

REMark

September 1991



The Official Zenith Data Systems Users Magazine





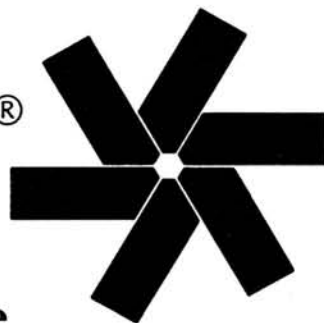
Share the Knowledge!

Have you done something interesting with your computer lately? Found a piece of software or hardware you don't know how you got along without? Designed a new product, be it software or hardware, for your system? By submitting this information in the form of a major article, you can share with others the knowledge of a particular subject. REMark magazine is currently looking for authors (novice or professional) to write articles. Even if you have never written before, give it a try! As a REMark author, you will receive up to \$400 for each article accepted and published. (For more information on current policies, call Lori Lerch at 616-982-3794.) So, Let's get to it and ...

Share the Knowledge!

REMark[®]

September 1991



The Official Zenith Data Systems Users Magazine

Introducing . . .

The 386 SL Microprocessor

John Ross 5

Batch Files

John Day 9

Hard Disk Survival Kit

Henry Fale 18

Heath/Zenith Useful

PC Computer Upgrades

Henry Fale 21

Getting the Most From Your Computer Part 4

John P. Lewis 26

Introduction to C++ Tenth Installment

Lynwood H. Wilson 31

Virus Protection and Disaster Recovery

Alan Neibauer 35

Getting Started With . . .

Laser Printer Font Tools

Alan Neibauer 39

Port-O-Call: COM1

Laura White 43

Modems

Part 4

Robert C. Brenner 45

Advertising

Page
No.

FBE Research Co., Inc. 29
WS Electronics 17
QuikData, Inc. 4

Resources

Software Price List 2
Classified Ads 34

Software

Managing Editor *Software Engineer*
 Jim Buszkiewicz Pat Swayne
 (616) 982-3837 (616) 982-3463

Production Coordinator *Secretary*
 Lori Lerch Lisa Cobb
 (616) 982-3794 (616) 982-3463

COM1 Bulletin Board *ZUG*
 (616) 982-3956 *Software Orders*
 (Modem Only) (616) 982-3463

Contributing Editor *Printer*
 William M. Adney Imperial Printing
 St. Joseph, MI

Advertising *Contributing Editor*
 Rupley's Advertising Service Robert C. Brenner
 Dept. REM, 240 Ward Avenue
 P.O. Box 348
 St. Joseph, MI 49085-0348
 (616) 983-4550

To Locate your Nearest:
 Dealer 1-800-523-9393
 Service Center 1-800-777-4630

	U.S.	APO/FPO & Domestic	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 ZUG Product List for availability of bound volumes of past issues. Requests for magazines mailed to foreign countries should specify mailing method and include appropriate, additional cost.

Send Payment to: Zenith Users' Group
 P.O. Box 217

Benton Harbor, MI 49023-0217
 (616) 982-3463

Although it is a policy to check material placed in REMark for accuracy, ZUG 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 Zenith Data Systems Computer Centers.

ZUG is provided as a service to its members for the purpose of fostering the exchange of ideas to enhance their usage of Zenith Data Systems equipment. As such, little or no evaluation of the programs or products advertised in REMark, the Software Catalog, or other ZUG publications is performed by Zenith Data Systems, in general, and Zenith Users' Group, in particular. The prospective user is hereby put on notice that the programs may contain faults, the consequence of which Zenith Data Systems, in general, and ZUG, in particular, can not 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 Zenith Users' Group, St. Joseph, Michigan.

Copyright (c) 1991, Zenith Users' Group

PRODUCT NAME	PART NUMBER	OPERATING SYSTEM	DESCRIPTION	PRICE
H8 - H/Z-89/90				
ACTION GAMES	885-1220-[37]	CPM	GAME	20.00
ADVENTURE	885-1010	HDOS	GAME	10.00
ASCIRITY	885-1238-[37]	CPM	AMATEUR RADIO	20.00
AUTOFIELD (Z80 ONLY)	885-1110	HDOS	DBMS	30.00
BHBASIC SUPPORT PKG	885-1119-[37]	HDOS	UTILITY	20.00
CASTLE	885-8032-[37]	HDOS	ENTERTAINMENT	20.00
CHEAPCALC	885-1131-[37]	HDOS	SPREADSHEET	20.00
CHECKOFF	885-8010	HDOS	CHKBK SOFTWARE	25.00
DEVICE DRIVERS	885-1105	HDOS	UTILITY	20.00
DISK UTILITIES	885-1213-[37]	CPM	UTILITY	20.00
DUNGEONS & DRAGONS	885-1093-[37]	HDOS	GAME	20.00
FLOATING POINT PKG	885-1063	HDOS	UTILITY	18.00
GALACTIC WARRIORS	885-8009-[37]	HDOS	GAME	20.00
GALACTIC WARRIORS	885-8009-[37]	CPM	GAME	20.00
GAMES 1	885-1029-[37]	HDOS	GAMES	18.00
HARD SECT SUPPORT PKG	885-1121	HDOS	UTILITY	30.00
HDOS PROG. HELPER	885-8017	HDOS	UTILITY	16.00
HOME FINANCE	885-1070	HDOS	BUSINESS	18.00
HUG DISK DUP UTILITY	885-1217-[37]	CPM	UTILITY	20.00
HUG SOFTWARE CATALOG	885-4500	VARIOUS	PROD TO 1982	9.75
HUGMAN & MOVIE ANIM	885-1124	HDOS	ENTERTAINMENT	20.00
INFO SYS AND TEL & MAIL SYS	885-1108-[37]	HDOS	DBMS	30.00
LOGBOOK	885-1107-[37]	HDOS	AMATEUR RADIO	30.00
MAGBASE	885-1249-[37]	CPM	MAGAZINE DB	25.00
MISCELLANEOUS UTILITIES	885-1089-[37]	HDOS	UTILITY	20.00
MORSE CODE TRANSCEIVER	885-8016	HDOS	AMATEUR RADIO	20.00
MORSE CODE TRANSCEIVER	885-8031-[37]	CPM	AMATEUR RADIO	20.00
PAGE EDITOR	885-1079-[37]	HDOS	UTILITY	25.00
PROGRAMS FOR PRINTERS	885-1082	HDOS	UTILITY	20.00
REMARK VOL 1 ISSUES 1-13	885-4001	N/A	1978 TO DEC '80	20.00
RUNOFF	885-1025	HDOS	TEXT PROC	35.00
SCICALC	885-8027	HDOS	UTILITY	20.00
SMALL BUSINESS PACKAGE	885-1071-[37]	HDOS	BUSINESS	75.00
SMALL-C COMPILER	885-1134	HDOS	LANGUAGE	30.00
SOFT SECTOR SUPPORT PKG	885-1127-[37]	HDOS	UTILITY	20.00
STUDENT'S STATISTICS PKG	885-8021	HDOS	EDUCATION	20.00
SUBMIT (Z80 ONLY)	885-8006	HDOS	UTILITY	20.00
TERM & HTOC	885-1207-[37]	CPM	COMMUN & UTIL	20.00
TINY BASIC COMPILER	885-1132-[37]	HDOS	LANGUAGE	25.00
TINY PASCAL	885-1086-[37]	HDOS	LANGUAGE	20.00
UDUMP	885-8004	HDOS	UTILITY	35.00
UTILITIES	885-1212-[37]	CPM	UTILITY	20.00
UTILITIES BY PS	885-1126	HDOS	UTILITY	20.00
VARIETY PACKAGE	885-1135-[37]	HDOS	UTILITY & GAMES	20.00
WHEW UTILITIES	885-1120-[37]	HDOS	UTILITY	20.00
XMET ROBOT X-ASSEMBLER	885-1229-[37]	CPM	UTILITY	20.00
Z80 ASSEMBLER	885-1078-[37]	HDOS	UTILITY	25.00
Z80 DEBUGGING TOOL (ALDT)	885-1116	HDOS	UTILITY	20.00
H8 - H/Z-89/90 - H/Z-100 (Not PC)				
ADVENTURE	885-1222-[37]	CPM	GAME	10.00
BASIC-E	885-1215-[37]	CPM	LANGUAGE	20.00
CASSINO GAMES	885-1227-[37]	CPM	GAME	20.00
CHEAPCALC	885-1233-[37]	CPM	SPREADSHEET	20.00
CHECKOFF	885-8011-[37]	CPM	CHKBK SOFTWARE	25.00
COPYDOS	885-1235-[37]	CPM	UTILITY	20.00
DISK DUMP & EDIT UTILITY	885-1225-[37]	CPM	UTILITY	30.00
DUNGEONS & DRAGONS	885-1209-[37]	CPM	GAMES	20.00
FAST ACTION GAMES	885-1228-[37]	CPM	GAME	20.00
FUN DISK I	885-1236-[37]	CPM	GAMES	20.00
FUN DISK II	885-1248-[37]	CPM	GAMES	35.00
GAMES DISK	885-1206-[37]	CPM	GAMES	20.00
GRADE	885-8036-[37]	CPM	GRADE BOOK	20.00
HRUN	885-1223-[37]	CPM	HDOS EMULATOR	40.00
HUG FILE MANAGER & UTILITIES	885-1246-[37]	CPM	UTILITY	20.00
HUG SOFTWARE CAT UPDT #1	885-4501	VARIOUS	PROD 1983 TO 1985	9.75
KEYMAP CPM-80	885-1230-[37]	CPM	UTILITY	20.00
MBASIC PAYROLL	885-1218-[37]	CPM	BUSINESS	60.00
NAVPROGSEVEN	885-1219-[37]	CPM	FLIGHT UTILITY	20.00
SEA BATTLE	885-1211-[37]	CPM	GAME	20.00
UTILITIES BY PS	885-1226-[37]	CPM	UTILITY	20.00
UTILITIES	885-1237-[37]	CPM	UTILITY	20.00
X-REFERENCE UTIL FOR MBASIC	885-1231-[37]	CPM	UTILITY	20.00
ZTERM	885-3003-[37]	CPM	COMMUNICATIONS	20.00

Price List

This Software Price List contains all products available for sale. For a detailed abstract of these products, refer to the Software Catalog, Software Catalog Update #1, or previous issues of REmark.

PRODUCT NAME	PART NUMBER	OPERATING SYSTEM	DESCRIPTION	PRICE
H/Z-100 (Not PC) Only				
CARDCAT	885-3021-37	MSDOS	BUSINESS UTILITY	20.00
CHEAPCALC	885-3006-37	MSDOS	BUSINESS UTILITY	20.00
CHECKBOOK MANAGER	885-3013-37	MSDOS	BUSINESS CPM EMULATOR	20.00
CP/EMULATOR	885-3007-37	MSDOS	DBMS	25.00
DBZ	885-8034-37	MSDOS	GAME	20.00
DUNGN & DRAGONS (ZBASIC)	885-3009-37	MSDOS	UTILITY	20.00
ETCHDUMP	885-3005-37	MSDOS	PRINTER PLOT UTIL	25.00
EZPLOT II	885-3049-37	MSDOS	GAMES	20.00
GAMES (ZBASIC)	885-3011-37	MSDOS	GAMES	25.00
GAMES CONTEST PACKAGE	885-3017-37	MSDOS	GAMES	25.00
GAMES PACKAGE II	885-3044-37	MSDOS	GAMES	20.00
GRAPHIC GAMES (ZBASIC)	885-3004-37	MSDOS	UTILITY	20.00
GRAPHICS	885-3031-37	MSDOS	UTILITY	20.00
HELPSCREEN	885-3039-37	MSDOS	UTILITY	20.00
HUG BKGRD PRINT SPOOLER	885-1247-37	CPM	UTILITY	20.00
KEYMAC	885-3046-37	MSDOS	UTILITY	20.00
KEYMAP	885-3010-37	MSDOS	UTILITY	20.00
KEYMAP CPM-85	885-1245-37	CPM	UTILITY	20.00
MATHFLASH	885-8030-37	MSDOS	EDUCATION	20.00
ORBITS	885-8041-37	MSDOS	EDUCATION	25.00
POKER PARTY	885-8042-37	MSDOS	ENTERTAINMENT	20.00
SCICALC	885-8028-37	MSDOS	UTILITY	20.00
SKYVIEWS	885-3015-37	MSDOS	ASTRONOMY UTILITY	20.00
SMALL-C COMPILER	885-3026-37	MSDOS	LANGUAGE	30.00
SPELL5	885-3035-37	MSDOS	SPELLING CHECKER	20.00
SPREADSHEET CONTEST PKG	885-3018-37	MSDOS	VARIOUS SPRDST	25.00
TREE-ID	885-3036-37	MSDOS	TREE IDENTIFIER	20.00
USEFUL PROGRAMS I	885-3022-37	MSDOS	UTILITIES	30.00
UTILITIES	885-3008-37	MSDOS	UTILITY	20.00
ZPC II	885-3037-37	MSDOS	PC EMULATOR	60.00
ZPC UPGRADE DISK	885-3042-37	MSDOS	UTILITY	20.00
H/Z-100 and PC Compatibles				
ADVENTURE	885-3016	MSDOS	GAME	10.00
BACKGRD PRINT SPOOLER	885-3029	MSDOS	UTILITY	20.00
BOTH SIDES PRINTER UTILITY	885-3048	MSDOS	UTILITY	20.00
CXREF	885-3051	MSDOS	UTILITY	17.00
DEBUG SUPPORT UTILITIES	885-3038	MSDOS	UTILITY	20.00
DPATH	885-8039	MSDOS	UTILITY	20.00
HADES II	885-3040	MSDOS	UTILITY	40.00
HEPCAT	885-3045	MSDOS	UTILITY	35.00
HUG EDITOR	885-3012	MSDOS	TEXT PROCESSOR	20.00
HUG MENU SYSTEM	885-3020	MSDOS	UTILITY	20.00
HUG SOFTWARE CAT UPD #1	885-4501	MSDOS	PROD 1983 - 1985	9.75
HUGMCP	885-3033	MSDOS	COMMUNICATION	40.00
ICT 8080 - 8088 TRANSLATOR	885-3024	MSDOS	UTILITY	20.00
MAGBASE	885-3050	VARIOUS	MAG DATABASE	25.00
MATT	885-8045	MSDOS	MATRIX UTILITY	20.00
MISCELLANEOUS UTILITIES	885-3025	MSDOS	UTILITIES	20.00
PS' PC & Z100 UTILITIES	885-3052	MSDOS	UTILITIES	20.00
REMARK VOL 8 ISSUES 84-95	885-4008	N/A	1987	25.00
REMARK VOL 9 ISSUES 96-107	885-4009	N/A	1988	25.00
REMARK VOL 10 ISSUES 108-119	885-4010	N/A	1989	25.00
REMARK VOL 11 ISSUES 120-131	885-4011	N/A	1990	25.00
SCREEN DUMP	885-3043	MSDOS	UTILITY	30.00
UTILITIES II	885-3014	MSDOS	UTILITY	20.00
Z100 WORDSTAR CONNECTION	885-3047	MSDOS	UTILITY	20.00
PC Compatibles				
CARDCAT	885-6006	MSDOS	CAT SYSTEM	20.00
CHEAPCALC	885-6004	MSDOS	SPREADSHEET	20.00
CLAVIER	885-6016	MSDOS	ENTERTAINMENT	20.00
CP/EMULATOR II & ZEMULATOR	885-6002	MSDOS	CPM & Z100 EMUL	20.00
DUNGEONS & DRAGONS	885-6007	MSDOS	GAME	20.00
EZPLOT II	885-6013	MSDOS	PRINTER PLOT UTIL	25.00
GRADE	885-8037	MSDOS	GRADE BOOK	20.00
HAM HELP	885-6010	MSDOS	AMATEUR RADIO	20.00
KEYMAP	885-6001	MSDOS	UTILITY	20.00
LAPTOP UTILITIES	885-6014	MSDOS	UTILITIES	20.00
PS' PC UTILITIES	885-6011	MSDOS	UTILITIES	20.00
POWERING UP	885-4604	N/A	GUIDE TO USING PCs	12.00
SCREEN SAVER PLUS	885-6009	MSDOS	UTILITIES	20.00
SKYVIEWS	885-6005	MSDOS	ASTRONOMY UTIL	20.00
TCSPELL	885-8044	MSDOS	SPELLING CHECKER	20.00
ULTRA RTTY	885-6012	MSDOS	AMATEUR RADIO	20.00
YAUD (YET ANOTHER UTIL DSK)	885-6015	MSDOS	UTILITIES	20.00

Attention!!

