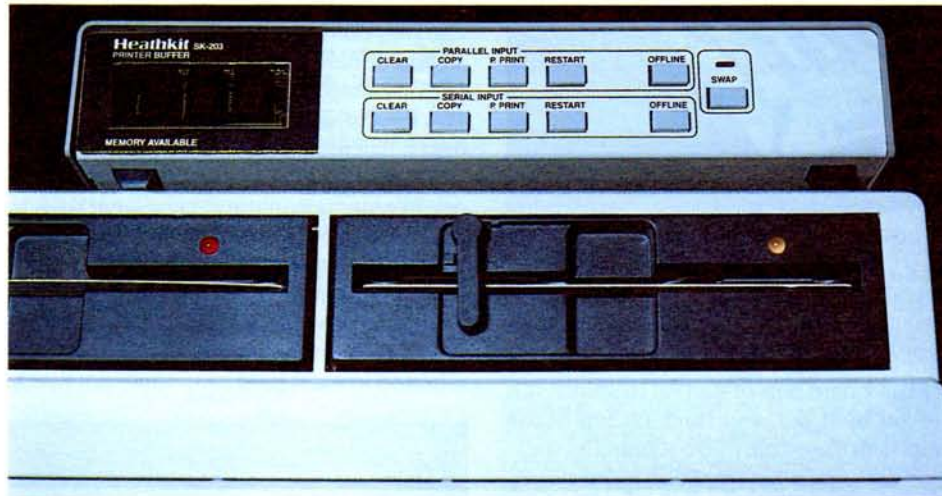
Z-158 Users
Extend your system to 768K bytes with 3 banks of 256K bit memory chips and ZPAL-158 decoder. ZPAL-158 Decoder **\$36.00**

Z-138/Z-148 Users
Extend your system to 768K bytes with 3 banks of 256K bit memory chips and ZPAL-148 decoder. ZPAL-148 Decoder **\$36.00**

Z-100 PC Expansion with COMPAL™

Z-151/Z-161 Users
Instead of disabling COM2 when using an internal modem or multifunction board, MAP it to COM3. Supported by ZSTEMpc-VT220 or the driver software included. COMPAL-3 **\$39.00**

The Jewel Of The ^FXile



Jim Buszkiewicz
HUG Managing Editor

Remember Heathkits? They were the things that, when you looked in the box all you'd see was, a manual and a mountain of loose parts. It was really hard to tell if you got what you ordered unless you snuck a peek at the wrapped and sealed front panel, or the manual just happened to have a picture of the product on the front page (many did!). You'd scramble for the hand tools, plug in the soldering iron, and curse because it wasn't hot enough to use five seconds later. Now you had to wait a couple of minutes. That was plenty of time to dump all the parts on the table in front of you, flip past all the preliminary junk in the assembly manual to the first step of construction, and still have to wait another minute for the soldering iron. You trusted Heath. In all the kits you built, never once was there a part missing, so why check them off against the parts list? Besides, if one does happen to get misplaced, you'd know it when you got to the step that called for its use. With that last minute to spare, you'd prop up the pictorial, and get a bag of potato chips and bottle of beer (or pop). Now you were ready. Remember Heathkits?

For me, those are good memories. Half of the fun was building the kit, and the other half was troubleshooting it. If any of you have recently built a computer kit from Heath, you know that those days are gone. You didn't even need a soldering iron to complete the job. But wait! There's still hope! For you kit-starved computer buffs, a ray of light shines through the gloom of Heath non-kits. In their most recent catalog, Heath has introduced the SK-203 printer buffer, a complete kit in the true Heath tradition.

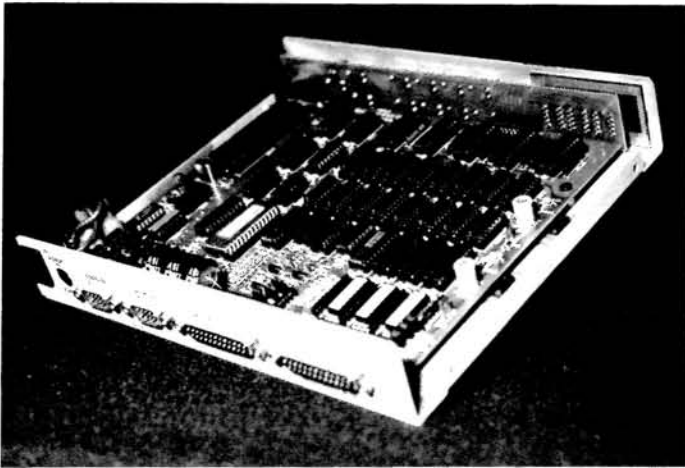
A printer buffer is like a giant storage tank. It collects and stores data that is received via its parallel and serial input ports. The buffer receives its data from a sending device, which can be a computer or modem, and stores data into its memory as fast as the sending device can send it. The buffer then sends the data to an output device, such as your printer, via the buffer's parallel or serial output ports. If you use the printer buffer with a computer, once the data is loaded into the buffer memory, you are then free to use your

computer for other purposes, while the buffer sends its stored information to the printer. The buffer can collect data simultaneously from two computers and can also send data to two printers at the same time.

Upon opening the box, I found chassis parts and circuit boards packed first. Next came the bags of loose parts, and finally the manual. This was not at all what I expected. I naturally assumed the assembly manual should come first, and probably would have if I had opened the top of the box instead of the bottom. Nevertheless, the results would still have been the same, everything was in a big pile on my work bench just as if I had opened the box from the top. My soldering iron was almost hot, but something was missing. It had been so long, I'd almost forgotten the scenario, beer and potato chips. Now I was ready. Step one, assemble the manual. What?! Yup, the first thing you have to do is assemble the three ring manual binder. With the three shades of blue striping along the bottom, the assembly manual also reflects the new Heath look.

Two circuit boards make up the entire printer buffer. The first board built is the switch circuit board. This board is made up of 11 push button switches, a single LED, and several socket pins. The style of socket pins used was something new to me and I wasn't sure exactly how they worked. These pins are pushed in from the front of the board and then soldered on the rear. The tops of the pins themselves appear to be filled with some sort of white material, which, according to the manual, is actually conductive. They are used in the same manner an IC socket is, only these pins allow the IC to rest flush against the top of the circuit board. When you install these pins, make sure they're pushed in all the way. They should be flush with the top of the circuit board, and may require an extra ounce or two of pushing to get them to seat properly.