Zenith Data Systems Owners

When ordering ZUG software, be sure to specify what disk format you would like us to put it on. If you have an H-8, H/Z-89, or H/Z-90, you have the choice of using hard- or soft-sectored disks depending on your drive type. Order soft-sectored by adding a -37 to the end of the part number (i.e., 885-8009-37). Leaving off the -37 specifies a hard-sectored disk (i.e., 885-8009). If you own an H/Z-100 (not PC) series computer, you will always use the -37 at the end of the part number. For PC users, you have the choice of 5-1/4" (-37), 3.5" (-80), or 2" (-90) disks. Just add this number to the end of the ZUG part number (i.e., 885-3009-37, 885-3007-80, 885-3007-90).

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

ORDERING INFORMATION

For VISA, MasterCard, and American Express phone orders, telephone the Zenith Users' Group directly at (616) 982-3463. Have the part number(s), description(s), and quantity ready for quick processing. By mail, send your order, plus 10% postage and handling (\$1.00 minimum charge, up to a maximum of \$5.00) to: Zenith Users' Group, P.O. Box 217, Benton Harbor, MI 49023-0217. VISA, MasterCard and American Express require minimum \$10.00 order. No C.O.D.s accepted.

Questions regarding your subscription? Call Lisa Cobb at (616) 982-3463.

QUIKDATA - 15 YEARS OF H/Z SUPPORT!

YOUR H/Z ENHANCEMENT EXPERTS!

ACCELERATE YOUR PC/XT/AT!

From Sota Technologies, Inc., the fastest and most proven way to give new life to your H/Z PC/XT computer, giving it -AT compatible speeds! Turn your turtle into a -286 rabbit with a 12.5 Mhz 80286 or 80386 16Mhz SX accelerator board. Complete with 16K on-board CACHE.

The EXP12 286i is the effective solution, making your H/Z150/160/150/158/159 series of computers, or any general PC/XT computer faster, in many cases, than a standard IBM AT type computer! **You won't believe your stop watch!**

EXP-12 - \$275

EXP386 - \$449 Much faster 16Mhz 80386 SX version

For your H/Z241 or 248 we have the MicroWay 20Mhz 386SX accelerator which plugs into a slot and cables to the CPU. Run enhanced mode software such as Windows V3. 32K Cache with optional 32K add-on for 64K cache. Landmark 27, Norton SI 22.4. Cable extra, specify which is needed.

MWFCACHE - \$450 20Mhz 386SX Accelerator card

MWCLCC - \$95 LCC interconnect cable

MWPLCC - \$95 PLCC interconnect cable

MWCA32 - \$65 32K Cache add-on option

387SX20 - \$149 Optional 20Mhz coprocessor

MEMORY UPGRADES

Note: All memory upgrades come without memory chips. 150ns 256K DRAMs are \$1.79 as of this printing.

Z150MP - \$17 Will allow you to upgrade your H/Z150/160 to up to 704K on the main memory board, using up to 18 256K DRAM chips.

EAZYRAM - \$89 Upgrades EaZy PC from 512 to 640K

ZMF100 - \$55 Will allow you to upgrade your H/Z110/120 (old motherboards; with p/n less than 181-4918) to 768K system RAM. Requires 27 256K DRAM chips.

Z100MP - \$55 Similar to ZMF100 above, but for new motherboards with p/n 181-4918 or greater.

3MB RAM BOARD for Z241/248 computers is an excellent memory card. Will backfill your 512 to 640K, and provide both extended and expanded RAM; all can coexist. Uses 100ns M256-10 RAM chips, 36 per megabyte desired. Minimum of 18 DRAM chips required (\$1.95 each).

EVATRD - \$119

Z248/12, Z286LP RAM UPGRADE Z605-1 consists of 2MB SIMM 80ns RAM kits to upgrade your H/Z systems.

Z605-1 - \$149

Z386/20, Z386/25, Z386/33, Z386 EISA 2MB SIMM 80ns UPGRADE to add increments of 2MB to these systems. Two required.

ZA3600ME - \$89

ZA3800MK - \$349 4 megabyte SIMM upgrade for Z386/20, /25, /33, /33E. Must have 4-1 meg SIMMs installed first.

WINCHESTER UPGRADE KITS

PCW20 - \$239 Complete MFM winchester setup for a H/Z150, 148, 158, 159, 160, PC etc. Includes 21 meg formatted half-height Segate ST-124 37ms drive, controller, cable set, doc.

ST-124 - \$195 Bare drive only

PCW30 - \$295 32 meg with 28ms Segate ST-138-1.

ST-138 - \$249 Bare drive only

PCW40 - \$319 42 meg with 28ms Segate ST-251-1.

ST-251-1 - \$269 Bare drive only

PCW80 - \$595 80 meg with Segate ST-4096 full size drive.

ST-4096 - \$549 Bare drive only

We also have the DTC controllers (\$59) and daughter board expansions (\$65) to place a hard drive in the H/Z148 computers.

WDCON - \$59 PC/XT hard drive controller board

WDATCONF - \$95 1:1 interleave HD/floppy controller for AT's

FLOPPY DRIVE SAMPLE

MF501 - \$71 5" 360K DS/DD drive

MF504 - \$79 96 TPI 1.2 meg AT/Z100 drive

MF353 - \$79 720K 3.5" drive in 5" frame

MF355 - \$85 1.4 meg 3.5" AT drive in 5" frame

TM100-2R - \$69 40tk DS reburb (H8/89/Z100 PC type)

Also other drives and full line for older systems

ADD AN EXTERNAL HARD DRIVE TO ANY LAPTOP OR PC DESKTOP

with our EXPORT hard drive. Plugs into a parallel port (do not lose the port) to give you an affordable way to easily add a hard drive to your computer. Fast!

EWIN20 - \$495 20Mb unit

EWIN40 - \$625 40Mb unit

ANY DRIVE IN YOUR PC/XT/AT

With the CompatiCard, you can install up to four additional drives, of any type in your PC/XT/AT computer. Add a 1.2 meg 5" floppy, or a 1.44 meg 3.5" floppy, or any other drive, including 8" to your system. The CompatiCard (CCARD) will handle up to four drives, and the CompatiCard II (CCARD2) will handle up to 2 drives. CCARD4 has boot ROM to allow it to be used as primary boot controller in systems that allow you to remove floppy controller in some systems. Also handles 2.8MB 3.5" floppy drives. Additional cables and external enclosures may be required.

CCARD2 - \$79 CCARD - \$99 CCARD4 - \$125

ADD ANY FLOPPY TO ANY LAPTOP OR PC WITH A PARALLEL PORT

easily and inexpensively. With Backpack, you simply connect the external unit to your parallel printer port (do not lose printer function), install software, and away you go! No expansion boxes needed for laptops, and no slots required. Too easy to be true, but it is. Want a 2.8mb/1.4mb/720K floppy on your MinisPort? Want to add a 1.2 meg to your laptop? Want to add an additional drive to your desktop? Plug it in and go. 2.8MB 3.5" version will read and write 2.8 meg, 1.4 meg, and 720K format. 1.2 meg version will read 1.2 meg and 360K and write 1.2 meg format.

BPACK2.8 - \$319

BPACK1.4 - \$269

BPACK1.2 - \$269

BPACK360 - \$269

8-BIT/Z100

We carry a full line of replacement boards, parts and power supplies for the H/Z89/90 and Z100. We also have some H8 boards available. We continue to fully support and carry a full line of hardware and software products for the H8/H89/90 and Z100 computers including almost the full line of Software Toolworks items. Other items we carry include diskettes of all types, printers, modems, hard drives, etc.

OTHER STUFF

Quikdata also carries BIOS ROM upgrades and batteries for most H/Z PC/XT/AT computers, spike protection filters, backup power supplies, tape backup units, modems, printers, cables and ribbons, disk drives and diskettes of all types, external hard drive and floppy drive enclosures, cables and connectors, laptop batteries, CMOS batteries, video monitors and video cards, memory cards, memory chips and ICs, joysticks, accessory cards, serial and bus mouse, a variety of useful and most popular software and much more! **Need a PC/XT/AT computer?** Tell us what you want and we will quote you a price on one of our custom assembled QD computers made up to your exact specifications!

Call or write in to place your order, inquire about any products, or request our free no obligation catalog. VISA and Master Card accepted, pick up 2% S&H. We also ship UPS COD and accept purchase orders to rated firms (add 5% to all items for POs). All orders under \$100 add \$4 S&H. Phone hours: 9AM-4:30PM Mon-Thu, 9AM-3PM Friday. Visit our bulletin board: (414) 452-4345. FAX: (414) 452-4344.

QUIKDATA, INC.

2618 PENN CIRCLE

SHEBOYGAN, WI 53081-4250

(414) 452-4172

Introducing . . .

The 386 SL Microprocessor

John Ross
208 S. Second Street
Wakeeney, KS 67672

An Overview of the 80386 Microprocessor Family

When Intel introduced the 80386 family of microprocessors in 1985, they rapidly revolutionized the world of IBM-compatible microcomputers. With one inventive stroke, that world benefited from the 32-bit internal architecture of the 80386. With 32-bit processing, the 80386 microprocessor family broke previously untouchable speed barriers, accessed more system random-access memory than possible with previous computer systems and gained multi-tasking power. Figure 1 shows the capabilities of the 80386 microprocessor family.

Since 32 data lines connect the micro-

processor to associated circuitry, the microprocessor could accept 8-, 16- and 32-bit word formats. From a software standpoint, the 32-bit platform allows the microcomputer to run more types of operating system software, such as MS-DOS, Windows, OS-2 and Unix. Its ability to address 4 gigabytes of real memory and 64 terabytes of virtual memory adds to the speed advantage of the 80386 microprocessor. Multi-tasking simply means that several software applications may run at the same time.

From its early stages, the 80386 family has seen a design evolution. Designated as the 80386 DX, the first version features full 32-bit processing. Designed for process-

intensive applications, such as large databases or desktop publishing packages, the 80386 DX gives high processing speed and quick input/output operations. While early editions of the 80386 DX had operating speeds of 16 and 20 megahertz, the newer additions to the DX line offer speeds of 25 and 33 megahertz. As technology and prices have changed, the 80396 DX-based microcomputers have become the answer to mid-range office automation needs.

Another version of the 80386 microprocessor, the 80386 SX, combines the internal 32-bit internal architecture of the microprocessor with an external 16-bit bus. This option saves cost while offering the

Figure 1
80386 Feature Chart

Feature	386 DX CPU	386 SX CPU	i486 CPU	386 SL CPU
Central Processing Unit	Real Mode, Virtual 8086 Mode, 80286	Real Mode, Virtual 8086 Mode, 80286	Real Mode, Virtual 8086 Mode, 80286	Real Mode, Virtual 8086 Mode, 80286
Compatibility Modes	Protected Mode, 386/32-Bit Protected Mode	Protected Mode, 386/32-Bit Protected Mode	Protected Mode, 386/32-Bit Protected Mode	Protected Mode, 386/32-Bit Protected Mode
Internal Data Precision	8, 16, 32 Bits	8, 16, 32 Bits	8, 16, 32 Bits	8, 16, 32 Bits
Virtual Address Space	4 Gigabytes	4 Gigabytes	4 Gigabytes	4 Gigabytes
External Bus Width	32 Bits	16 Bits	32 Bits	16 Bits
Physical Address Space	4 Gigabytes	16 Megabytes	4 Gigabytes	32 Megabytes
Clock Frequency	20, 25, 33 MHz	16 or 20 MHz	25 or 33 MHz	D.C. 0Hz to 20 MHz
Co-Processor Support	80387 DX	80387 SX	Built-in	80387 SX
Cache-Controller	80385 DX or 80395 DX	80385 SX or 80395 SX	Built-in	Built-in

Courtesy of Intel Corporation

advantages of the 80386 microprocessor. Currently, the 80386 SX microprocessors have an operating speed of either 16 or 20 megahertz. With the appearance of the 80386 SX, 32-bit processing became available to the low-end desktop microcomputer market.

Continuing to answer the need for "managerial desktop" or "power office systems," Intel broke the 33 megahertz processing speed barriers with a late-1989 introduction. As a member of the 80386 family, the 80486 retains the now-familiar 32-bit architecture. Nevertheless, a higher level of integration sets the chip design apart from other family members. Over a million transistors produce a combination of the microprocessor, an 8-kilobyte cache memory, a math co-processor and memory management on one semiconductor device. Combining these units onto one chip reduces time lost during input/output transfers. Not surprisingly, the advances offered by higher integration continue to solve other performance dilemmas.

During October 1990, Intel introduced a new highly integrated microprocessor chip set again based on their successful 80386 microprocessor family. Two integrated circuits — one a microprocessor dubbed the 80386 SL and the other a peripheral support device called the 82360 SL — provide better support for the notebook microcomputer market. This better support takes the form of advanced power management and an increased operating speed of 20MHz. Fewer integrated circuits translate into reduced power consumption and higher processing speeds. Because of the high integration, ten components — on a four-by-six inch card — offer complete 32-bit software compatibility within a small, lightweight package. Recently, Zenith Data Systems announced a new line of notebook-sized microcomputers built around the Intel 386 SL chip set.

What Does the Newest Offspring Offer?

Like the 80486, the 80386 SL microprocessor utilizes over one million transistors on a single integrated circuit. However, those million transistors have a different purpose than the transistors found within the 80486. Instead of using the very large scale integration technology to boost computing speed, Intel designed the 80386 SL components around the specific needs of the laptop user. In the past, portable microcomputer users gave up size, weight and battery life to gain the power found in high-end desktop microcomputers.

With the 80386 SL SuperSet, Intel let the need for total system integration, reduced power usage, increased performance and maximum system flexibility to dictate the design and production of the microprocessor. Intel design teams talked with system manufacturers about needed features and the portable computing requirements

of the end user. Even with the concerns of the portable computer user in mind, Intel retains the standard 80386 capabilities on the 80386 SL.

Those standards include paged-memory-management, Lotus/Intel/Microsoft compatible expanded memory and the ability to directly support large amounts of system memory and an optional high-speed cache memory. Since the 80386 SL uses a ISA bus as a common interface, it also supports the large number of available third-party upgrade boards. Also, the SuperSet fully supports all standard peripherals. Figure 2 illustrates how the functions of a microcomputer can fit onto the two components of the SuperSet.

Saving Power the Old Way

A look at the world of portable computing before the 80386 SL highlights the improvements given by the new design. For maximum conservation of power, the microprocessor should stop entirely. However, conventional 32-bit microprocessors can not operate below a specified frequency. Although the system goes into a "standby" mode, the still-operating microprocessor continues to use power.

Since software governs power consumption in traditional portable systems, some microprocessor functions serve in a dual role. Conventional power management techniques require that input ports read the status of peripheral devices; output ports control the operations of peripheral devices; control algorithms fill the memory; and power management interrupt requests flow on the same lines required for either operating or application software. Before the LCD display dims or the disk drives shut down, dedicated hardware monitors peripheral usage. After the system enters the "standby" mode, the same hardware interrupts microprocessor action before any input/output operations with the disabled peripherals can take place.

All these additional requirements may tax the microprocessor to the point where reliability and compatibility become legitimate concerns. Software that runs with no difficulty on desktop microcomputers may not work as well on the portable microcomputer. An operating system contending for the same interrupt request lines as the power management software can cause a system "crash" and the resulting loss of data. If multiple software applications compete for the same interrupt routines or the same memory region, the functions of one application may interfere with the functions of another.

This problem becomes more apparent with the use of terminate-and-stay-resident programs, graphical user interfaces and utility packages. Another problem may occur on the power management side. As "DOS-extendors" and other protected-mode applications change or install inter-

rupt routines, the constant enabling and disabling of interrupts can disrupt power management software.

Inside the 80386 SL

Figure 3 depicts the internal structure of the 80386 SL chip. As seen in this figure, the chip features a standard 80386 central processing unit, static and dynamic RAM controllers, an integrated cache controller, the CPU clock and controllers for the internal and external buses. Each section features design characteristics tailored for the portable microcomputer user.

Even though the SL retains the features of its 80386 predecessors, the new design offers some new wrinkles. Using the new System Management Mode extension, the SL gives a safer, more intelligent operating level for battery management. Firmware running on the SL SuperSet allows an almost unlimited number of intermediate reduced-power modes. That is, elements of the microcomputer — such as the video display, the hard disk drive, the floppy disk drive and the main system memory — stay longer in the lowest-power scheme allowed.