The main circuit board is a whopper. It packs 42 ICs, 13 resistor packs, 8 or 9 dozen resistors and capacitors, and a handful of diodes, connectors, and ferrite beads. Most of the resistors, capac-

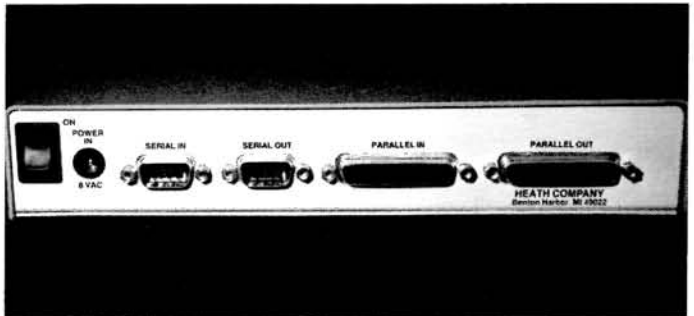


itors, and diodes come on a taped strip in the order that they're used. This, of course, saves a lot of time hunting for the individual parts. Installing all of the parts was straight forward and no problems were encountered. I would like to mention that when installing IC sockets, make sure that ALL the leads of the socket are protruding through the circuit board before starting to solder. It is very easy for one lead to get bent out, and stuck on top of the board. Also, be extra careful with the 64 pin micro-controller socket. The leads on this socket are somewhat small and more closely spaced than those on a standard socket. This board can take anywhere from four to six hours to complete, varying, of course from builder to builder.

The switch and main circuit boards are soldered together with right-angle connector pins. The board combo is then mounted in the chassis bottom using 4-40 screws. The back panel is secured to the 'D' style connectors and then to the chassis bottom. The cover is held in place with four 4-40 screws. To completely disassemble the unit for servicing, only 12 easily accessible screws need be removed. The circuit board combo and back panel can then be lifted out. The buffer is powered by an 8.5 VAC power pack that plugs directly into a standard wall outlet.

Configuring the buffer is accomplished by way of 3, eight-position DIP switches, and several small, finger movable, jumper plugs. Configuration parameters include serial input and output port settings, auto form feed, port routing, and the size of the dynamic ram ICs. Specific information on configuring the buffer was spread out among several pages in the assembly manual, as well as the Illustration Booklet and the configuration chart stuck to the inside top chassis cover. Extracting the proper settings was only slightly confusing, and made the build more interesting.

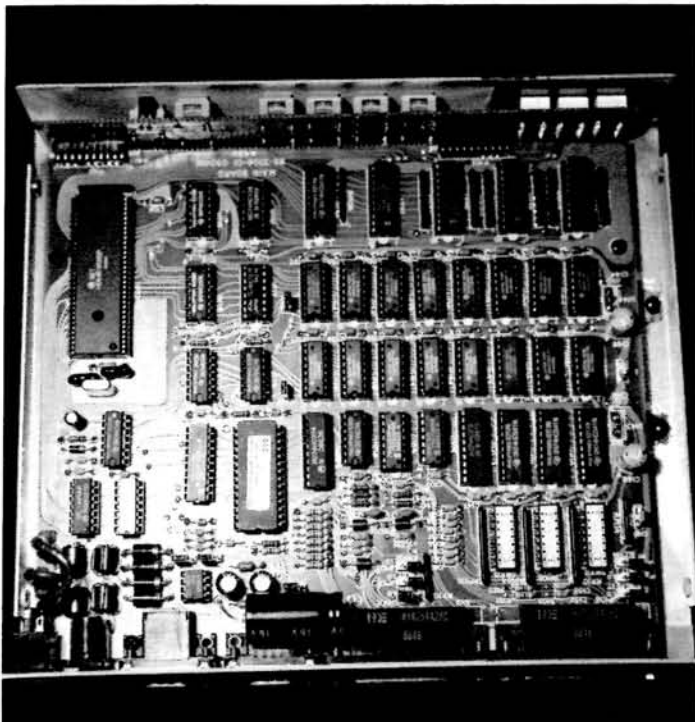
The kit is normally supplied with eight 64k ram chips. You can add an additional bank of 64k, or 256k chips. Ideally, both banks can be populated with 256k devices making the total buffer size 512k. This final value was my choice. I figured if my print file was larger than a half meg, I wouldn't be able to put up with over an hours worth of noise from my printer anyway. Unfortunately, Heath doesn't mention in the catalog what parts to purchase to increase the buffer size. The buffer will accept 4164 - 64k chips, or 41256 - 256k chips. I would suggest purchasing the higher capacity devices from anywhere you can find them the cheapest. Since the main processor (A 64180 IC) is running at a little over 6 MHz, make sure the chips you buy have an access time of 200ns or less. Eight are required in each bank, and there are two banks.



One part of the manual that wasn't skimped on, was what types of cables were needed to connect the buffer to the computer and the printer. You can buy pre-made cables from Heath, or you can construct your own. Detailed cable wiring diagrams are included in the manual not only for all Heath/Zenith computers, but also for IBM PC and AT computers, and (groan) APPLE II, II+, IIc, and IIe systems. Each cable data line is clearly labeled with the name of the line, data direction, and short description of the line's purpose. Also included are timing diagrams for both the parallel and serial data sequence. Although it may exist, I can't recall ever seeing these simple, but very useful, timing diagrams in ANY Heath computer manual! The assembly also has a troubleshooting chart that helps out with the most obvious problems that one could encounter. I'm afraid, however, if a major failure occurred, you'd have to rely on your own personal digital background. All in all, I must commend Heath manual writers for a job well done!

Perhaps not as useful to some people as it may be to others, one of the buffer's most desired features for myself is its ability to input data to its parallel port, and then send it out its serial port. As I've mentioned earlier, my printer is an old rugged H-25 serial device, and the one thing that's NOT plentiful on a PC compatible, are serial ports. IBM only defined two, COM1 and COM2. Now my serial printer can use the parallel printer port, freeing up one of the serial ports for other uses like a mouse, plotter, modem, etc. A second computer can be connected to the unused serial input port, and a second printer to the unused parallel output port. Then, by way of the front panel 'swap' button, the printers can 'change places' if you like. One situation that might occur would

Continued on Page 83



ASCII

We See It Mentioned Often But What Is It?