Referring back to Figure 1, we see that the 80386 SL has a clock frequency that varies from 0 to 20 MHz. In itself, this marks a dramatic change from previous designs. When the need for a live battery outweighs the need for system performance, the CPU clocks and peripheral clocks can either stop or run slower for greater efficiency. An idle system will exhibit a completely stopped CPU clock with no data loss. A slowed clock frequency results when tasks that require little computational power — such as data entry or downloading — happen. Manipulating the clock frequency allows the 80386 SL-based portable to run for a month on the same charge that would normally last two days.

Gains in power management also take place through reduced power consumption in the main system memory. Taking over from traditional memory device controllers, the 80386 SL eliminates unnecessary transfers, precharge cycles and control line transitions. Instead of enabling every memory device during any given data transfer, the 80386 SL activates only the memory devices needed for the particular transfer. In addition, the microprocessor uses a programmable refresh rate for memory devices which allows refreshing to occur at the slowest, most power-efficient rate.

Another asset of the SL-based portable computer again sets it apart from conventional designs. Utilizing a cache allows code blocks to load from the dynamic RAM of main memory into a memory cache of static RAM. The faster access time of the static RAM speeds the transfer of data and instructions to the microprocessor. To eliminate unnecessary main-memory cycles, the design of the 80386 SL features

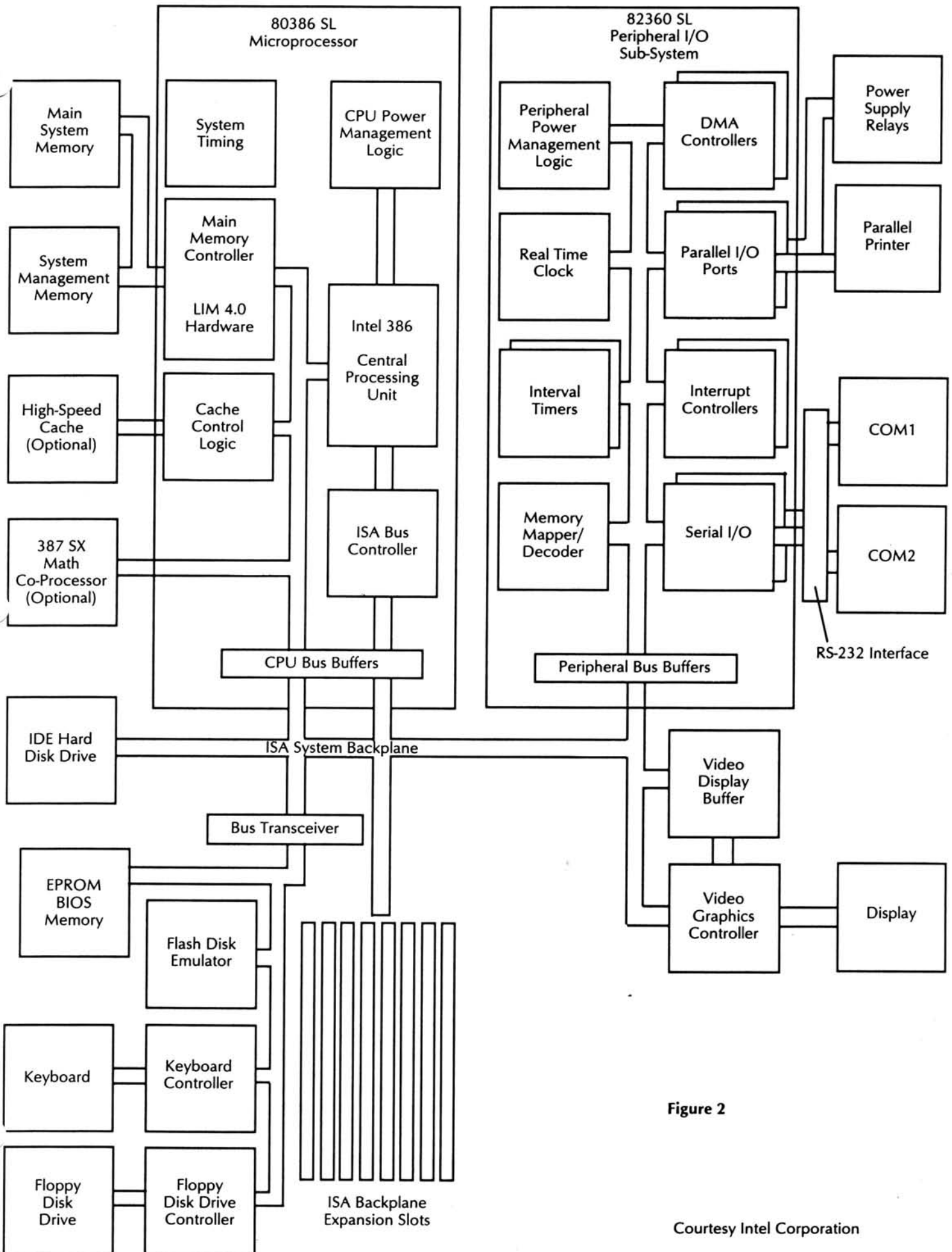


Figure 2

Courtesy Intel Corporation

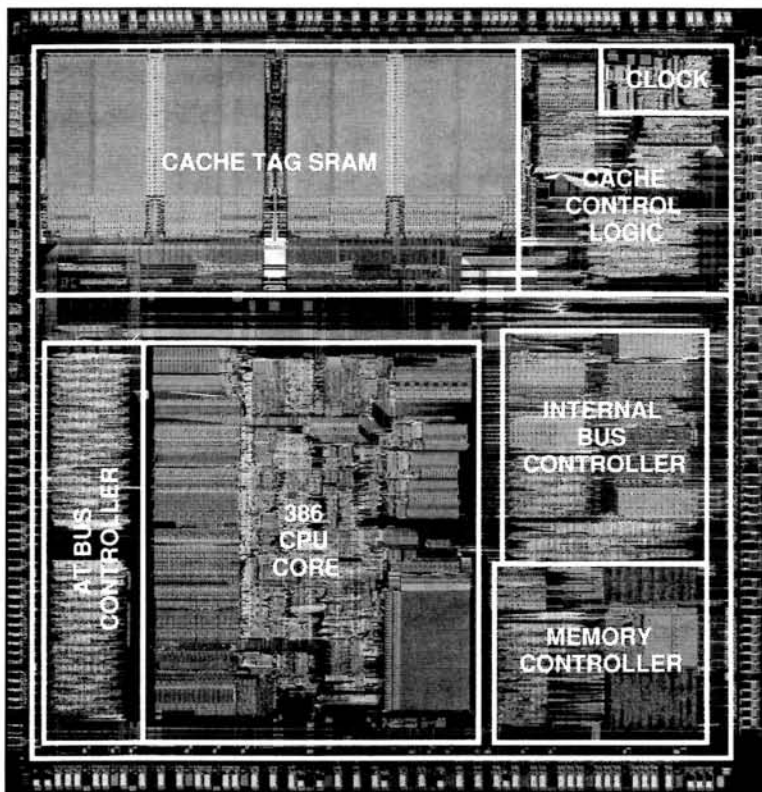


Figure 3

The 386 SL Microprocessor contains 855,000 transistors in 516 square mils, and is fabricated on Intel's one-micron CHMOS-IV process technology.

cache-control logic on the same chip as the microprocessor. Conventional designs use different integrated circuits for the microprocessor and cache control. Employing the cache-control logic on the same chip allows the SL SuperSet to employ a control algorithm that prevents unnecessary accesses to the main memory or the expansion backplane. Consequently, the entire cache operation becomes even more efficient.

The 82360SL Peripheral Subsystem

As the name suggests and Figure 4 shows, the 82360 SL Peripheral Subsystem incorporates the control circuitry for the serial ports and a parallel port. Moreover, the chip provides control signals and logic for the external floppy disk drive controller and the IDE hard disk drive interface. Along with that circuitry, the Peripheral Subsystem contains support circuitry for input/output operations. This circuitry, which maintains synchronization between the microprocessor and the peripherals, consists of two timer/counters, two programmable interrupt controllers, two direct memory access controllers and various registers. To maintain compatibility with existing systems, all the small pieces of the subsystem exhibit standard design specifications. Also, the 82360 includes a system clock which has a small block of random-access memory. While the clock retains the current time

Flexibility — found through programmable options such as hardware timers, event monitors and input/output interfaces — allows control of system activity.

Firmware on the 82360 SL manages the power needs of any given system configuration. This allows the 82360 SL to control the System Management Interrupt operation of the microprocessor. Rather than allow software or peripheral devices to control power usage, the microprocessor oversees usage. Furthermore, the power management portion of the system independently works on top of the standard system resources. With the SMI operation taking space in a small area of allocated RAM, the microprocessor can enter a power management mode that stays transparent to the operating system and application software.

Effectively, the SMI operation allows the system to handle multiple suspend and resume requests without bothering the operating system or application software. Because the microprocessor can enter the SMI operation from any operating point, the previous state of the CPU becomes saved when it enters the "rest" state. When the microcomputer resumes its operation, the SMI operation restores the microprocessor to the previous state and the application software continues to operate.

How the SL SuperSet Works to Save Power

and day, the memory maintains system configuration settings.

Power Management With the 82360 SL

While the high level of integration found within the 82360 SL conserves power, other functions also aid power management. A new option of the SL SuperSet saves battery life while guaranteeing compatibility between software packages.

Suspend. As they work together to conserve power, the 80386 SL and 82360 SL support two different types of suspend/resume options. If the working environment dictates the need for a fully-functional suspend/resume state, the system can enter a low-power state. All control signals and the power supply voltages remain active. Data contained within the random-access memory stays refreshed through the suspend/refresh function of the System Management Interrupt. Components such as the 80386 SL, the 82360 SL, the power regulator and optional memory continue to receive a constant five volts. Given the resume signal, the system can transparently go back to its normal operating state from the original software application.

The other suspend/resume option is the lower-power suspend/resume condition. Here, the power supply regulator becomes turned off. Only the 3-volt battery which maintains the system memory remains powered up. Since all configuration registers within the 82360 SL go to their default state, a suspend system management code retains the register settings for the resume state. With the presence of the resume signal, the system management code reconfigures the microprocessor and the peripheral sub-system and then executes the resume instruction.

StandBy. Aside from the two suspend/resume functions, standby options also exist with the SL SuperSet. Leaving the microcomputer idle for a given length of time will send the system into a global standby. A programmable global standby timer sets the length of time needed to initiate the global standby state. A time out issued by the timer generates a system management interrupt. Then, system management code turns off the video display, the disk drives and stops the CPU clock. The global standby option saves up to 60 percent of the total system power.

Another standby provision allows the system to shut down any unused local device, such as a disk drive. Idle device timers within the 82360 SL can control as many as six attached devices. When the idle device timers see no activity from their respective device, a timeout happens. Working like the global standby, the local standby generates a system management interrupt code which both saves the status and eliminates the power of the device. Shutting down the unused device saves normally wasted power.

By implementing the SL SuperSet design, Intel and Zenith Data Systems satisfy a growing group of users. As we have seen, Intel has refined already-attempted functions into a more efficient package. Portable computer users have asked for compatibility and reliability in a small package. Desktop computing power is at the fingertips of the portable computer user.

Continued on Page 20

Batch Files

What, How, Why, Who and When

John Day
5 rue Sauer
77500 Chelles, France

Recent discussions on batch files in REMark have agreed that they're a Good Thing, but haven't gone much into how to build them. Since I use them extensively, I feel I'm well-placed to give a fairly complete run-down on batching with DOS 3. Examples were checked under 3.21. The full keystroke program given later on is in regular production on an assortment of machines running 3.21 and 3.3+. Zenith Data Systems have never released DOS 4.0 in France, and this article was written just before 4.01 was made available. I just had time to revise this article to include 4.01 differences, and the final Basic example has been modified so that it will run with all these versions of DOS.

This article starts with a general description of batch files, with some undocumented technical information. "Undocumented" means it works, but Microsoft (bless their little cotton socks) reserved the right to change anything they like whenever they like and without warning anyone beforehand - like adding six bytes to the COMMAND.COM batch control block between DOS 3.2 and DOS 3.3+. (Luckily, they usually keep things the way they are.) The article then goes on to give detailed guidance, with examples, on building high-performance batch files, followed by sample code for support programs. Finally, I've grouped some technical definitions at the end; if you're an old hand, you won't need them.

What is a batch file is easy. It's just an ordinary text file, such as you handle with EDLIN or a word processor in program mode, containing a series of ordinary DOS commands. It *must* have the extension .BAT. Typing *myfile* at the DOS prompt will set DOS to executing all the commands in the file MYFILE.BAT. You also have some rather rudimentary programming devices. Labels let you skip forward or loop back. An *IF* instruction lets you test parameters, the

existence of a file, environment variables or the completion of a preceding step. Parameters given on the initial DOS command line, and also environment variables, can be included in batch commands. And that's about your lot. Don't look for anything fancy, it ain't there.

How batch files work is ingenious and simple. When you start a batch file, COMMAND.COM reserves a small block of memory, not more than 192 bytes. This contains (undocumented information):

- DOS 3.21: a zero byte, use unknown to me, or DOS 3.3+ and 4.01: 7 bytes, 00 01 00 00 00 00 00
- a word pointer to the first byte of the next instruction to execute in the batch file
- a zero word
- 10 word pointers to the first ten parameters in the command line, which is further on in the block; null parameters are marked by FF FF in the corresponding pointer
- an ASCIIZ string with the full name (drive+directory+name+extension) of the batch file, which always starts at byte 19h (3.21) or 1Fh (3.3+ and 4.01)
- a second ASCIIZ string, containing the initial command line but with each separator(s) replaced by a single CR character 0Dh.

Note that multiple separators - like *batch par1,,par3* - are not taken into account. In this example, *par1* becomes the first argument, "%1", and *par3* the second, "%2". You can define a null argument with quotes - *batch par1,"",par3* - but the second argument will now be "" (two quotes).

To substitute a parameter (%0 to %9) into a command during processing, COMMAND.COM looks at the proper pointer, and reads from the memory block up to the next 0Dh delimiter. The *SHIFT* command moves all the pointers up one, but doesn't change anything in the ASCIIZ

strings. When the batch file calls for a program to be executed, COMMAND.COM closes the file before loading the program. This way, no file handles are taken up at the program's expense, and if the batch file is on a diskette, it can be removed during processing. When the program terminates, COMMAND.COM reopens the file and uses the first pointer to pick up where it left off. If the file was on a diskette and COMMAND.COM can't find it on that drive any more, you will be invited to replace the right diskette. If you have identical batch files on several diskettes, COMMAND.COM isn't fussy. Look out for surprises, though, if you switch to a different batch file with the same name, or if you edit the running batch file - COMMAND.COM will try and start again at the byte where it stopped, which may not correspond to the beginning of a command any more.

Only COMMAND.COM knows where this block of memory is. It's quite easy to find, though, even with Basic - as long as you use certain undocumented DOS features (who'd have guessed?). I'll come back to what you can do with it at the end of the article.

Why you need batch files is a question with several answers. There are three main reasons.

First, you won't have to retype commands several times because of typing errors. Once you've got your batch file right, it stays that way.

Secondly, a batch file never forgets. Much software requires several DOS commands to start, such as associated TSR programs to load (and unload afterwards), *PATH* statements, and *SET* commands like the *set nosnowcontrol=true* you use if you still have WORD 4 and a CGA monitor. Once you've got everything into your batch file in the right order, it will be repeated unflinchingly.

Third, the batch file is fast. Starting something up by hand may entail switching the default drive, then switching the default unit, then calling the program. A batch file can contain all this, and be called up with two to nine keystrokes. It can even be worthwhile to write a batch file to contain a single DOS command, if one of your programs has a lot of command line arguments to give but you always use the same options.

It's nearly impossible to start up a desktop without the AUTOEXEC.BAT initial batch file. Once you're running, you should have batch files to start your most-used software; only start something by hand if it takes a single command, or if you use different command arguments each time. Any group of DOS commands which you use regularly could be combined into a single batch file. As an example: when a seminar at the *Institut Control Data* is over, all the disk files on the students' workstations must be put back in shape. Students may modify the start-up procedures as part of their training, and are quite likely to tailor software defaults to their own liking. This must be set to rights, so that for the next seminar all machines start up with identical prompts, screen colors and running modes. Instead of checking file by file, I run a (hidden) batch procedure which clears the student data subdirectory, and restores key files - AUTOEXEC.BAT, CONFIG.SYS and the xxx.INI files for software start-up configurations - from a hidden directory. What used to take ten minutes per desktop can now be done in about five minutes for a whole classroom. If you're doing development and test work, think about automating the test cycle with a batch file for compiling-and-linking-and-testing your program in one lump. If clumsy people are using your desktop, use batch files to trap commands you don't want them to use unsupervised; a favorite used to be *FORMAT.BAT* to replace *FORMAT.COM*. The batch file would check that the dear user wasn't trying to reformat your hard disk, before forwarding the command to the real format program. DOS 3 disk protection has made this unnecessary now, but there are other similar traps you may need to lay.