Kenneth Mortimer, PE
352 Green Acres Drive
Valparaiso, IN 46383

There isn't a copy of REMark or any other respectable computer journal in which the acronym ASCII does not appear. The acronym also appears in the body of many computer textbooks, but other than a usually undecipherable table in the appendix of the text there is very little explanation of the term. It is something the experts think is so obvious that we all know everything about it.

What is this AS KEY that computer people talk about so often? Where does this "CHR\$(27)" in basic programs come from? What does all of this mean to each of us who sit in front of a terminal and where does it come from?

In the early part of the 1960's the codes for the transmission of information were many in number and confusing in nature. Perhaps the most common was the BAUDOT code used in the teletype machines. While well established, it had the disadvantage in that the same code could represent either a character or a figure, depending upon whether the machine was in "letters" or "figures" mode. The code 10100 represented the letter "U" in letters mode and rang the warning bell in the "figures" mode. Since a teletypewriter had no provision for lower case type, Baudot had no provision for lower case characters. The Baudot code was not only used for the punched paper tape but also for transmission. The five digit Baudot code had a strong competitor in the eight channel code which was used in both the Freiden Flexewriter and the IBM 046 tape punch. The unit record card also had three major competitors: the 80 column by 12 row Hollerith (IBM) code, the 90 column by 6 row Univac card and the British Bull code.

It was recognized that a standard code would soon be required for the communication between the components of a system, between systems and even internally within a component of a system. On June 17, 1963 the first version of the American Standard Code for Information Interchange (hence ASCII) was adopted by the American Standards Association X3.2 Subcommittee. It was a seven bit code with some limitations and contained many codes required for telecommunications including such codes that are now anachronistic to those in the computer area, but were essential twenty years ago. If you will look at the table of ASCII

characters in your Heath/Zenith computer manual you will find such codes as "RU" (are you?) or "SYNC" (synchronous idle) which show their teletype heritage. The H/Z manuals are among the few places that both the "nonprinting" and the "printing" codes are shown with a description of the "nonprinting" codes.

It was soon recognized that the seven bit code (having upper case letters only) was severely limited so an extension of the ASCII code to 8 bits was adopted in 1967.

Since the ASCII code is organized on the basis of bit pattern, it is most easily explained in terms of the hexadecimal number system. However, we are accustomed to thinking in the decimal number system, and since the BASIC and Pascal functions that access the ASCII codes use decimal numbers, I will use both number systems but will follow the decimal number with "D" and the hexadecimal number with "H".

The Nonprinting Codes

The first 32D (0D/0H to 31D/1FH) ASCII characters are nonprinting characters. When they are used, most of them normally cause no output in the monitor. I will show you later how to test the output of the ASCII codes with a simple BASIC program. I have tried it with two combinations; A Z-89 with a Daisywriter and both a Z-148 and a Z-150 with a Panasonic KX P1092 printer. I will discuss the results as we go along. Normally, one would expect most of these characters to have no effect on the CRT screen. However, some ingenious computer designers have found that they can store some fancy graphic characters in the character generator that can be addressed by the "nonprinting" ASCII codes not needed for their primary purposes. The Z-150 has some neat ones hidden in the character generator but they will not be transmitted to the printer. Some of the codes that are common to most computers and will be transmitted to the printer are shown in Table 1.

The Printing Codes

The tables in the back of most computer texts contain a table of the printing ASCII code. Each author tries his or her best to

generate a table that will be of the greatest use to them. My own version of the table of printing codes is shown as Table 2. Since the code is based on a bit pattern it is most easily understood when each column represents a hexadecimal number with the same highest order digit. I previously mentioned that the first 32D numbers represented "nonprinting" characters. These would be hexadecimal numbers with 0 and 1 as the high order (or left hand) digit. Since Table 2 consists of only the printing characters, the columns which would have 0 and 1 in the high order position when listed in hexadecimal form do not appear since they contain the nonprinting characters. Now let us study Table 2. The first column contains special symbols. The second column contains decimal numbers plus more special symbols. Note that the low order (right hand) digit of the code in hexadecimal form is the decimal value of the number (30H is the number 0 and 39H is 9). Also, note that very often the two characters that occupy the same key are in the same row in two left hand columns. The explanation point (!) is 21H and the digit 1 is 31H. The greater than symbol (>) is 3EH and the period (.) is 2EH. The designers of the code attempted to have only one bit different between characters that would normally occupy the same key on a typewriter keyboard.

When we examine the third and fourth column (those whose high order hexadecimal digits are 4 and 5) you will find that they contain the upper case letters used in the English (Roman) alphabet in alphabetical order. Since there are only twenty-six letters in our alphabet, there are six codes left which are used for some more of the symbols. However, if you speak a language "mit umlaut" or "con tilde" there is a good chance that codes 5BH to 5FH will include the upper case letters used in your language that are not in the twenty-six used in the English-speaking world. Note that the two right hand columns contain the lower case letters. Again, the same comment will apply to these that applied to the middle two. The ASCII code for a lower case letter is therefore always 20H or 32D greater than its upper case equivalent.

| Decimal Code | Hex Code | Description |
|--------------|----------|-----------------------------------|
| 7 | 7 | Rings the "bell" on terminal only |
| 8 | 8 | Back Space (1) |
| 9 | 9 | Horizontal tab |
| 10 | A | Line feed |
| 12 | C | Form Feed (2) |
| 13 | D | Carriage Return (3) |
| 27 | 1B | Escape |

Notes:

1. Backspace will erase the preceding character on the monitor. It will cause an overstrike on the printer.
2. The Form feed will probably cause no action on your monitor.
3. 13D/DH caused both a carriage return and a line feed on the Z-150.

Be very careful using the code 19D/13H because the XOFF command will turn off communication between the computer and the printer. Depending on the method of "handshaking" between the computer and printer you could "crash" your system.

Table 1
Selected "Nonprinting" Codes

Using The ASCII Codes

Let's take a look at a few BASIC programs than will permit us to examine some of the many uses of the ASCII codes. Here is the coding for the program in CP/M or GW versions of MBASIC:

```
100 REM A PROGRAM TO PRINT THE EQUIVALENT OF THE ASCII
    CODES
110 PRINT "ENTER A NUMBER BETWEEN 0 AND 127"
120 INPUT N
130 IF N > 255 THEN END
140 PRINT N,"( ";CHR$(N);")"
150 LPRINT N,"( ";CHR$(N);")"
160 GOTO 110
```

If you are using BHASIC or the HDOS version of MBASIC, you will have to modify Line 150 and add the appropriate opening statements to permit output to the printer.