There are drawbacks, of course. Typical batch files run to one or two hundred bytes. Since files are allocated in blocks of up to 4K, this means a lot of wasted space. Also, batch files as Microsoft foresaw them are very uninteractive. The only DOS user response provided is the *PAUSE* command, which lets you either continue or bomb out. The *ERRORLEVEL* (which is an exit code, not necessarily an error condition) can be tested within a batch file, but DOS has only a handful of programs, such as *BACKUP* and *RESTORE*, which use this code - mostly programs introduced from DOS 2 onwards. Don't be downhearted; a four-instruction *.COM* program can con-

vert a keystroke to its ASCII equivalent and feed it back to the batch program as an exit code. An even better - albeit longer - program is given at the end of this article, to convert pairs of keystrokes to arbitrary exit codes.

Who writes these batch files, and when?

Well, much of the time, it's you. You know what you need most in the way of shortcuts, so you just copy your most-used command sequences to a disk file. You will often receive batch start-up files with software, but don't take them as gospel, as they can usually be improved upon. An example is the French LCECOM telecoms program, which starts up by displaying a lovely screen logo with a background that scintillates for about thirty seconds. The fourth time you've watched it, you reach for a line editor in order to delete the call to the *LOGO* program. The only batch files I would hesitate to change are installation files for copying software to disk, as they call up undocumented programs with unexplained parameters. Get it wrong, and the software won't work and maybe your disk won't work any more either.

If you're maintaining systems for other people, you should provide a set of batch programs for everything they need to use, to be sure they are using it properly. Use file attributes to protect anything you don't want clobbered. *Read-only* means that you can see the batch files in directory lists and use them (and even change their names!), but you can't erase them. *Hidden* means that you can't see the batch file in the directory list, but if you know its name you can execute it (From DOS 3 on - DOS 2.0 won't execute a hidden file). I usually put "dangerous" software like *PREP.EXE* and *PART.EXE* in a hidden directory, with a hidden batch file to include the directory in the system search paths if needed. Any good disk maintenance program will let you ferret out the hidden goodies, but any disk maintenance program on a desktop I'm maintaining is hidden too. *ATTRIB* lets you set the read-only attribute, but to hide files and directories you need a disk editor like Norton Utilities or PC-Tools.

A batch possibility which escaped my notice (and that of a lot of other people, too) was included in DOS 2; Zenith Data Systems put the write-up in with the *SET* instruction, which isn't exactly where you'd look for guidance on batch files, and I have found no reference whatsoever to it in my IBM DOS 2.0 handbooks. Admittedly, these last are in French, so it could be a translation slip - or IBM may have drawn a veil so as not to have to fix some bug. It's in the IBM system, even if it's not in the handbook; I checked. The first reference I actually saw was in Robert Brasfield's article on batch files in the April '89 *REMark*. From DOS 3.3+ onwards, you will find the write-up under "Batch processing". What it is is the possibility of using environment variables - your own as well as the system's -

in batch files. Let's look at an example. From the DOS prompt, key in these few lines:

```
C>set editor=Jim
C>echo %editor%
%editor%
C>
```

It didn't work, did it? This is one of the few DOS commands that you can't play with directly to get the feel of it. Now, without changing the *set* instruction, key in:

```
C>copy con test.bat
echo %editor%
^Z          (F6 function key, or Ctrl-Z)
          1 file(s) copied (DOS acknowledgment)
C>test
C>echo Jim
          (the batch file test is now running)
Jim
C>
```

See the difference? When a batch file is running, *%variable_name%* is replaced with the *value* of the environment variable. The biggest use I make of this is lengthening and shortening search paths. I always set an environment variable to my basic path in the AUTOEXEC file:

```
PATH my_starting_search_paths
SET OLDPATH=%PATH%
```

and keep my standard path as short as possible.

Before the *SET* instruction is executed, *%PATH%* is replaced with *my_starting_search_paths*. Now, I can use *%oldpath%* in any batch file. *PATH extra_path;%path%* will substitute the current search path before executing, so the extra path will be added before the existing path - and *PATH %path%;extra_path* would have added it behind. *PATH %oldpath%* restores the standard search path from the AUTOEXEC file. My batch files to start software applications include the proper search directories before calling the software, and remove them at the end of execution. This way, I avoid 12-foot long search paths and the corresponding interference which would be possible between directories. If the *PATH* statement in the AUTOEXEC file is changed, all my batch files will include the new path without my having to alter them, and will reset to the new path on exiting. Note that environment variables are about the one batch device which you can't check out interactively before writing your batch file, because outside of a batch file they just don't work. I'll give an example of changing and restoring paths a bit further on.

A reminder about the *PATH* command, which doesn't work quite like you'd think. From DOS 3 on, you can execute any program from anywhere by giving its full path explicitly:

```
C>e:\util\menu
```

will execute the *menu* program (*menu.com*, *menu.exe* or the batch file *menu.bat*) in directory *e:\util*. If the program is not found, DOS will not search further. It is only if you do *not* give an explicit directory that DOS will search. It starts with the default directory on the indicated drive if you give one

- as "e:menu" - or on the default drive if you do not, then looks in every directory in the standard search path, in order. So, you can always prevent a full directory search by giving an explicit call to the directory you want. "e:\util\menu" will not search further, whereas

```
C>e:
E>cd \util
E>menu
```

will carry out a directory path search if menu is not found in e:util.

```
C>e:menu
```

(DOS 3 and 4) will look in the default directory for e:. It will not look on c:, the default drive, unless c: is included in the search path. It will, however, try all entries in the search path before abandoning.

I'm against rambling path statements that include nearly every directory on the disk. If you are running an application that requires a directory towards the end of a long path chain:

- at best, it will have to sift through all the intermediate directories to find the one it wants; if it's using overlays, you'll notice the extra wait;
- at the worst, it will find some other program with the same name but belonging to a different application or version, and your system will bomb out.

For reliable operation, use only four or five elements in your path, in this order:

1. The directory (or directories) used by your applications software;
2. Your utility directory, with non-DOS utilities and drivers, libraries and batch procedures;
3. DOS, with the release system only - additions go in with utilities;
4. The boot root, usually c:, because some applications can't shell if COMMAND.COM is in a subdirectory.

Now, some examples of batch files. First, a file WHEN to show the date and time. The DATE and TIME commands do the same thing, but with a lot more work - two commands to type, and they will wait for input before proceeding. Batch files are all about saving work, remember? Here is WHEN.BAT:

(for DOS 3.21 and earlier)

```
echo off
prompt $d, $t$_
echo on
echo off
```

(for DOS 3.3 and later)

```
@echo off
Prompt $d, $t
echo on
```

```
echo off
prompt $p $g
```

(deluxe version)

```
@echo off
set xprmt=%prompt%
prompt $d, $t
echo on
```

```
@echo off
prompt %xprmt%
```

```
set xprmt=
```

Sample printout:

```
C:\ >when
Sun 6-16-1991, 8:43:50.12
C:\ >
```

Only include the beginning @echo off (echo off prior to 3.3+) command once the batch procedure is working properly. It mutes screen output, to avoiding filling it with running messages. The procedure changes the system prompt to "\$d, \$t" - that is, "date, time". The blank line elicits a system prompt, which is what we wanted. The procedure then resets the system prompt to my preferred value, \$p \$g - unit_directory space >. The deluxe version saves the current system prompt in the environment space under the name xprmt, then restores it before clearing the value from the environment space (which otherwise gets cluttered up quite fast). Before 3.3, there was no way of muting the command echo off - so the 3.21 procedure displays "echo off" on the screen, preceded by the date, the time and a line feed. The other two give cleaner output. Just type when to display the current date and time, and return immediately to the DOS prompt. Include this code in the AUTOEXEC.BAT for a pc-AT if you want to check the real-time clock at power-on time.

Next, the trap program to limit format to drives a: and b:. I shall suppose, for this and subsequent examples, that there are two subdirectories on drive c:, \SYS with all DOS programs and \UTIL with batch procedures and non-system utilities. The default directory will always be the root, c:., and the standard path will be c:\util;c:\sys;c:.. Put a procedure FORMAT.BAT in c:\util :

```
@echo off
set abortflag=
for %%u in (A: a: B: b:) do if %1.==%%u.
    set abortflag=ok {3}
if %abortflag%.==ok. goto fmt {2}
echo You can't format drive %1
goto end {1}
:fmt
set abortflag=
c:\sys\format %1 %2 %3 %4 %5 %6 %7 %8 %9
:end {1}
```

Explanations (in order of difficulty):

1. goto label (i.e., goto fmt) without executing intervening instructions. You can skip forward or backward; on the system I tested, goto starts by rewinding the batch file and searches for :fmt from the beginning. The label :end lets you skip to the end of the procedure, which returns you to DOS.
2. if text1==text2 command only executes command if text1 and text2 are identical. The two periods are there to avoid bombing out on a null string. COMMAND.COM does parameter substitution in a simple, idiot manner: a) substitute strings, b) look at what you get. Let's try if %abortflag%.==ok goto fmt, without the periods. If abortflag is equal to ok, this gives if ok==ok goto

fmt, which works fine and skips to :fmt. If, however, abortflag is null (not defined), it gives if ==ok goto fmt, which gives a syntax error. The periods give you if .==ok. goto fmt instead, which is syntactically correct and does not skip.

3. for %%%u in (list) do command sets %%%u to each value in list in turn, and executes command for each value. This instruction seems to use an internal pipe; goto label is not accepted in the command field, and blows the instruction with a pipe error. Since if... is case sensitive, you have to test for every reasonable spelling of your target; hence the upper- and lower-case drive letters. The example compares the drive letter in the format drive_letter command (%1) with "A:", "a:", "B:" and "b:". abortflag will only be set to "ok" if one of the comparisons is true.

Since the call to the DOS format program gives an explicit path, DOS will not try the search path if format.com is not found, and will stop with a Bad command or file name message. This is important, because it lets us use the same name - format - for the batch file and the program. As long as you work from the root, DOS will pick up whichever comes first in the search path: in this case, c:\util is searched before c:\sys. Do not use any call which will let DOS try the search path, because in this case it would loop back to the batch file if it didn't find the program, putting you in an endless loop. The parameter string - %1 %2 %3 .. %9 - is there to shove the rest of the command line over to format.com without checking it.

For, if and goto are all you get to structure your procedures - I said at the beginning that DOS batch programming is rudimentary. You can still do a fair bit with what is provided. Let's write a batch file WP.BAT to let an Unskilled Person start up your word processor wp.com in subdirectory \wproc, after switching to the default document subdirectory \docs. Since the Unskilled Person won't understand the message "Bad command or file name" given if wp.com isn't there and will start pressing every key in sight in the hope that the stupid computer will end up by understanding, we will replace the message by something a little more explicit:

```
@echo off
if exist c:\wproc\wp.com goto wpok
echo The word processor is not available
goto end
:wpok
c:
path c:\wproc;%path%
cd \docs
\wproc\wp
c:
cd \
path %oldpath%
:end
```

If c:\wproc\wp.com exists, the if instruction executes the goto wpok, and the procedure sets up your directories and paths, executes the word processor, then

puts everything back like it was and returns you to the root directory. Otherwise, you get a polite message and the procedure quits.

There are two more possibilities I've never had the occasion to try: wildcard characters in the *if file* filename command, and also in the *for* command. Look at the official DOS write-up for examples. You may negate tests with *not*, but you can't combine conditions or do anything more complicated. You also have the *pause message* command, which stops processing. This is mostly used to give you a chance to break out of a procedure with Ctrl-C if something has obviously Gone Wrong. Finally, I'll cover *if errorlevel . . .* under interactive batch processing.

How to start a commercial program depends very much on how well it's written. Some programs (Microsoft software in particular, *noblesse oblige*) can be started anyhow from anywhere. I've never managed to start dBASE III+ except from its own subdirectory (although one of my colleagues has had some success using the *SUBST* command). Other software lies between the two. The problem lies in the environment space. Each program has its own copy of the system environment, containing: one ASCII string for each environment variable, an extra byte of zeros, a couple of odd bytes, and a further ASCII string with the full name of the program - unit+directory+name+extension. Any program can access its environment copy, using a pointer at 2Ch in its PSP, to find out what unit and directory it was loaded from, and what search path has been declared. It can then load all its support modules from the same subdirectory it was loaded from itself. "Can", however, does not mean "will". Some commercial software has obviously never heard of the environment copy. You will find four categories of software, which require different handling:

- Runs anywhere without PATH statements (such as WORKS, WORD 5) and can be moved around as necessary (such as the Microsoft lessons LEARN) - but beware of copy protections, which may not move properly;
- Runs without PATH statements, but only where it was installed - such as WINDOWS 3, which has full unit/path/program designations in the Program Manager and can't be copied to a different directory;
- Only runs if a PATH statement is present or if it is started from its own directory - such as WORDSTAR 2000;
- Can only be run from its own directory, such as dBASE III plus.

The only way to find out a program's category is to try it. Install it in its own directory, be sure the install program hasn't added a path statement, and switch to the root or to some other directory. Call the program with an explicit command *unit\directory\program*. See if it starts up,

and try a few functions like spelling checks or graphs. If everything works, it's a well-written class A or a class B - you can find out which by copying it to another unit or directory and trying again.

If the program doesn't run like this, type in a new path statement including the program's directory, and try again - success means class C. If it still doesn't work, then it's class D and you will have to switch to the program directory in your batch file before running it.

Extra considerations, however, may lead you to add PATHs or change to the software's own directory, even if it will run without. A lot of software lets you shell out to DOS, such as with WORD 4.x/5.0's Library/Run instruction. But if you want to do this to access Microsoft utility programs like *RONLY*, *RWRITE* or *MSD*, DOS will require a PATH statement to access the WORD directory. Similarly, many MicroSoft LEARN programs keep a progress file, which is always in the default directory. I would therefore suggest adding a path statement if the software subdirectory contains utility programs you can shell to, and switching to a consistent subdirectory for teaching programs which don't access user data, but keep a course profile on disk. And if you're still under DOS 2, you can't give a path in the command line to start a program, and you'll have to use a lot more PATH statements than under DOS 3 or 4. If you have to start up from the application's subdirectory, you will most likely mix up your own files with the release software, and you'll have the devil's own job spring-cleaning without deleting application files by mistake. To sum up:

Types A and B: switch to the subdirectory where you keep your data, and call the software with an explicit *unit\directory\name* command. Only use a path statement if you intend to shell out to utilities in the software subdirectory.

Type C: switch to your data subdirectory, add a path statement and call the software.

Type D: switch to your program subdirectory and call the software. Hope that a *CONFIG* file will let you set your data subdirectory.

Sample batch files to start up with default data on diskette A:

A,B

```
@echo off
a:
c:\wproc\wp
cd \
```

C

```
@echo off
path c:\wproc\;%path%
a:
c:\wproc\wp
c:
path %oldpath%
cd \
```

D

```
@echo off
c:
cd \wproc
wp
c:
cd \
```

The first two examples won't run with DOS 2, because *c:\wproc\wp* is only accepted by DOS 3 and 4. For the third example, you will note that you can't default data to a separate drive/directory, so you must switch as best you can once the software is running.

This sums up what you can do with DOS as it's delivered. To go interactive, you must add programs to feed keyboard input through to the batch processor. The hoary old ancestor of these programs takes less than a minute to create with *DEBUG*. I'm supposing you have no program called *KEY.COM*; then type

```
C>debug key.com
File not found
-a
xxxx.0100 mov AH,07
xxxx.0102 int 21
xxxx.0104 mov AH,4C
xxxx.0106 int 21
xxxx.0108
-r cx
CX 0000
:8
-w
Writing 0008 bytes
-q
C>
```

The "File not found" message confirms that you are creating a new file *KEY.COM*. If you proceed without getting this message, you will overwrite and destroy an existing file. "-", ".", and "xxx.nnnn" are *DEBUG* prompts. *xxx* can be any value, depending on your configuration. Register *CX* contains the length of your program. You must set this yourself before the "w", or write instruction; the length is confirmed in the "writing 0008 bytes" message. The first *int 21* system call, with 07 in register *AH*, waits for a keystroke and returns the character in register *AL*. The second call, with 4C in *AH*, exits and sets *errorlevel* to the value in *AL*.