Lines 140 and 150 are the key to the program. The first item in the output string is an echo of the input variable so that both the monitor and the printer will show the decimal value of the ASCII code. The comma following the N causes the cursor or print head to skip to the beginning of the next print field.

The next item in the output string is an opening parenthesis "(" as a literal. It is followed by a semicolon so that the cursor or print head does not skip to the next print field. The third item in the output string is the BASIC function that converts the value N to its ASCII equivalent. It is again followed by a semicolon.

The final item in the output string is the closing parenthesis as a literal. The result is a line containing the numeric value of the particular ASCII code, followed by its equivalent enclosed in parentheses. If there is no conversion of the number to its ASCII equivalent, the two parentheses will be adjacent. If the backspace occurs, the monitor will show only the closing parenthesis, while the printer will show an overstrike of the two parentheses.

Tabs, line feeds and carriage returns will be obvious from the position of the opening and closing parentheses. AGAIN, I MUST REPEAT THE WARNING ABOUT USING THE DECIMAL VALUE NINETEEN. The XOFF command does weird things to the communication between the computer and the printer. You may end up having to turn off both the computer and the printer to recover from the problem that this transmission causes. Let us enter the program and see what happens.

The Output Of The Nonprinting Codes To The Monitor Of The H-89

When the first 32 decimal values are entered most of them cause no output to the screen of the 89. The two parentheses just follow one another. The following outputs are noted:

| N | Code | Screen Indication |
|----|----------------|---|
| 7 | Bell | rings the bell |
| 8 | Back Space |) [the opening parenthesis is overstruck] |
| 9 | Horizontal tab | () [the second parenthesis tabbed over] |
| 10 | Line Feed | ([line feed and carriage return] |
| 12 | Form Feed | skips 13 rows but does not cause return |
| 27 | Escape | ([I cannot find out what "Esc" is but it does not print or show any change in the next line] |

Note that ASCII 27 generates the escape code and the result is the same as pressing the ESCAPE key. That is why you see CHR\$(27) in so many BASIC programs.

For the Z100

WITH

ShowOff

YOUR GRAPHICS

WILL BE OUTSTANDING



U.S. Census Divisions

High Resolution Advantages

- Professional Appearance
- Special Effects
- Less Distortion

Improve your presentations and reports with ShowOff, the hi-res graphics editor for the Z100. ShowOff helps you draw, paint and include text to create outstanding graphics.

640 x 480 resolution
92 fill colors • 92 patterns
25 text styles

Easy to use, ShowOff will also display and enhance MacPaint pictures, Lotus graphs, CAD drawings, and other graphics.

ShowOff \$79 ShowOff with Logitech Mouse \$174 demo disk \$3
ShowOff with Epson Laser Printer \$CALL

ShowOff versions are available for Logitech, Microsoft, and PC Mouse; HIPAD and Summagraphics digitizers.

Order direct • VISA • MASTERCARD • check • CALL TODAY
min requirements: Z100 • COLOR RAM • 384K RAM • MS-DOS 2.0



HOGWARE COMPANY
470 BELLEVIEW
ST LOUIS, MO 63119
(314) 962-7833



SPECIAL SPRING SALE

CLEAN-UP YOUR DISK FILES AND ORGANIZE THE OPERATION OF YOUR COMPUTER SYTEM WITH THESE POWERFUL UTILITIES



★ A MUST FOR HARD-DISK OWNERS ★

ARCHIVE/FCEU (for PC, Z100 & CP/M-80 systems) ARCHIVE allows you to conveniently save groups of files under one file name with significant savings in disk space. Especially effective for large groups of small files. FCEU is a File Compression and Encryption Utility which can compress files as much as 50%. Password protection is available. Especially useful for large ASCII text files. List Price - \$39.95 each
Special Spring Price only \$34.95 each, or \$59.95 for both.

AUTOMENU (for PC systems only) This utility will completely automate your hard-disk operation. With AUTOMENU, you will never have to enter a DOS command or subdirectory name again! Just select the commands or programs you want from a menu. List Price - \$49.95
Special Spring Price only \$44.95

DS BACKUP + (for PC systems only) Now a fast and effective back-up program for your hard-disk. Much faster and more flexible than the standard DOS back-up utilities. Selective or full back-ups with a compression option to reduce the number of floppies required. List Price - \$79.95
Special Spring Price only \$59.95

MACE UTILITIES (for PC systems only) This powerful utility set allows you to reclaim files from damaged disks, undelete files, and restore files from disks that have been accidentally re-formatted! Also, utilities to optimize disk speed by using cache memory and by defragmenting disk file space. List price \$99.95
Special Spring Price only \$89.95

★ MORE GREAT UTILITIES ★

CATALOG-MASTER (or PC, Z100 & CP/M-80 systems) Now you can get a sorted directory listing of all the files contained on several floppy disks. Disks are catalogued by merely putting them into your computer! You save time looking for files because you know exactly which disk they are on. List Price - \$39.95 each
Special Spring Price only \$34.95

FILE-PATCH (for PC, Z100, & CP/M-80 systems) This file display and edit program is especially useful for patching binary program files. Data is displayed in both ASCII and Hex formats. A full-screen edit mode and on-screen help key make this utility very easy to use. List Price - \$39.95
Special Spring Price only \$34.95

F-TRANS (for PC, Z100, & CP/M-80 systems) Allows you to transfer files between two computer systems regardless of system and disk format differences. Binary and text files can be transferred over modems or through a null-modem cable. Wildcards allow groups of files to be transferred with a single command. This utility is ideal for transferring files from old 8-bit machines to the newer 16-bit computers. List Price - \$59.95
Special Spring Price only \$49.95

PRNSPOOL (for PC & Z100 systems) With PRNSPOOL, you don't have to wait for your printer any longer! PRNSPOOL allows you to use your computer while the output from your word processor, spread-sheet, or other program is printing. Multiple printers and plotters can be supported on the same computer. List Price - \$49.95
Special Spring Price only \$39.95



ORDER TODAY
and let
GENERIC
help you with your
spring cleaning!

For faster delivery,
call
906-249-9801
DEPT. 47R, POB 790
MARQUETTE, MI 49855



Add \$3.00 per order for shipping and handling and 4% sales tax for Michigan residents. C.O.D. shipments available for \$2.00 additional. Specify computer model and operating system when ordering.



| DEC | HEX | CODE | DEC | HEX | CODE | DEC | HEX | CODE | DEC | HEX | CODE | DEC | HEX | CODE | DEC | HEX | CODE |
|-----|-----|------|-----|-----|------|-----|-----|------|-----|-----|------|-----|-----|------|-----|-----|------|
| 32 | 20 | SP | 48 | 30 | 0 | 64 | 40 | @ | 80 | 50 | P | 96 | 60 | ~ | 112 | 70 | p |
| 33 | 21 | ! | 49 | 31 | 1 | 65 | 41 | A | 81 | 51 | Q | 97 | 61 | a | 113 | 71 | q |
| 34 | 22 | " | 50 | 32 | 2 | 66 | 42 | B | 82 | 52 | R | 98 | 62 | b | 114 | 72 | r |
| 35 | 23 | # | 51 | 33 | 3 | 67 | 43 | C | 83 | 53 | S | 99 | 63 | c | 115 | 73 | s |
| 36 | 24 | \$ | 52 | 34 | 4 | 68 | 44 | D | 84 | 54 | T | 100 | 64 | d | 116 | 74 | t |
| 37 | 25 | % | 53 | 35 | 5 | 69 | 45 | E | 85 | 55 | U | 101 | 65 | e | 117 | 75 | u |
| 38 | 26 | & | 54 | 36 | 6 | 70 | 46 | F | 86 | 56 | V | 102 | 66 | f | 118 | 76 | v |
| 39 | 27 | ' | 55 | 37 | 7 | 71 | 47 | G | 87 | 57 | W | 103 | 67 | g | 119 | 77 | w |
| 40 | 28 | (| 56 | 38 | 8 | 72 | 48 | H | 88 | 58 | X | 104 | 68 | h | 120 | 78 | x |
| 41 | 29 |) | 57 | 39 | 9 | 73 | 49 | I | 89 | 59 | Y | 105 | 69 | i | 121 | 79 | y |
| 42 | 2A | * | 58 | 3A | : | 74 | 4A | J | 90 | 5A | Z | 106 | 6A | j | 122 | 7A | z |
| 43 | 2B | + | 59 | 3B | ; | 75 | 4B | K | 91 | 5B | [| 107 | 6B | k | 123 | 7B | { |
| 44 | 2C | , | 60 | 3C | < | 76 | 4C | L | 92 | 5C | \ | 108 | 6C | l | 124 | 7C | ~ |
| 45 | 2D | - | 61 | 3D | = | 77 | 4D | M | 93 | 5D |] | 109 | 6D | m | 125 | 7D | |
| 46 | 2E | . | 62 | 3E | > | 78 | 4E | N | 94 | 5E | ^ | 110 | 6E | n | 126 | 7E | ~ |
| 47 | 2F | / | 63 | 3F | ? | 79 | 4F | O | 95 | 5F | _ | 111 | 6F | o | 127 | 7F | DEL* |

* DEL is not used in most computers

Table 2
The Printing Codes

The Output Of The Nonprinting Codes To The Monitor Of The Z-150 And Z-148

It seems that the designer of the character generator in the Z-150 has decided to store a whole bunch of neat graphic characters in that ROM. When you use the nonprinting ASCII codes they are most intriguing. Among (and only some) of them are:

- 2 a face
 - 3 a heart
 - 4 a diamond
 - 5 a chess king
 - 6 a chess pawn
 - 24 an up arrow
 - 25 a down arrow
- and many others

It should be noted that these symbols are not transmitted to the printer. The same character generator is used in the Z-148.

The Nonprinting Codes And The Printer

The nonprinting codes can be useful in controlling the printer. Eight, the backspace code, causes the printer to backspace and then print the next character. This will permit you to overstrike or underline a character. Nine, the horizontal tab code, causes the print head to tab over to the next set tab location.

Ten will either generate a line feed or a line feed and a carriage return. Eleven will generate a vertical tab on some printers. Twelve generates a form feed and advances the paper to the top of the next page. Thirteen will generate a carriage return but no line feed on some printers. Fourteen through eighteen are used by some dot matrix printers to vary the pitch (horizontal width) of the characters. WATCH OUT FOR THE EFFECT OF NINETEEN!!! XOFF can cause trouble.

Since printers and their device drivers can be configured to respond to these nonprinting codes in various ways, only a check using the test program or thorough reading of the instruction manual will tell you exactly how your particular combination of printer and device driver will respond to a particular code. Since the dot matrix printers have their own character generating ROMs, do not expect the characters that the non-printing codes place on the monitor screen to be copied by the printer, even when you depress the screen copy key.

The Printing Codes And The Monitor

If the character generator in your monitor is designed to generate the standard English language characters, you should have no

problem and the response should match the table. If you have a monitor with a character generator configured for some other western language, some of the special characters will be replaced by the appropriate letters. The MS-DOS KEYBXXXX command permits you to replace the character generating ROM with a set character generator for several languages that can be stored in RAM. These commands convert the keyboard of the computer to the keyboard equivalent for the particular language.