To use *KEY*, you write a batch file that calls *KEY*, tests the exit code, and acts accordingly. I'll show a sample batch file to call a spreadsheet "sp" if you type "S", a word processor "wp" if you type "W", and exits to DOS if you type "O" (which I like as a code for exiting, because it's just under my right pinkie). ASCII codes needed: 0 - 48, S - 83, W - 87, s - 115, w - 119. And the catch with the *if errorlevel nnn . . .* batch command: this is implicitly interpreted as "if errorlevel => nnn" (greater or equal), and you can't have it any other way. This means that you must test possible exit codes in decreasing order, and you must test for invalid codes - *if errorlevel 83 . . .* tests for "S" or any other character with an ASCII code greater than 83. You would need a little welcome message to remind you what options are available; I've supposed this a separate file *WELCOME.TXT*.

That's another 2K or 4K on disk, but the type command is still a bit faster than echo for multi-line output, even on a pc-AT hard disk, and lets you include blank lines to improve presentation - echo with nothing else on the line doesn't give you a blank line, it displays the current status of the echo flag.

```
@echo off
:display
cls
type c:\util\welcome.txt
:waitkey
c:
cd \
key
if errorlevel 120 goto waitkey
if errorlevel 119 goto wordp
if errorlevel 116 goto waitkey
if errorlevel 115 goto spread
if errorlevel 88 goto waitkey
if errorlevel 87 goto wordp
if errorlevel 84 goto waitkey
if errorlevel 83 goto spread
if errorlevel 49 goto waitkey
if errorlevel 48 goto end
goto waitkey
:wordp
cd \docs
\wproc\wp
goto display
:spread
cd \spsheet
\sproc\sp
goto display
:end
echo Exiting to DOS...
ver
```

This works, although the string of *if* commands is quite horrible. You can also add checks that the software is installed, as in one of the previous examples, and extra key reads to choose between several default startup units. Problems, though: (minor) there's no screen echo, because invalid keystrokes would stay on the screen and clutter it up; (fairly minor) you can also call the word processor with Shift-F4 or Ctrl-Home, the spreadsheet with Del or Ctrl-left, and exit with Alt-B; (medium) separate tests are required for lower- and upper-case; and (major) when a desktop for seminars is equipped with WordStar Pro, WordStar 2000, Word 5.0, Word 5.5, Works, Windows 2.11, Windows 3, WinWord and WordPerfect, you wonder how to manage with only one W in the alphabet.

Pat Swayne gave an improved program "OPTION.COM" in his article on menu systems in the April '88 REMark. It handles upper- and lower case, but requires your (single-key) options to be consecutive characters, like A through E or 1 through 5. It doesn't help any for mnemonic choices for the "W" forest; I want codes like "wo", "w2", "w5", "w+", "wk", "wi", "w3", "ww" and "wp" for the above-listed programs. The answer I came up with, and which satisfies me, is a rather longer program, which can still be keyed in with DEBUG (and was). The program waits for two keystrokes, compares them with a list in memory, and returns a user-supplied exit code for the pair - so I can have any exit codes I wish for my software abbreviations.

Invalid keystrokes are intercepted by the program which just loops back, so the batch file needs only to test for valid codes. Keystrokes are echoed to the screen; the program first echos a CR code, so the characters are always in columns 1 and 2. The first character is checked immediately; if no abbreviation corresponds, the program loops back, if it's a one-character abbreviation the program exits straight away (after echoing an extra space as acknowledgment), and for a two-character abbreviation it waits for the second keystroke. All characters are mapped to lower-case or numerics, which gives proper handling of function and direction keys - Ctrl-Home is decoded as "7", and not "w". It is given as the first sample program; don't key in the comments (everything after a ;), as DEBUG can't handle them (you can, though).

"a" is the DEBUG assemble directive, and DEBUG assembles until you give it a blank line. "f" is the fill directive. *f cs:168 1200 FF* fills two tables, each 100h bytes long and starting at 168h and 268h, with FFh (-1). *f cs:368 4FF 00* fills a table running from 368h to 4FFh with zeros. Note that the *DB* instructions which follow load values into parts of these tables, overlaying the background fill. All the assembler instructions were explained in Pat Swayne's 1990 series of articles "Getting Started With . . . Assembly Language." *XLAT*, however, only got eight lines (in February), so I'd like to explain it a bit more. It's a very fast instruction for translating between two code sets. A 256-byte table in memory contains the codes you want to obtain. *XLAT* uses the value in register AL as an index to the table, takes the byte it finds in the table and puts it in AL in place of the former value. *XLAT* has no arguments; *BX* contains the address of the table, and the byte to recode must be in AL. Watch what the program will do if you key in ")", which is a shifted 0. The instruction at 0106 sets *BX* to point to the table at 0168. The *INT 21* at 010B reads the keyboard, and returns 29h, the ASCII code for ")". The *XLAT* then looks up the table at 0168 + 29, which makes 0191. The *DB* instruction assembled at 0189 put 30h, the ASCII code for 0, at this location; so the ") becomes a "0", as though the shift key hadn't been pressed. Not very important for a US keyboard, but essential for the French keyboards I always use, as on these you have to shift in order to get numerics.

And what if you type the numeric pad "0" without shifting, which gives "Ins"? This code requires two read interrupts, as it occupies two bytes - 00h 52h. The test for zero at 010D/010F detects the double-byte coding, and loops back to read the second byte after having set *BX* to the alternate table at 0268. The *XLAT* will now use address 0268 + 52 = 02BA, which contains, once more, the ASCII code for 0, 30h. This works for all digits except 5, which isn't decoded when the pad is not

shifted.

You can see now why the 8-byte KEY.COM program gave odd results. "Del" is coded 00h 53h. The program picked up the first half and returned with an exit code of 0 for the batch procedure, which then looped back and called KEY.COM again. The second call got the other half of the code, 53h, which happens to be the ASCII code for "S". The big KEY.COM ignores all double codes - function keys, Alt-combinations and cursor commands - unless they are explicitly included in the alternate *XLAT* table.

And what if you type a real "0", code 30h? Well, the *XLAT* instruction will go to 0168 + 30 = 0198, and locations 0198 through 01A1 contain the digits 0 to 9, codes 30h to 39h. So the 30h in AL is replaced with 30h, which just keeps the "0" where it is. Waste of time? No. Coding tests to decide whether to convert a character code or to keep it as it is would take up a lot of code, and testing would take some time. *XLAT* is *f a s t*, 11 clock cycles on an 8088 if you're interested, and coding to avoid unnecessary *XLAT*s is much less efficient than bunging everything through the mangle. The same considerations apply to the alphabet; upper-case is converted to lower, and lower-case is converted to itself.

The last *DB* instruction defines the abbreviations KEY.COM will recognize. Each entry takes three bytes, and since the table is 407 bytes long (to just fill two disk sectors with the entire program, I like being tidy . . .), you can have up to 135 of them, plus the byte of zeros that marks the end of the table. For a single character "c" returning a code "r", code *DB "c" 00 r*. For two characters "c" and "d" returning code "r", code *DB "cd" r*. The example given will return with code 0 if you type "0", 1 if you type "ss", and 2 if you type "wp", and will reset to beginning-of-line for any other keys. For "0", read "any key with a 0 on it, whether the keyboard is properly shifted or not", and the same goes for the other codes. This would give us the following batch program to call our spreadsheet or our word processor or exit (the smudge in the line *echo .* is meant to be there, ASCII character 250 is unobtrusive and lets you echo a nearly-blank line):

```
@echo off
:display
cls
type c:\util\welcome.txt
c:
cd \
key
if errorlevel 2 goto wordp
if errorlevel 1 goto spread
goto end
:wordp
cd \docs
\wproc\wp
goto display
:spread
cd \spsheet
\sproc\sp
goto display
:end
echo .
```

```
echo Exiting to DOS...
ver
```

That's much simpler than the previous batch example, isn't it? You can customize KEY.COM by including just the key combinations you want to recognize, together with the exit codes you want, and then write the batch file to go with it. In the production versions I use, I set up the first few digits returning their natural codes - DB "0" 00 00 "1" 00 01 "2" 00 02 ..., a few drive letters returning codes from 10 up - DB "a:" 0A "b:" 0B "c:" 0C ..., and then the abbreviations for whatever software is on the desktop, using exit codes from 16 (10h) onwards. Don't forget that if you want to use special characters like the colon, you must include them in the XLAT table. I'm not giving a complete production example, because it is quite lengthy and is customized to my software layout. I can propose a useful sequence, though, to start up a word processor from any drive following a prompt, and selecting a default drive c:\docs if drive c: is requested. I include the root directory on drive d:, to show how you must get the selections in the right order. If you get a message "Out of environment space", increase the space to about 300 bytes with the shell statement in CONFIG.SYS (for DOS 3.2 and later - sorry 'bout that).

```
:wproc
echo .
echo a: - diskette a:
echo b: - diskette b:
echo c: - directory c:\docs
echo d: - hard drive d:
:wloop
key
if errorlevel 14 goto wloop
set subdir=\
set drive=d:
if errorlevel 13 goto wload
set subdir=\docs
set drive=c:
if errorlevel 12 goto wload
set subdir=\
set drive=b:
if errorlevel 11 goto wload
set drive=a:
if errorlevel 10 goto wload
goto wloop
:wload
%drive%
cd %subdir%
c:\wproc\wp
goto...
```

[loop to the start of the batch file]

Commercially available menu programs are more snazzy, but are slower and take up more space. The Novell Netware menu system, for example, has pop-ups with multiple color palettes, a tree menu structure, and the possibility of keying command-line parameters before executing a batch sequence. It also takes 14 Kb as a TSR program, whereas mine takes about 60 bytes for the batch control block. Mine's also faster. Commercial menu programs use scripts containing all the options; when you select an application, the program reads the script file to find your choice, copies it a temporary batch file, then hands the temp file over to COMMAND.COM. That's three disk operations, whereas the

KEY.COM program does it in one.

The only thing you don't get with KEY.COM and batch menus is the possibility of reading parameters from the keyboard and passing them to the batch file. Commercial menu programs read your variables as you choose your standard script, then insert these values into the text as it is copied from the script file to the temporary batch file. There is no Microsoft-approved way of moving values directly to the running batch parameters, but ... COMMAND.COM doesn't mind in the least if a program hooks into the batch control block and changes values. It just keeps going, using the new values it finds. If you wish to do this, you must respect the pointer structure, and you must never spill over the end of the allocated block, as this will freeze the system instantly. Use a dummy batch file to call the real menu file with a giant parameter line:

menu.bat:

```
@echo off
menuz abcdef.....mnopq
```

menuz.bat:

```
@echo off
cls
:display
type ...
```

and the rest of your batch menu

The giant parameter can run to about 120 characters, and will force COMMAND.COM to reserve 11 or 12 paragraphs for the control block, which is about the maximum. This will then let you plug in any values you like, up to a maximum of 120 bytes.

The final program I give here is not a production version. I have tested hooking in to the batch parameters with an interpreted Basic program, as most people can read Basic easily. If you want to try it, don't forget to start Basic from a batch file with a big parameter list, otherwise it will detect that the batch control block is missing or too short. The program shows how to break in to the memory allocation chain, locate the batch control block, and modify values. You should convert this to your own programming language as an .EXE or (better) .COM file, and use the command line (COMMAND\$ in compiled Basic) to pass better prompt messages than the example's "Parameter 1:", "Parameter 2:" and so on. COMMAND.COM will not embed a null parameter in a string of values - batch a,c gives "a" as parameter 1 and "c" as parameter 2. The sample program imitates this and stops on the first null, but my tests show that you can put a null pointer in the middle of pointers to real values - pointer_1 null pointer_3 - without upsetting the processor.

The program has been tested, and picks up the memory block reliably. However, it will only pick up the primary copy of COMMAND.COM; if you shell to a secondary copy of COMMAND.COM, I know

of no way to get into it. This method is based on the method given in Robert G. Brasfield's article "CRTSAVER revisited", REMark March '88. I have improved identification of the memory chain, because we are looking first for the COMMAND.COM PSP. The memory control block for the PSP must be in the paragraph immediately preceding the PSP, which makes it easy to find. Also, I prefer to start somewhere inside COMMAND.COM and scan backward through memory, whereas Robert Brasfield scanned forward.

System interrupt 2E is hooked to COMMAND.COM, so if you pick up the segment address at 0:00BA, you are in COMMAND.COM somewhere - with luck, you will be bang on the PSP. You can identify a PSP because:

- it starts with CD 20
- the previous paragraph is an MCB, starting with 4D (exceptionally 5A)
- the MCB owner (bytes 1 and 2) points to the PSP.

Once you've got the right PSP, you can walk the memory chain forward. The usual contents under DOS 3 when a batch file is running are:

- DOS, including buffers and device drivers
- COMMAND.COM PSP and code
- System environment space (owner: COMMAND.COM)
- A very short unused block (owner = 00 00), was for AUTOEXEC.BAT at boot time.
- Two entries (environment copy, PSP and code) for each TSR
- Batch control block (owner: COMMAND.COM)
- Environment copy for the running program
- PSP and code for the running program
- Any memory blocks reserved by the running program.

(You will only pick up the block for buffers and drivers if you use Vern Reisenleiter's method, see next paragraph). Now, from the COMMAND.COM PSP you should chain forward to the second following block belonging to COMMAND.COM. If this isn't the batch control block, Microsoft (bless their little cotton socks) has changed something without warning. If it is, you're in business. The sample code takes a possible SHIFT command into account, and restores the first and second pointers to the start of the existing batch command line. After that, you can change parameters %1 through %9 to whatever you want.

The alternate way of hooking into the memory pointer chain is via interrupt 21h, function 52h, as described in the letter from Vern Reisenleiter in the April '89 REMark, "CRTSAVER revisited". In this case, you would walk the chain until you find a PSP which englobes the COMMAND.COM pointer at 0:00BA - that is, MCB segment less than the pointer, and

MCB + block length + 1 greater than the pointer. After that, just walk forward to the second following COMMAND.COM block as previously.

The program works with DOS 3.21 and DOS 3.3+, because it checks the block to see which structure is being used. 3.21 has the pointer to %0 at 05h/06h, whereas 3.3+ has this same pointer at 0Bh/0Ch, and 05h/06h contains zeros. This should be reliable for distinguishing between the two. *OFST* in the program is set to the pointer to the next command to execute, and from there on the two structures are the same. With each new version of DOS, it will be necessary to hook in to the MCB chain with *DEBUG*, look at the structure of the block, and modify the program each time Microsoft decides to do something different.

DOS 4 needs an extra step. The memory allocation strategy has been changed from DOS 3, and there is now an available block freed up between the DOS kernel and COMMAND.COM. On the system I tested, over 200 bytes were available, and all TSR programs were loaded from CONFIG.SYS, which meant that the batch control block was certain to be allocated below COMMAND.COM and not above it. By following the allocation chain, you will be going away from the block you are looking for. Unfortunately, you can only follow the allocation chain forward, not backward. The block we are looking for, however, is certain to be not very far from COMMAND.COM going towards low memory. It can be found reliably by searching backward from the COMMAND.COM PSP, paragraph by paragraph, looking for the block signature — 4D followed by the PSP address. There's not much chance of finding the same three bytes in random code, but an extra check can be made by verifying that the candidate block does indeed chain forward to another allocation block with a 4D signature. All this is done by the subroutine called from line 180 in the sample BASIC program. If only two blocks are found by walking forward — "endchain" indicator true — the subroutine searches backward from the COMPSP marker. If it finds a third block, it clears the "endchain" flag and processing continues.