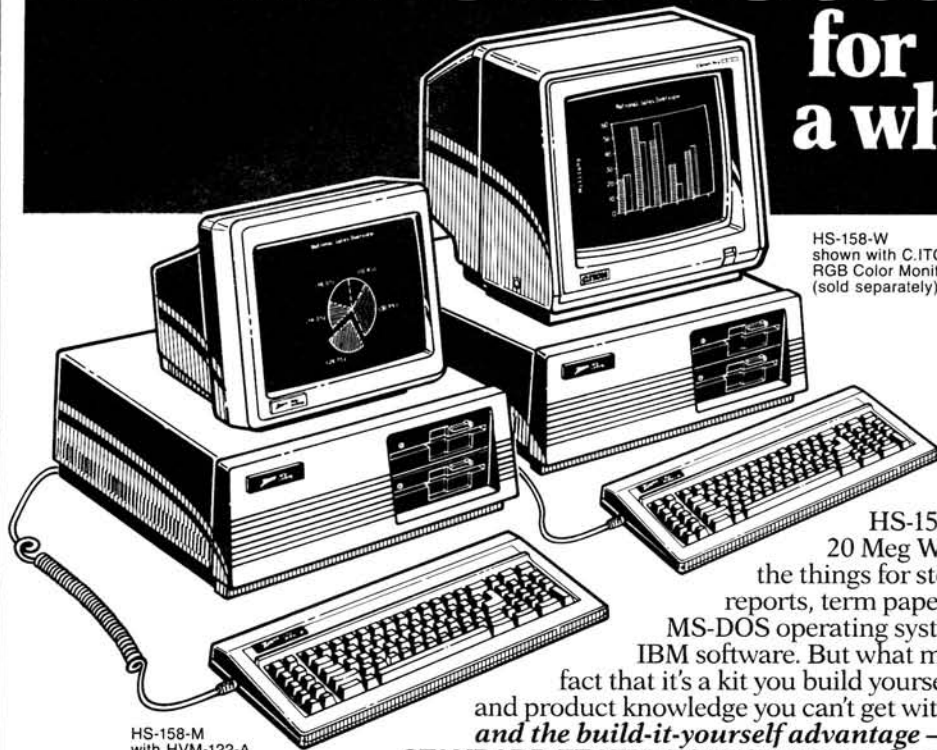
The Printing Codes And The Printer

Normally one would expect the printer to respond to the printing codes in a one to one match. There are some problems that can occur. If the printer is configured for some language other than English, some of the special characters will be replaced by characters from that language. Some impact printers come with print wheels or thimbles that do not respond to the ASCII codes but were designed for typewriter applications. These will print the characters as they would appear on a normal typewriter keyboard and not on the computer keyboard. One of the print wheels for my Daisywriter is set up in this manner and among other things, prints a superscript 2 in place of the karat "^^". It makes program listing unreadable. However, most printers will respond to the printing codes in the expected manner. The MS-DOS KEYBXXXX command only changes the character generator for the monitor and not for the printer.

The Extended Character Set

Up to this point we have only used 128 ASCII codes. The conversion to the eight bit ASCII coding system presents the opportunity for 256 different codes (0D to 255D). That is why line 130 in the BASIC program sets an upper limit of 255D. Integers greater than 255D will generate an error. These additional 128 codes may be used in different ways. The H-89, for reasons that I cannot phathom, repeats the first 128 ASCII codes. The Z-148 and Z-150 have a true extended character set in their character generator. These include many non-English (I should say non-Roman) letters, graphic symbols and such goodies as plus/minus (241), equal or greater (242), equal or less (243), the common division symbol (246), and the square root symbol (251). While these characters appear on the monitor screen they are not printed by the Panasonic printer. Lotus 123 has its own extended character set. If your operating system has the GRAPHIC and GRAFTBL function, you may be able to print these characters on a dot matrix printer.

A whole lotta meg for a whole lot less



HS-158-M
with HVM-122-A
Monitor

HS-158-W
shown with C.I.TOH
RGB Color Monitor
(sold separately)

The IBM PC compatible kit computer with memory to spare

Having enough memory and storage for your business, education, or home use is important. And the Heath/Zenith

HS-158 was designed with that in mind. Its 20 Meg Winchester and 640K memory are just the things for storing volumes of important financial reports, term papers, and budgets. The HS-158 uses the MS-DOS operating system and runs most industry-standard IBM software. But what makes this PC even more special is the fact that it's a kit you build yourself. For that pride of accomplishment and product knowledge you can't get with other PCs. *More memory/storage and the build-it-yourself advantage – a combination that can't be beat.*

STANDARD FEATURES INCLUDE: • color and mono support • serial/parallel port • 6 expansion slots • 5 or 8 Meg speed switch • PC compatibility

HS-158-M with FREE
HVM-122-A monochrome
monitor, one 5¼" drive,
640K RAM, diagnostic
software and DOS

ONLY
\$1379*

HS-158-W with one 5¼" drive,
640K RAM, 20 Meg Winchester
with controller, diagnostics and
DOS. (Monitor not included.)
C.I.TOH RGB Color Monitor **ONLY \$369**