I have tested a compiled version of the Basic program using a section of batch procedure for formatting a diskette. I've supposed a system with two diskettes, 5¼" on drive a: and 3½" on drive b:. A batch procedure can ask for the drive and capacity, low density calling for parameters /4 on drive a: and /n:9 on drive b: (DOS 4.01 requirements are slightly different). Other options are too complicated to check with a batch file, so the program BPARAM reads a list of options (such as /S, /V and so on), and plugs it into %1. The parameters to BPARAM are the line to use on the screen (the tenth, here, but depends on what went before) and the prompts to give the user -

the program will accept several values, but one is enough here and covers any number of values for *format*. They will all be lumped together into one value for COMMAND.COM, including any spaces. The tests were done using the PDS 7.0 Basic compiler, and the Basic example in this article modified to pick up and handle the command line parameters (which pads it out to 250 lines, too long to include in this article as it will only interest people with 7.x PDS Basic). The major disadvantages are the size of a Basic stand-alone .EXE program, 32½ Kb for this one, and the automatic conversion of the command line to upper-case, which I don't like. MASM or C will give much more polished programs. Loading is reasonably fast; on my 8 MHz ZW-148 I timed less than 2 sec. from leaving the KEY.COM program to getting the BPARAM prompt, which is reasonable. More elderly Basic compilers will be slower, and may insist on clearing the screen before executing. If you implement this sequence in one of your batch procedures, think about adding IS.COM (REMark March '89, "Is This Disk Formatted", Pat Swayne) to warn you if you are formatting a diskette that already has information on it. As before, *echo Alt-250* lines are included to give line feeds where necessary. *set* being much faster than *goto*, you will find multiple sets with *ifs*, to avoid using *gotos* where I can. KEY.COM, in this example, returns 16 for "HI" and 17 for "LO".

```
:format
echo .
echo format diskette a: or b?:
:formdsk
key
if errorlevel 12 goto formdsk
set drive=b:
if errorlevel 11 goto formdsx
set drive=a:
if errorlevel 10 goto formdsx
goto formdsk
:formdsx
set dens=
echo .
echo capacity - hi or lo?
:formcap
key
if errorlevel 18 goto formcap
if errorlevel 17 goto formlo
if errorlevel 16 goto formhi
goto formcap
:formlo
set dens=/4
:formhi
if %drive%.==b:. set dens=/n:9
bparam 10 "OTHER PARAMETERS?"
format %drive% %dens% %1
set drive=
set dens=
```

That's it. To sum up, batch files can lighten your typing load, decrease errors, let you hand your desktop over to Unskilled Persons without risking a nervous breakdown, and brighten your life. The only disadvantage is that once detailed start-up sequences are coded in your batch files, you're going to forget them, and you will become unable to use your software without invoking your batch procedures. Standard Microsoft instructions, once you've got them well in hand, let you do a

lot. Programs to convert keystrokes to exit codes, following Microsoft conventions, let you do a lot more. Undocumented DOS features let you do almost anything. All this is with minimum overhead, about 200 bytes in low memory.

Technical Terms Used in This Article

ASCIIZ — null-terminated ASCII string, a string of characters with a zero byte to mark the last character.

Environment space — a memory block used only by COMMAND.COM, containing ASCII strings. COMSPEC=command processor and PATH=search path are always included. Any string you like can be included with the command SET NAME=VALUE - no extra spaces, please, and NAME will be converted to capitals. Use the command SET without arguments to see how DOS has mangled your intentions. If you want to use the environment space, it's too small. Increase its size from 128 bytes with *shell = command.com /p /e:nnn* in your CONFIG.SYS file; *nnn* (documented under "COMMAND" and not under "SHELL") is the requested size in paragraphs (early DOS 3) or words (DOS 3.21 onward). Don't forget the parameter /p, which makes COMMAND.COM ignore any EXIT commands it may receive. Each program has a copy of the environment space, whose size depends on what strings were defined when the program was called.

MCB — Memory Control Block, undocumented. It occupies a whole paragraph, but only uses the first 5 bytes. Every block of memory is preceded by an MCB, and these are chained right through memory. The first byte is either 4D, or 5A for the last MCB in the chain. Second and third bytes give the segment address for the PSP of the owner program. Fourth and fifth give the number of paragraphs in the following block; add 1 when you're walking the chain, to allow for the size of the MCB itself.

Paragraph — 16 bytes, at addresses xxx0 to xxxF. Segments and memory blocks always start on a paragraph boundary, because of the way the 8088 and 80x86 address memory.

PSP — Program Segment Prefix, 256 bytes tacked on to the beginning of every executable program when it is loaded. It always starts with an INT 20h instruction (CD 20), bytes 2C and 2D contain the segment address for the program's copy of the DOS environment, and the second half contains the command line used to start the program from the DOS prompt.

Segment address — the paragraph you start at, i.e., the byte address divided by 16. The microprocessor puts the segment address in a segment register, then uses byte addresses relative to the starting paragraph. The BASIC DEF SEG = lets you set a segment register for PEEK instructions.

Sample DEBUG Code for KEY.COM

C:\debug key.com

```

-a
xxxx:0100 MOV DL,0D
xxxx:0102 MOV AH,06
xxxx:0104 INT 21
xxxx:0106 MOV BX,168
xxxx:0109 MOV AH,07
xxxx:010B INT 21
xxxx:010D CMP AL,0
xxxx:010F JNE 116
xxxx:0111 MOV BX,268
xxxx:0114 JMP 109
xxxx:0116 XLAT
xxxx:0117 MOV SI,368
xxxx:011A CMP BYTE PTR [SI],0
xxxx:011D JE 106
xxxx:011F CMP [SI],AL
xxxx:0121 JE 128
xxxx:0123 ADD SI,3
xxxx:0126 JMP 11A
xxxx:0128 MOV DL,AL
xxxx:012A MOV AH,06
xxxx:012C INT 21
xxxx:012E CMP BYTE PTR [SI+1],0
xxxx:0132 JNE 138
xxxx:0134 MOV DL,20
xxxx:0136 JMP 15D
xxxx:0138 MOV BX,168
xxxx:013B MOV AH,07
xxxx:013D INT 21
xxxx:013F CMP AL,0
xxxx:0141 JNE 148
xxxx:0143 MOV BX,268
xxxx:0146 JMP 13B
xxxx:0148 XLAT
xxxx:0149 MOV AH,AL
xxxx:014B MOV AL,DL
xxxx:014D CMP BYTE PTR [SI],0
xxxx:0150 JE 100
xxxx:0152 CMP [SI],AX
xxxx:0154 JE 15B
xxxx:0156 ADD SI,3
xxxx:0159 JMP 14D
xxxx:015B MOV DL,AH
xxxx:015D MOV AH,06
xxxx:015F INT 21
xxxx:0161 MOV AL,[SI+2]
xxxx:0164 MOV AH,4C
xxxx:0166 INT 21
xxxx:0168
-f cs:168 1200 FF
-a cs:368 4FF 00
-a cs:189
xxxx:0189 DB "1" FF "3457" FF "908"
xxxx:0193
-a cs:198
xxxx:0198 DB "0123456789:."
xxxx:01A4
-a cs:1A8
xxxx:01A8 DB "2abcdefghijklmnopqrstuvwxyz"; @ + upper-case
xxxx:01C3
-a cs:1C6
xxxx:01C6 DB "6" FF FF "abcdefghijklmnopqrstuvwxyz"
xxxx:01E3

```

```

-a cs:2AF
xxxx:02AF DB "789" FF "4" FF "6" FF "1230"; numeric keypad, cursor keys
xxxx:02BB
-a cs:368
xxxx:0368 DB "0" 00 00 "ss" 01 "wp" 02 ; abbreviations
xxxx:0371
-i cx
CX 0400
:400
-w
Writing 0400 bytes
-g
C>

```

Sample BASIC Code for Batch Parameters

```

10 ' $TITLE: 'BPARAM - change batch parameters', $LINESIZE: 96
20 ' This program breaks in to the MCB chain, locates the BATCH control
30 ' block, and alters up to 9 parameters.
40 ' Demo interpreted program - for any serious work, compile the code and
50 ' use the COMMAND$ line for feeding in useful prompts.
60 ' For really serious work, use C or MASM and the undocumented DOS call
70 ' MOV AH,52H
80 ' INT 21H
90 ' MOV AX,ES:(BX-2) ; AX now contains the start of the MCB chain
100 DEF SEG = 0
110 ST = PEEK(&HBA) + 256 * PEEK(&HBB) ' segment within COMMAND.COM
120 ENDCH = 0
130 GOSUB 300
140 IF ENDCH THEN PRINT "Memory allocation chain not found": END
150 GOSUB 500
160 IF ENDCH THEN PRINT "No blocks found for COMMAND.COM": END
170 GOSUB 500
180 IF ENDCH THEN GOSUB 700
190 IF ENDCH THEN PRINT "No batch file": END
200 GOSUB 900
210 GOSUB 1100
220 SYSTEM
270 '
280 ' 300 - decrement ST until it points to the first preceding MCB / PSP
290
300 IF ST = 0 THEN ENDCH = -1: RETURN
310 ST = ST - 1
320 DEF SEG = ST
330 IF PEEK(0) <> &H4D THEN 300 ' not MCB
340 COMPSP = PEEK(1) + 256 * PEEK(2) ' tentative owner address
350 IF COMPSP <> ST + 1 THEN 300 ' not MCB/PSP pair
360 IF PEEK(16) <> &HCD OR PEEK(17) <> &H20 THEN 300 ' not PSP
370 RETURN
470 '
480 ' 500 - Walk through memory chain to block belonging to COMMAND.COM
490
500 MCBLN = PEEK(3) + 256 * PEEK(4) + 1
510 ST = ST + MCBLN
520 DEF SEG = ST
530 PSP = PEEK(1) + 256 * PEEK(2) ' get owner
540 IF PSP = COMPSP THEN RETURN
550 IF PEEK(0) = &H4D THEN 500 ' not end-of-chain
560 ENDCH = 1
570 RETURN
670

```

W S Electronics

(513) 376-4348

**** Since 1975 ****

(513) 427-0287

1106 State Route 380, Xenia, Ohio 45385

W S ELECTRONICS INTRODUCES KRIS TECHNOLOGIES

System 48C-3
80486-33MHz CPU
and
System 48C-2
80486-25MHz CPU

- Internal co-processor and 8KB cache.
- 128K secondary cache onboard.
- Socket for optional Weitek co-processor.
- 2MB 70ns RAM, expandable to 16MB onboard.
- Shadow RAM for System & Video BIOS.
- ROM-based setup utility.
- Six 5.25" drive bays available.
- Eight 16-bit expansion slots.
- Dimensions: 8.5"Wx18.5"Hx18"D.

\$3550⁰⁰
and
\$2899⁰⁰

System 38SX-2
386SX-20MHz CPU
and
System 38SX-1
386SX-16MHz CPU

- Socket for 80387SX co-processor.
- 1MB 80ns RAM, expandable to 8MB onboard.
- Page mode interleaved memory design.
- Shadow RAM for System & Video BIOS.
- Two 5.25" & two 3.5" drive bays.
- Six 16-bit & two 8-bit expansion slots.
- Dimensions: 7"Wx13"Hx16.5"D.

\$1225⁰⁰
and
\$1025⁰⁰

System 38M
80386-33MHz CPU
64K cache

- Socket for 80387 or Weitek co-processor.
- 2MB 70ns RAM, expandable to 16MB.
- Page mode interleaved memory design.
- Shadow RAM for System & Video BIOS.
- ROM-based setup utility.
- Two 5.25" & two 3.5" drive bays.
- Six 16-bit & two 8-bit expansion slots.
- Dimensions: 7"Wx13"Hx16.5"D.

\$1999⁰⁰

System 28
80286 60-bit CPU

- Socket for optional 80287 co-processor.
- Standard 1MB RAM onboard, configurable to 640K base & 384K extended memory.
- Three 5.25" & one 3.5" drive bays.
- Six 16-bit & two 8-bit expansion slots.
- User selectable 8/4/2 DMA clock speed.
- Dimensions: 17"Wx6.5"Hx16.5"D.

\$750⁰⁰

System 38
80386-35MHz CPU

- Socket for 80387 or 80287 co-processor.
- 2MB 80ns RAM, expandable to 16MB.
- Page mode interleaved memory design.
- Shadow RAM for System & Video BIOS.
- Four 5.25" & two 3.5" drive bays.
- Six 16-bit & two 8-bit expansion slots.
- Dimensions: 19"Wx6.5"Hx16.5"D.

\$1499⁰⁰

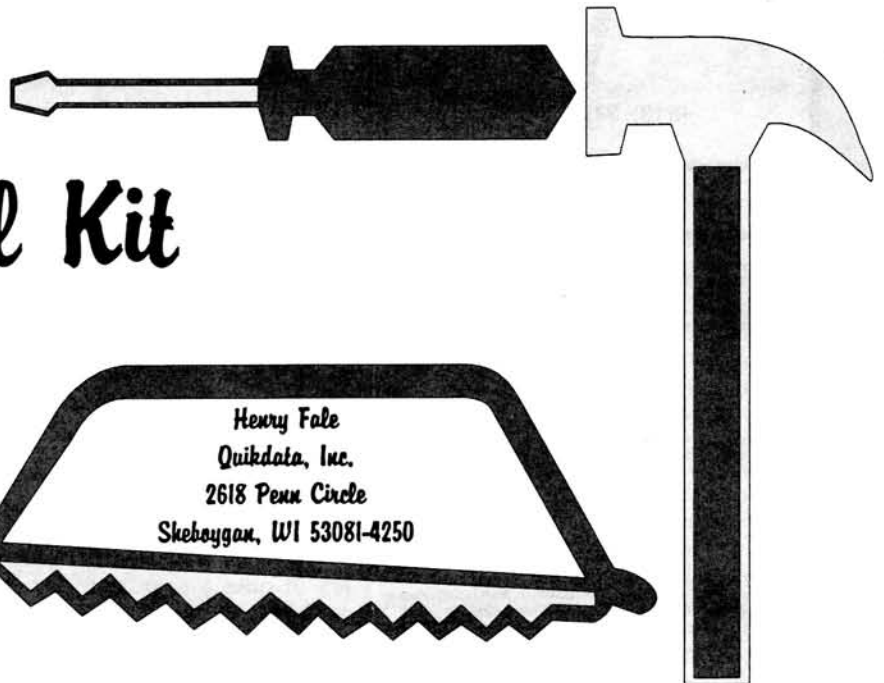


'ALL COMPUTERS HAVE A 2 YEAR PARTS, 18 MONTH LABOR WARRANTY'
*Monitors, Video Cards, Harddrives are optional.

```

680 ' 700 - work back from COMMAND.COM to kernel (only necessary for DOS 4)
690
700 ST = COMPSP - 1
710 ST = ST - 1
720 IF ST <= &H50 THEN RETURN
730 DEF SEG = ST
740 IF PEEK(0) <> &H4D THEN 710
750 IF PEEK(1) + 256 * PEEK(2) <> COMPSP THEN 710
760 MCBLN = PEEK(3) + 256 * PEEK(4) + 1
770 DEF SEG = ST + MCBLN
780 IF PEEK(0) <> &H4D THEN 710
790 DEF SEG = ST
800 ENDC = 0
810 RETURN
830
840 ' 900 - set two variables.
850 ' PCHAR: points to beginning of parameter 1 (old pointer %1 is not
860 ' reliable, there may have been a SHIFT command)
870 ' PCEND: points to the last usable position in the block; depends on
880 ' the lengths of the batch name and the initial parameter list.
890
900 IF PEEK(21)+PEEK(22)=0 THEN OFST = 7 ELSE OFST = 1
910 PCHAR = 40 + OFST
920 C = PEEK(PCHAR)
930 PCHAR = PCHAR + 1
940 IF C>0 THEN 920
950 POKE 20+OFST, (PCHAR-16) MOD 256
960 POKE 21+OFST, 0
970 C = PEEK(PCHAR)
980 PCHAR = PCHAR + 1
990 IF C<13 THEN 970
1000 MCBLN = PEEK(3) + 256 * PEEK(4) + 1
1010 PCEND = MCBLN + 16 - 3
1020 RETURN
1070
1080 ' 1100 - read up to 9 new parameters
1090
1100 I = 1
1110 GOSUB 1300
1120 FOR J = I TO 9
1130 POKE 20+2*J-OFST, 255
1140 POKE 21+2*J-OFST, 255
1150 NEXT J
1160 POKE PCHAR, 0
1170 RETURN
1270
1280 ' 1300 - read and poke a new parameter
1290
1300 IF I>9 THEN RETURN
1310 PRINT "Parameter"STR$(I): " ";
1320 LINE INPUT P$
1330 IF P$="" THEN RETURN
1340 POKE 20+2*I-OFST, (PCHAR-16) MOD 256
1350 POKE 21+2*I-OFST, (PCHAR-16) \ 256
1360 I = I + 1
1370 J = 1
1380 POKE PCHAR,ASC(MID$(P$,J,1))
1390 J = J + 1
1400 PCHAR = PCHAR + 1
1410 IF J<=LEN(P$) AND PCHAR<=PCEND THEN 1380
1420 POKE PCHAR,13
1430 PCHAR = PCHAR + 1
1440 IF PCHAR<=PCEND THEN 1300
1450 RETURN
    
```

Hard Disk Survival Kit



Henry Fale
Quikdata, Inc.
2618 Penn Circle
Sheboygan, WI 53081-4250

Hard disk drives have become a necessity of life. More and more of us use them. The price has come way down. You really can't get along without them if you're into serious computing of any kind. We don't always take care of our hard disks and we're not always ready for problems.

Several unfortunate instances that took place with customers recently have prompted me to write this article. I have had two frantic local customers lose their hard disk. I had to go out and get them going again. Also, several others have called and I've had to talk them through getting their hard disks back up. Besides this, I get several calls every week from folks who have had hard disk crashes and do not have current backups. I learned my lesson long ago, after losing my hard disk several times and not having an adequate backup. Thus, I've developed some helpful hints that I wish to pass along.

OK, so we've had a problem. Are we ready for it? In this article, we'll deal with my method, a proven method, of keeping out of cardiac arrest or a major panic anxiety attack. If it happens to you, and it will sooner or later, one way or another, be ready. You may accidentally delete the data on a directory or subdirectory. You may get a major power surge, get hit by lightning, crash the system, get a static jolt in the system, bang the table and drop the computer. Or you could transfer the hard disk and drop it on the floor (don't laugh that really happened to a customer). Or, as will happen sooner or later, the unit will die a natural death without giving you advanced warning, etc. But it will happen. Murphy's law says that sooner or later it indeed will happen. And if you don't have

a current backup, that's exactly when it will happen.

In the cases I am thinking about, the customer who crashed the hard disk did not have a current backup, and did not have the means to quickly bring the system up again. Thus, I'm going to deal in this article on what to have ready. I'll take a thorough look at the survival kit and explain the whole thing.

I know you've heard this till you're blue in the face, but I cannot overemphasize this point — KEEP YOUR HARD DISK BACKED UP-and DEFRAGMENTED! That's the only safe thing to do. With today's backup programs, backing up a hard disk is a fast and painless process. There is absolutely no reason not to keep current backup copies of your hard disk. If you don't have PC Tools, Fastback, a tape backup unit, or some similar means to easily back up your hard disk, put this article down and run out right now and get one. If you have some bucks and are lazy, then get a tape backup. With the tape, all you do is pop in a tape and walk away for a few minutes. With the floppy backup, you have to sit there and feed the computer disks.

I like the tape backup. You don't have to sit there, and a tape costing \$20 to hold 80MB of data is a lot cheaper than 80MB of reliable disks. Thus, the tape backup will not only save you time, but money in the long run. And the tapes store in a much smaller area per megabyte. This is very important if you want to store them in a fireproof vault. Ever price one? The bigger they are, the more they cost. My tiny one cost over \$1,000. I remember when my floppy backup data on 1.2 meg disks occupied my entire vault. Now my tapes sit in a

small corner in the back and there's room for some priceless family VCR tapes, photo's, etc. So I save money there. But if you don't use tape, then use the highest density floppy that you have. It will take less diskettes. Also using compression, which most backup programs have, can cut your backup media in less than half, since you normally average about a two to one compression ratio.

But the point is, one way or another, back up. The question is, how often do you back up? That depends on you and what you are doing with your computer. If you process lots of data each day, then you must back up the system daily. If you don't use your system a lot, perhaps once a week. The rule of thumb is, back up often enough so if something goes wrong and you lose all or some of your data, you can easily restore the hard disk to the current state before the crash or data loss. If you feel weekly is enough, and you lose your data right before your next scheduled backup, then I hope you won't have much manual entry to do to get the system back to where it was before the loss. Get the point? How much can you lose without a major loss in time or money? And don't only think of the whole hard drive going down.

I just had a personal experience where I did my usual del *.*., thinking I was on my ramdisk. Turns out I forgot to change drive designators and erased a directory on a partition. Only lost about 15 files, but they were important files. I knew I didn't touch that directory since last month, so I pulled up the EOM D: tape, brought up the restore menu, selected the directory to restore, and I was back to work in five min-

utes. No panic even.

I had a call from a customer who's boss uses a laptop and never backs it up. His hard disk went out. The customer managed to save many files, but not any of the spreadsheet files. Since the boss didn't have any backup whatsoever, and all of his spreadsheet files were gone, he will have to spend perhaps hundreds of hours reconstructing data.

I back up my total hard drive each month and do incrementals each day. In addition, I alternate backup sets. Have a tape for even months and one for odd months. I also save the end of year backups so if for tax purposes, payroll problems or something else, I can restore to last year end if need be. The increment tape I store for several months before reusing, and I use one per month.

Thus, it may be wise, especially if you are a business, to always keep an end of year backup set. This is very important where end of year processing has all the totals for inventory, payroll, etc. If you ever need to go back for IRS purposes, or anything, you can. If you are just a casual computer user, the end of year backup may not be a big deal for you.

Another important time to back up is when you do major software changes or upgrades to your system. For example, when I first installed Windows 3 and QEMM386, I backed up my entire hard drive in case something went wrong. I've heard from several customers who installed new software that they either wiped out their entire hard disk or severely modified it. So another rule is **MAKE A TOTAL BACKUP BEFORE INSTALLING NEW SOFTWARE**.

Alternate backups are very important. I recall one personal experience when I corrupted part of my hard disk. I did not know it. My customer database was corrupted. Since I did not yet know that, I backed it up. Unfortunately I was too cheap to buy extra disks for multiple backup sets, and used the set I had used the week before (See, I have learned my lessons well. I am older and wiser now, but I was sure foolish then. If I was lucky I backed up once a week, month, or whatever. Please learn from me). When I realized it was corrupt, I restored it, and that was also corrupt. Had I had one backup set earlier available, I would have restored it and saved a weeks worth of work. The way it turned out, I had to reconstruct what I could (and many times most folks are not capable of doing this, so don't count on it!) and start digging out scraps of paper and hire help to enter bits and pieces and rebuild the database — yes, my business, my records! Lesson, the more multiple backup sets you have (can afford, have time and ambition for — again, tape makes this easier) the better off you are. Some have monthly backups they keep for several months, and then they

alternate 5 tapes for each day of the week. That's a very excellent method. A tape for each day of the week. The point is, you have to decide that. But do it. Just remember. The file or files you may want to recover may be ones that you deleted months ago.

As for storage location, you decide that. I keep my data in a fireproof vault. It may be wise to have alternate backup disks in different locations — especially if you don't have a fireproof vault. If you have a fire and your backup set is in the same location, the backup set will be destroyed, thus worthless to you. Or if a thief steals your system and your backup is right there, chances are it will also be taken. Some folks keep a set at work and one at home, in a bank vault, etc.

And before we leave backup, use only good quality name brand diskettes or tapes. Cheap media will eventually damage your floppy drive because they tend to be abrasive and wear drive and tape heads more than good disks. And cheap media clogs the heads giving unreliable read/writes. This translates to drive repair time folks — helps keep me in business! Cheap disks will also give many data errors and will not reliably hold your data. What good is it if you lose data on your hard disk and your backup set starts to choke when restoring to the hard disk. If you trust your data to cheap disks, I hope that's exactly what you have — cheap data. Data that you don't mind if you have to input and reconstruct over again. Data that you don't mind if you can never get back. If saving a buck is that important to you, find another way. Don't skimp on disk quality. I can tell you many horror stories here from customers. I never did buy cheap media — I've done enough drive repairing to actually see what happens to heads), but I think you get the point.

That calls to mind another good point. Be certain you use error correction and verification options with your software. I use the CMS backup utility for their tape drives. It has both compression (which takes longer to backup, but can save disks or tape) and verification. I always verify a backup after it's done. Then you know you have good data you can depend on. If something happens, you'll know you can restore without problems. If you don't do this, you may find when you want to restore, the data is not reliable. I'd say a very wise thing to do would be to do a backup and a restore when everything is working properly just to be certain everything is working. If you continually back up, but have never had to restore, you may be in trouble once you have to restore your data. I have changed hard drives and controllers so many times that I have thoroughly tested my restore option. That brings to mind an important item. If you change your hard disk, make a double backup just in case something goes wrong. When I change

hard disks, like upgrading to a larger one, I make double backups, and keep the old hard disk with files intact just in case. If I have to, I can re-install the old hard disk and back it up again. In some cases, you can even transfer directly from one hard disk to another.

Thus, the first thing in the survival kit is common sense, coupled with a scheduled backup scheme. Develop a backup strategy and stick to it.

As Columbo would say, "one more thing". Don't only think of backup as a means to safeguard your hard disk data. One of the most common things I hear is "my hard drive is full, I need more space". Fine. If you have programs you hardly use, or old data you want to keep, use the backup to achieve it and then safely put it away. Then you can get it off your hard disk and get needed space back. If you ever want the files or data again, simply load it back up.

And one more use for a backup set. You can use it to easily transfer data from one computer to another. For example, I have some files on my work system that my kids want, or that I want on my home system. I simply back up that directory on tape, and restore it to my home system without hassle. It's much easier, and faster, than having a bunch of floppies with this file and that file that I have to work with.

The next important point is to **KEEP THE HARD DISK OPTIMIZED, OR DEFRAGMENTED!** Unfortunately, this is often taken for granted. Keeping your hard disk optimized will make disk I/O faster, which will cause the whole system to operate faster. If the data is available to the CPU faster, the system will run better. When the data is all in order, the movement of the hard drive head is much less to retrieve and write files, thus keeping hard drive movement to a minimum. This is one thing that will make your hard drive last longer. Not only that, but trying to recover files or "undelete" files on a defragmented disk is much easier than if the files are all fragmented. I have found Optune to be the best and fastest defragmenter on the market.

So now you're prepared. You keep your drive defragmented and have a good backup strategy. Next, it is most important to **HAVE A READY-TO-GO BOOT DISK WITH ALL NEEDED FILES**. This would include any DOS files you may need, such as **DEBUG, PREP, PART (FDISK), FORMAT**, possibly an **UNDELETE** and/or **UNFORMAT** program if available, your **RESTORE** program, and anything else you may need to get the system up and running. You may have enough space to also hold a system diagnostic test routine on the same disk. This would be a plus. And be certain that the floppy was made from the hard disk DOS; i.e., the DOS versions and files must be the same, or you may find you cannot

get on the hard disk if you have to. Best thing is to boot from your hard disk and do a `FORMAT A:/V/S` to create your boot disk. Then copy the files needed, and test to be sure it boots and works. Then, put it away in a SAFE place. When your hard disk does crash or die, you have a disk readily available with all the important files you need. Simply boot from that disk, run your restore and you're back in business. Or if you have to check the hard drive to see if you can get on it, the disk is all ready. Don't wait until the system crashes to gather all the files you need to check the hard drive and/or restore it.

For reasons not so obvious, ALWAYS HAVE A BARE SYSTEM BOOT DISK WITH THE PROPER DOS WITH ALL NEEDED FILES ready to go. Note that this is not the same survival disk we talked about previously. This is basically a virgin DOS with no TSR's or junk in `CONFIG.SYS` or `AUTOEXEC.BAT`. Sometime when you run optimize programs (defragmenters) or other things, you are requested to run a system without TSRs loaded, without cache, etc.

I've also had programs I had to run which told me I did not have enough RAM. In times like this, it is very handy to have a virgin disk to boot from. Then, you can access the hard drive after booting DOS from the floppy and you'll have a pure DOS system without drivers, EMS installed, CACHE, etc. This disk will give you the maximum amount of RAM from your system with no drivers, etc., loaded.

And last, but certainly not least, know your system parameters! Know your hard drive type, the number of heads, cylinders, write reduce, write precomp, step rate and shipping zone. If you have a '286 or '386 class machine, go into the SETUP and record all pertinent information. On a Zenith Data Systems machine, CTRL-ALT-INS will get you to the monitor ROM level where you type SETUP and the screen will appear. On most PC clones, you hit the DEL key when the system begins to boot and from there you should be taken into an option screen with one choice of CMOS setup or something similar.

This is really important when the bat-

tery backup power device fails, which they will do after several years. If you enter the improper hard drive type, or the wrong parameters, you can really screw up your hard disk. In any case, if you have to restore the hard disk, you must be certain it is specified correctly in the SETUP screen before you proceed, or you'll get in more trouble than when you started.

We all hope you will never need to restore the hard disk, but if you do, at least you will be ready. You have the backup disk. If you have problems, boot from it. The first thing you want to check is if you can get on your hard drive, usually C:. If you can't, the next thing you want to do is `FORMAT C:/V/S`. If you can't even do that, go into PART (or FDISK) and see if you have valid partition data, and a valid boot drive. If you do, you should be able to FORMAT the drive; if not, you'll have to do a low-level format to the drive. PREP can be used in most cases, especially if you have a Zenith Data Systems computer. With some controllers you enter a DEBUG routine, so check your documentation. KNOW what to do before you need it, so when it happens, you are ready. Know your drive heads, cylinders, etc.

If the DEBUG must be used (as a rule this is for PC/XT controllers, and sometimes ESDI controllers, but generally not MFM or RLL AT controllers), then know in advance the hard drive controller address, which is usually C800, and the form to invoke DEBUG, which is: `G=C800:5`.

But you must know all your drive parameters to use this. In any case, if you need to do a low-level prep and you can do it, then you must go through PART (or FDISK) after that, and then the FORMAT (high-level format). From this point, or after you verify that C: is valid, you can then run the RESTORE part of the backup program and load in all your disks. After that, your hard drive should contain exactly what it did at your last backup.

If you can't even do a low-level format, then either the controller or the drive is bad and must be replaced.

I certainly hope this has helped you. If you do not have a decent backup and have not been following a schedule, now is the time to do it. Be ready for Mr. Murphy, and you will be glad that you did. ✨

Continued from Page 8

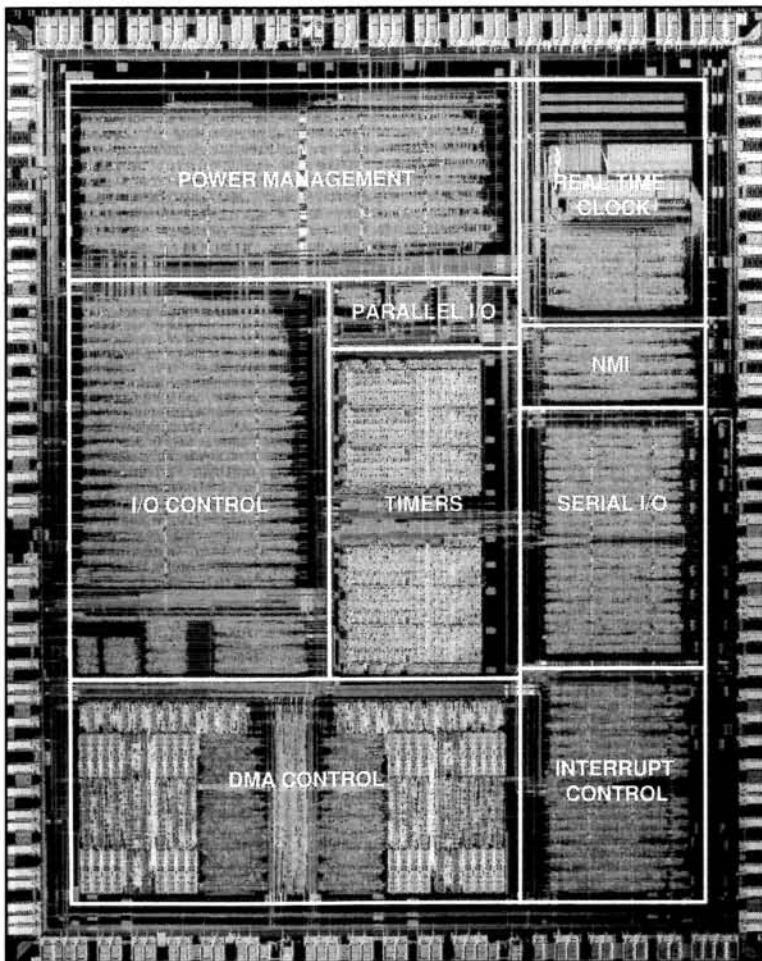
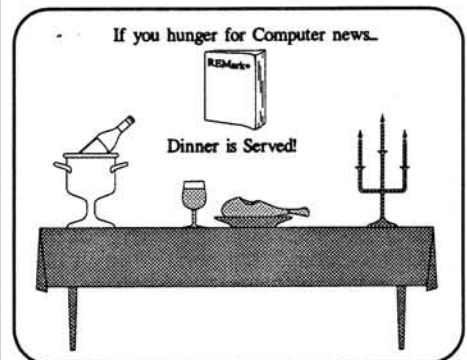
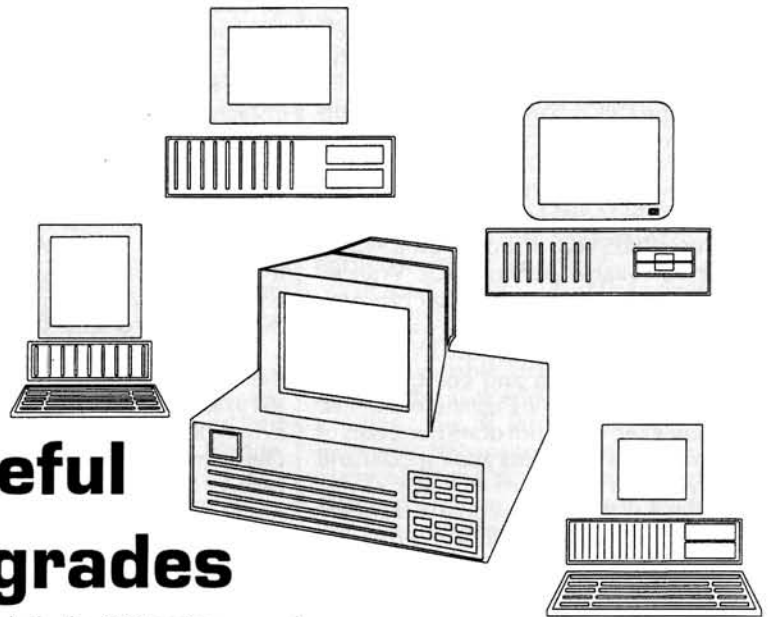


Figure 4

The 82360SL I/O sub-system chip contains 226,000 transistors in 460 square mils, and is fabricated on Intel's one-micron CHMOS-IV process technology. ✨



**Henry E. Fale
Quikdata, Inc.
2618 Penn Circle
Sheboygan, WI 53081-4250**



Heath/Zenith Useful PC Computer Upgrades

There are quite a few of the older generation Heath/Zenith computers still out there, and they are too valuable to scrap or put away in some closet just because you may not know about possible low-cost upgrades to those systems. In fact, there are a lot of neat upgrades out there for the entire line of Heath/Zenith PC/XT/AT computers, which can make your system more useful and valuable to you. The problem is, many folks don't know some of these things exist.