ONLY
\$1579*

*HUG members get 10% off

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

PHOENIX, AZ
2727 W. Indian School Rd.
602-279-6247

TUCSON, AZ
7109 E. Broadway
602-885-6773

ANAHEIM, CA
330 E. Ball Rd.
714-776-9420

CAMPBELL, CA
2350 S. Bascom Ave.
408-377-8920

EL CERRITO, CA
6000 Potrero Ave.
415-236-8870

LA MESA, CA
8363 Center Dr.
619-461-0110

LOS ANGELES, CA
2309 S. Flower St.
714-749-0261

POMONA, CA
1555 N. Orange Grove Ave.
714-623-3543

REDWOOD CITY, CA
2001 Middlefield Rd.
415-365-8155

SACRAMENTO, CA
1860 Fulton Ave.
916-486-1575

WOODLAND HILLS, CA
22504 Ventura Blvd.
818-883-0531

WESTMINSTER, CO
8725 Sheridan Blvd.
303-429-2292

JACKSONVILLE, FL
8262 Arlington Expressway
904-725-4554

HALEAH, FL
4705 W. 16th Ave.
305-823-2280

PLANTATION, FL
7173 W. Broward Blvd.
305-791-7300

TAMPA, FL
4019 W. Hillsborough Ave.
813-886-2541

ATLANTA, GA
5285 Roswell Rd.
404-252-4341

HONOLULU, HI
98-1254 Kaahumanu St.
808-487-0029

CHICAGO, IL
3466 W. Devon Ave.
312-583-3920

DOWNERS GROVE, IL
224 Ogden Ave.
312-852-1304

INDIANAPOLIS, IN
2112 E. 62nd St.
317-257-4321

MISSION, KS
5960 Lamar Ave.
913-362-4486

LOUISVILLE, KY
12401 Shelbyville Rd.
502-245-7811

KENNER, LA
1900 Veterans Memorial Hwy.
504-467-6321

BALTIMORE, MD
1713 E. Joppa Rd.
301-861-4446

ROCKVILLE, MD
5542 Nicholson Lane
402-881-5420

PEABODY, MA
242 Andover St. (Rt. 114)
617-531-9330

WELLESLEY, MA
165 Worcester St. (Rt. 9)
617-237-1510

FARMINGTON HILLS, MI
29433 Orchard Lake Road
313-553-4171

EAST DETROIT, MI
18149 E. Eight Mile Rd.
313-772-0416

ST. JOSEPH, MI
2987 Lake Shore Drive
616-982-3215

HOPKINS, MN
101 Shady Cak Rd.
612-938-6371

ST. PAUL, MN
1645 White Bear Ave.
612-778-1211

BRIDGETON, MO
3794 McKelvey Rd.
314-291-1850

OMAHA, NE
9207 Maple St.
402-391-2071

OCEAN, NJ
1013 Slate Hwy. 35
201-775-1231

FAIR LAWN, NJ
35-07 Broadway (Rt. 4)
201-791-6935

AMHERST, NY
3476 Sheridan Dr.
716-835-3090

JERICHO, LI
15 Jericho Turnpike
516-334-8181

ROCHESTER, NY
937 Jefferson Rd.
716-424-2560

N. WHITE PLAINS, NY
7 Reservoir Rd.
914-761-7690

GREENSBORO, NC
4620C W. Market St.
919-299-5390

CINCINNATI, OH
131 West Kemper Rd.
513-671-1115

CLEVELAND, OH
28100 Chagrin Blvd.
216-292-7553

COLUMBUS, OH
2500 Morse Rd.
614-475-7200

TOLEDO, OH
48 S. Byrne Rd.
419-537-1887

OKLAHOMA CITY, OK
7409 South Western
405-632-6418

FRAZER, PA
630 Lancaster Pike (Rt. 30)
215-647-5555

PHILADELPHIA, PA
6318 Roosevelt Blvd.
215-288-0180

PITTSBURGH, PA
3482 Wm. Penn Hwy.
412-824-3564

WARWICK, RI
558 Greenwich Ave.
401-738-5150

DALLAS, TX
12022C Garland Rd.
214-327-4835

FORT WORTH, TX
6825-A Green Oaks Rd.
817-737-8822

HOUSTON, TX
1704 W. Loop N.
713-869-5263

SAN ANTONIO, TX
7111 Blanco Rd.
512-341-8876

MIDVALE, UT
58 East 7200 South
801-566-4626

ALEXANDRIA, VA
6201 Richmond Hwy.
703-765-5515

VIRGINIA BEACH, VA
1055 Independence Blvd.
804-460-0997

SEATTLE, WA
505 8th Ave. N.
206-682-2172

VANCOUVER, WA
516 S.E., Chkalov Dr.
206-254-4441 – Washington
503-241-3776 – Oregon

MILWAUKEE, WI
5215 W. Fond du Lac Ave.
414-873-8250

Phone orders accepted
OFFER GOOD THROUGH
APRIL 15, 1987

Your TOTAL SERVICE computer center • Service • Support • Software • Accessories • User Training • Competitive Prices

HZC-289

Units of Veritechnology Electronics Corporation

Heath® ZENITH®
Computers & Electronics

How To Access The ASCII Codes

BASIC, Pascal and Lotus have functions that allow one to convert the numeric value of the code to its ASCII equivalent and the ASCII equivalent to its numeric value. In BASIC, they are the ASC function for converting a symbol to its numeric equivalent and the CHR\$ function which will convert the numerical value to its equivalent symbol. The ASC function is part of the ASCII standard BASIC and should appear in every dialect of BASIC. While its inverse, the CHR\$ function is seen more often in programs. It is not part of the standard BASIC and might have other forms in other dialects of BASIC. In Pascal, the corresponding functions are ORD, for ordinal value, and CHR. These functions only correspond directly when the internal representation of the characters is in the ASCII code. Lotus uses the CODE and the CHAR function for the same purposes.

For What Purposes Do I Use These ASCII Codes?

There are many uses for the conversion to and from ASCII code. I will only show three simple examples in BASIC. If you prefer another language, you can easily convert to the language of your choice. I am certain that these will help you to find more uses on your own.

1. The generation of the ESCAPE code: The use of the CHR\$ subroutine to generate an escape code in a BASIC program is commonly used. Assigning CHR\$(27) to a string variable permits the use of the ESCAPE codes for screen or printer control. There have been so many articles in REMark about the use of the ESCAPE codes that I would show my prejudice if I were to list only a few.
2. To determine if a character is a number or a letter: Since the decimal numbers are represented by the ASCII codes 48D to 57D, we can ask:

```
IF ASC(n)>47 AND ASC(n)<58 THEN
```

If the above statement is true, "n" is a decimal digit.

3. Convert a lower case letter to an upper case letter: Since the lower case letters are represented by the codes 97D to 122D and the upper case letters are represented by the codes 65D to 90D, one can convert a lower case letter to its upper case letter by subtracting 32D from the lower case code.

```
IF ASC(str) > 96 AND ASC(str) < 123  
THEN str = CHR$(ASC(str) -32)
```

where "str" is the letter to be converted to upper case if it is lower case.

I hope that this article leads you to a better understanding of what the ASCII codes are and what they can be used for. If you are not of the programming type, they will help you to increase your computer literacy. *

**Are you reading
a borrowed copy of REMark?
Subscribe now!**

50% MORE STORAGE ON YOUR HARD DRIVE

The KONAN KXP-230 increases disk storage capacity by 50% to 100%. An ST-255 drive rated at 20 Mb will reformat to hold 36 Mb. Cache memory increases access speed by up to 50%. Disk driver dynamically makes files contiguous. ST-506 interface works with most drives. Requires DOS 3.0 or greater.

| | Type | Board only | With ST-225 |
|-------------|------------|------------|-------------|
| For PC & XT | KXP-230Z | \$219.00 | \$619.00 |
| For AT | KXP-230ZAT | \$259.00 | \$659.00 |

Z-100 20Mb Drive \$749.00

Hard drive system with KONAN DGC-2000 controller, 1/2 height Seagate drive. Software supports 1 to 4 partitions on 1 or 2 drives. Requires DOS 2.0 or greater.

| | Drive | Internal | External |
|-------|--------|-----------|-----------|
| 20 Mb | ST-225 | \$ 749.00 | \$ 849.00 |
| 45 Mb | ST-251 | \$1199.00 | \$1299.00 |

One year manufacturer warranty on above items.
Add \$5.00 for UPS shipment.

LINDLEY SYSTEMS

21 Hancock St. Bedford MA 01730 (617) 275-6821

HOME FINANCE SYSTEM VERSION 2

—An extensive Home Finance System that keeps track of checking, asset accounts (cash, savings, IRAs, CDs), and regular bill payments. Let your printer write your checks for you on any business-sized check (design your own check format).

—Checks have user defined codes and a separate flag for tax deductible items.

—Many reports, including listing all checks, or checks by codes or tax flag.

—System consists of 130 page users manual with 5 program disks (5-1/4") and a sample data disk.

Hardware: H8/HZ89 (64K) or HZ100 with 2 disk drives. Any Heath®, Zenith® or other printer.

Software: CP/M or CP/M-85/86 (Ver. 2.2) and MBASIC 5.21 for CP/M.

Order: Complete System \$89† (specify hard or soft sector 5 1/4", HZ89 or HZ100). Manual alone \$21.†

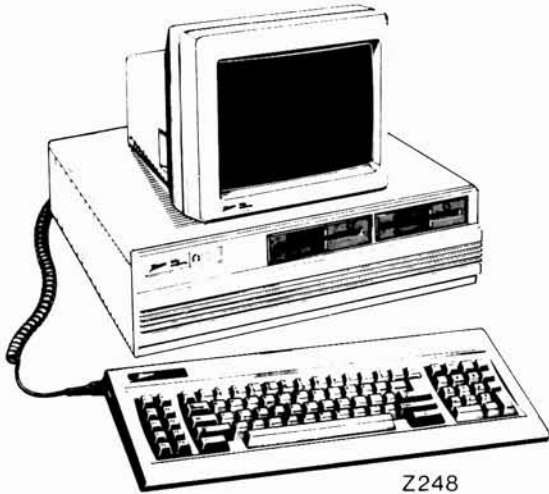
MasterCard/Visa accepted, please include your phone number.

†Prices include shipping.



Jay H. Gold, M.D.
Jay Gold Software
Box 2024, Des Moines, IA 50310
(515) 279-9821

First Capitol Delivers Zenith. Your Way.



Z248



Z181

Specialists in
CAD Systems
Local Area Networking



Z148

Have it your way.

If you're looking for a PC, you couldn't do better than Zenith. And you couldn't buy your Zenith at a better place than First Capitol. We'll put together a system the way YOU want it. And we'll do it for a price that competes with the guy who just sells boxes that fall on his loading dock.

Great Prices. And A Great Machine.

We're committed to Zenith. And to you. the standard Zenith models that come from the factory are a good buy. And we'll be glad to sell you one at a price that will match anyone. But if you want the full potential, we'll fix it up for you. Larger disk drives. More memory. Internal tape back up. Removable winchesters. All at a budget pleasing price.

We make Zenith Scream.

When a new model comes out, we're among the first to get it. Our Zenith fanatics take it apart, prod it and poke it until it screams for mercy. Then we put it back together (usually with some new parts) so that it REALLY screams. With power.

You'll be back.

Check with us. Check elsewhere. We'll put together a custom quotation for you we GUARANTEE you'll find interesting. We think you'll come back to us. There's a reason for the growing list of companies, government agencies, and end users that know us as the source for Zenith. Your Way.



1106 First Capitol Dr.
St. Charles, MO 63301

1-800-TO-BUY-IT (800-862-8948) Orders and quotes

1-314-724-2336 Technical support and order status

The StockSystem Challenge.

We will meet or beat any dealer's price on new stock Zenith systems. And for enhanced systems, no one can touch our combination of value, performance, and custom service.

Before you buy

ZENITH | data
systems

AUTHORIZED SALES AND SERVICE

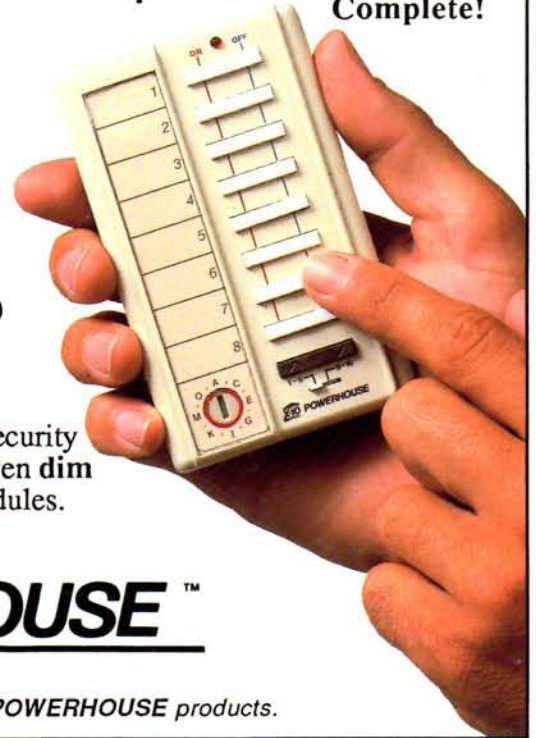
Call Us First. And Last.

Never come home to a dark house again!



The *NEW* Hand held BC5000 Radio Controlled Security System lets you control your home from inside or outside - even from your car as you enter your driveway - for only

\$49.99 Complete!



The revolutionary *X-10 POWERHOUSE* Radio Controlled Security System lets you control your home at the touch of a button.

The BC5000 set includes a hand held Transmitter and a Transceiver / Appliance Module. The hand held Transmitter (BC504) controls a light (or appliance) plugged into the Transceiver /Appliance Module (BC501) from up to 150 ft. away and the BC501 also re-transmits over your house wiring to control up to 8 other *X-10 POWERHOUSE* Modules.

You plug a lamp into a Lamp Module*, plug a television, stereo or air conditioner into an Appliance Module* and replace important outdoor security lights with the Wall Switch Module*. The hand held Transmitter can even **dim** and **brighten** lights connected to Lamp Modules and Wall Switch Modules.

* Additional Modules are just

\$14.99 each

X-10 POWERHOUSE
NUMBER ONE IN HOME CONTROL

Visit your nearest Heath/Zenith store for more details on our exciting line of *X-10 POWERHOUSE* products.



Hilltop Road
Saint Joseph, Michigan 49085

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

POSTMASTER: If undeliverable,
please do not return.

P/N 885-2087