We have been working with Heath/Zenith systems for the past fifteen years and have collected quite a few tricks and products to breathe new life into those systems. The purpose of this article is to discuss some of these updates for specific models of computers, and then mention some products that work on all the computers. You'll be surprised to learn what you can do with your system.

Although this may sound like a large advertisement, that is not the intent of this article. I mention what we have worked with here at Quikdata and what we can supply. Many other vendors can supply the same or similar products in many instances, so feel free to check around.

H/Z-151/161 Line

The H/Z-151 was Heath/Zenith's first attempt at a PC-compatible computer. It was one of the earlier PC computers on the market, and it seems to me that there are tons of the H/Z-151/161 systems out there, because we are selling tons of upgrades of all sorts for them. Besides being a tad slow, this has got to be one of the best PC computers to hit the market. The '151 was the desktop version and the '161 was the "transportable" version. They are both the same as far as the bus, cards, etc. It's just that the transportable '161 was constructed with video built-in to make it an all-in-one luggable! Thus, when we talk about the

'151 we also include the '161 series.

More memory is one of the most popular requests of any computer owner. It just so happens that the first improvement to appear, which is still available, was a sensible way to increase system RAM from the supplied 320K to at least 640K without having to add a second board which costs a fortune, takes up an additional slot, consumes power and radiates heat. For those who have one of these systems, you'll remember that the standard RAM board in that system used 5 banks of 64K chips (256K chips were not readily and inexpensively available when that system was designed and sold) giving a total memory of 320K.

It was not long before a scheme was devised whereby changing the memory mapping PAL would allow 256K DRAM chips to be used on the board designed for 64K DRAM chips. We have been selling a very successful memory mapping PAL (current price is \$17) which will allow you to replace two banks (18 chips currently at \$1.79) of 64K chips with 256K chips giving a total 704K RAM on the memory board. The whole works comes to under \$50 compared to several hundred with a second 320K card the H/Z way.

All this upgrade involves is replacing a PAL chip, swapping two banks (18) of RAM chips, moving a jumper plug and setting of CPU dip switches. Very easy to do, full documentation is provided. Presto — 704K system RAM.

If you're not satisfied with that, we can give you the MEGARAM, which will give you a total of 1.2 meg of RAM on that board! Of this, 704K can be allocated to the system and over 500K (half a meg) is allocated to a RAM disk or print spooler. With this PAL change you need to swap out all 5 banks, or 45 chips to get the full upgrade. It can use a minimum of one bank each of 256K and 64K chips. You can set

the main system memory for 640K, which is needed for those upgrading to EGA or VGA video to keep the memory out of the video RAM area, which for EGA and VGA starts after the 640K boundary. This kit costs \$43 plus the RAM chips. RAM chips, 256K dynamic types, at 150ns are currently going for \$1.79, and again, for the full 1.2 meg you require 45 chips. Total cost of that upgrade is \$123.55.

One more spin off of this is the LIM150 for the same \$43 price. This gives 704K system RAM and 512K simulated LIMv3.2 EMS RAM. Good for LOTUS 1-2-3 and some other software that can use EMS memory. It will be slower than a regular EMS board, but it's also cheaper.

How about video? Well, those systems were only equipped with composite monochrome and CGA video. That's all there was folks! Now many want to run EGA and VGA video. Problem. Since nobody ever dreamed of other video schemes in those days, no provisions were made (they are there in the Z-158 and '159) for removing the video card. And since the video card also contains some RAM that is absolutely necessary for the system CPU, you can't just remove the card either.

Long ago, ZDS came up with the ZCA-6 video disable PAL. This allowed you to swap a PAL on the video card and disable it. You could then add other video cards. ZDS has since discontinued that product (but Quikdata has not — \$20 will get you a video disable kit). It works, but why disable a video card that will no longer be used (except for the RAM) and consume valuable power, take up an expansion slot and radiate extra heat when it is not necessary!

Those without many slots filled and on a tight budget may still opt for the ZCA-6 option. Those finding slots at a premium or concerned about power and heat problems will opt for the VCE-150 for \$45. The

VCE-150 (video card eliminator) is an easy to install mod for the CPU board which adds the needed RAM to the system so the video card can be totally removed! It installs on the CPU board. Thus, you free up a slot, free up power, and eliminate excessive heat problems. Then, you simply put your EGA or VGA video card where the original video card resided.

Your BIOS ROM should be a -15 or higher for reliable performance. With either the ZCA-6 or VCE-150, you can now add an EGA card or VGA card with appropriate monitor, and you now have some really nice resolution and color capabilities. Well worth beefing up the old beast.

However, we're not done here. Lots of folks have had problems adding EGA and VGA video. It was hard to trace at first because of all the mods people were doing with their systems, but it boils down to this. If you have only 640K of system RAM, there should be no problem. There have been many folks that have installed EGA or VGA and have had subtle problems. Some software worked fine, some worked erratic, and some would not work at all. It was finally narrowed down to memory above 640K. With many of the memory mapping PAL chips on the market for the systems, anything above 640K could possibly interfere with the video RAM. You see, EGA and VGA both contain on-board RAM for their operation. If you recall that the total addressing capability of the 8088 is 1 megabyte, you can see that the system 640K RAM, Video RAM, and ROM must all reside someplace in that area. Turns out that EGA and VGA use RAM addresses right above the 640K, and if extra SYSTEM RAM (which is not possible in many systems, but is possible in the '151/'161 series) is installed, it will conflict with the EGA/VGA video RAM. The solution has been to be certain that the system RAM is set for 640K maximum (and of course, that the upgraded BIOS monitor ROM is installed).

H/Z-148 Series

A while after the '150/'160 series appeared, the H/Z-148 series came out. This was another "transportable" computer, but unfortunately, it had no expansion slots of its own. Heath/Zenith came out with one which would accept a full size and a half size card (1-1/2 expansion slots), but of course, have since discontinued it. Thus, anyone wishing to add a hard drive, internal modem, upgraded video, or anything else requiring a PC expansion slot was at a dead end. Enter third parties again who produced a similar board, which is once again available.

FBE Research makes what they call the ZEX-148 which is a small daughterboard and all needed fasteners, etc., to allow mounting the board in the Z-148 computer to once again give the 1-1/2 expansion slots. We handle that card and it is available

for \$65. I have heard that with some coaxing it may work in an H/Z-138 computer also.

Once you have this daughterboard installed, you can easily add another video card, such as a VGA card, and one other card, such as a hard disk controller card. Thus, it is possible to install both a hard drive in place of the B: floppy drive, and upgrade your video. When you upgrade the video, you simply locate the video enable jumper on the main board and cut the jumper to disable the CGA video.

The only other useful specific item for the '148 is a PAL which we call a ZP-148 (\$19) which will allow you to have 704K of system RAM with 3 banks of 256K DRAM chips instead of the 640K that you had before. Same caution applies to this upgrade. If you are running or are going to run EGA or VGA video, then keep your system RAM to 640K.

H/Z-159 Series

Not much in the unique department for this machine, just the EMS logic chips set upgrade which ZDS used to call the Z-315. The Z-159 was made to accommodate 5 banks of 256K chips (45 chips for a total of 1.2 megabytes). Of this, 640K (3 banks) was devoted to the system. If one had the additional EMS logic chips installed, then by filling the remainder of the memory sockets, anything above 640K could be used as EMS memory.

The later Z-159 systems sold contained this chip set automatically. The earlier units did not and it was available as a special item which sold at that time for around \$300.

We still market a kit for those who purchased the earlier versions of the machine and would like to add the additional RAM. Our Z-315 for \$79 consists of one bank (9 chips, or 256K) of RAM and the 5 EMS logic chip set. To go all the way, you simply add one more bank (9 256K DRAM chips).

Z-241/248 Series

ZDS' first entry into the AT class machines was a 5MHz 80286 system known as the Z-241 computer. Shortly after this, they introduced an upgraded model known as the Z-248. This came in two flavors, the earlier ones were 5MHz and the later ones were 8MHz. ZDS' latest Z-248/12 is an entirely different machine, so we don't want to confuse them. The '248/12 deviated from the ZDS approach to separate CPU board to plug into the motherboard. Like everyone else was doing, ZDS finally made a system with CPU and support on the main board, along with the RAM. This really keeps down RAM cost, since you plug in SIMM modules to increase your RAM (up to 6 meg is possible on the main board).

The Z-241 and '248s only had 512K RAM on the CPU board, which as far as I'm

concerned was one of the biggest mistakes ZDS ever made in a computer design. This already was when everyone had to have 640K of RAM. That meant one had to go right out and purchase a ZDS memory board to backfill from 512K to 640K. Unfortunately, the ZDS boards are no longer available.

Many third party enhancements do not work in many H/Z PC computers, but that's another story in itself. We did find a board that worked reliably in the H/Z-241/248 systems. The Everex RAM 300 Deluxe is a 16-bit AT board which can hold up to 3 megabytes in 256K DRAMs (100ns if a 8MHz system, 120 if a 5MHz system). The board is unique in that you can use the RAM for backfill, extended and expanded at the same time from the same board. For instance, you could allocate 128K to backfill the base to 640, have one meg of extended and one meg of expanded memory. Our part number for this board is EVATRD and it sells for \$119, much less than the original H/Z card sold for. In any AT class memory board, you populate 18 chips at a time (this gives the 16-bit memory, plus the two needed parity bits) which gives 512K. Thus, 36 chips gives one meg, and 108 chips gives the entire 3 meg. More than one board can be used to go beyond the 3 megabytes. At this writing, 120ns chips are going for \$1.89 and 100ns chips are going for \$1.95 each.

Another popular item for the AT class systems is I/O boards. Suffice it to say, there is a wide variety of serial and parallel port boards available for these machines. We have been selling one by Boca Research that contains two additional serial and two additional parallel ports for \$75. Thus, you can easily add extra ports to your system.

To upgrade from the supplied CGA video on these systems, you simply remove the video card and add an EGA or VGA video card. In these cases, it is best to upgrade your BIOS ROM to the latest version.

Before I leave this section, I want to mention some new upgrades that we are presently evaluating. We have had, for some time, the Sota EXPRESS 386 accelerator for the Zenith Data Systems Z-241 and Z-248 computers (not the 12MHz version). They have both a '386 SX 16 MHz and a 20 MHz version available. We have had some problems with this and are still working with it. It's a neat device that plugs directly into the CPU socket, which means it does not take up a slot. The problem with the '241/'248 series is you require two adapters which increase the price by another \$100 or so. Another problem was with the supplied PLC chip puller, you could not get the 80386 out of the socket. We had to purchase a rather expensive chip puller to accomplish this just to get the CPU chip out to test the thing. Sota is

working on a board just for this computer to offset the accelerator board so the two additional adapters are not needed, but the cost will be about the same. It will probably be in the \$600-\$700 range for the 16MHz version, and another \$100 for the 20MHz version. It does work in the machines and it was verified that things like Windows 3.0 and QEMM386 will work also. However, some problems exist. Certain test software that I used would not run on the system with the accelerator installed. Also CTRL-ALT-INS would not reset the machine. We are still working with this to see if it's worthwhile handling it or not.

We've checked with several other companies including AOX and Kingston who have recently come out with '286 upgrades to a '386SX. I checked with both of them and their upgrades will not work with Zenith Data Systems' machines.

MicroWay is also working on one which should be out by the time you read this. Their version takes up a slot, but will probably be cheaper since the adapters may not be needed. But since I haven't seen it yet, it's too early to tell. We'll have a future article on these if they are worthwhile, so keep reading REMark.

EaZy PC Computers

ZDS created and sold a very limited number of EaZy PC computers, which were very basic bare PC systems with a built-in monochrome monitor and very limited expansion. These never went over well, but there are many out there because they dumped them in the liquidation channels. Many purchased through these channels and got them at low prices. They were shocked to find that ZDS discontinued the memory upgrade and serial port.

Well, again a third party, FBE Research came up with the needed upgrades. There is the EZM-128 (which we sell for \$89 as EASYRAM) which will plug onto the back of the system and add the additional 128K RAM to bring the 512K system RAM up to 640K which is more useful to run the full range of software available. If you have that, you can add a standard serial port (the supplied port is a mouse port and will not readily handle external modems or modem software properly). This board piggybacks onto the FBE EZM-128 board to give you a standard serial port to which you can add a serial printer or a modem. Our part number for that is the EASYSER for \$43. If you just want the serial port and not the memory upgrade, you must add the stand alone serial port option (EASYSPO) for \$20. Unfortunately, if you already have the ZDS RAM upgrade and want a serial port, you'll have to remove it and go one of the above ways. The FBE serial port option (and clock option to be discussed next) is not compatible with the ZDS RAM option.

The last option for the EASYRAM is the clock option, which also piggybacks onto

the EASYRAM board. This gives your system a clock/calendar with a lithium backup battery. We call it the EASYCLK and it sells for \$33. If you just want to add a clock and don't have the EASYRAM, and you're bold enough, you can take apart the machine and install the SMARTCLOCK described later in this article.

Z-100 Series of Computers

Not to be forgotten, is the Z-100 series of computers. Although non-PC, there are some enhancements you may want to know about since there are many of these out there.

First off is the memory upgrade to these systems. The majority of them came with a maximum of 192K on the motherboard, and that's all you could have without adding an expensive S-100 bus board. Well, there is a plug-in modification out which is easy, does not require an S-100 slot and brings the RAM up to 768K. You simply install a few chips on the motherboard, remove the 27 (3 banks) of 64K DRAM chips and install 27 new 256K, 150ns DRAM chips. Total cost of the upgrade from 192K (or 128K, whatever you had) to 768K is just over \$100.

Note that if you already have either 256K or 512K RAM, you either have one of the mods installed or have the latest motherboard (the 8MHz one) which does not require anything except extra 256K DRAM chips.

There are two versions of the upgrades available. You must first determine which motherboard you have. Look on the inside of the computer at the rear. On the top of the J2 serial port connector there should be a white sticker with a 181-xxxx number. If you have a 181-4918 number or greater, then you need the Z100MP upgrade (\$55 without RAM chips). If you don't see the sticker, then check the assembly number of the CPU main motherboard for an 85-xxxx number. If it's 85-2806-1, then you need this upgrade. If it's 85-2653-1, then you need the other one to be discussed next. This is simply five replacement chips which get swapped. You just add 27 256K DRAM chips in their place and you're finished.

If you have a number less than 181-4918 (assembly 85-2653-1), then you require the ZMF100 (also \$55), plus the 27 256K DRAM chips. This mod also replaces several chips. But in addition, a bus bar is provided which plugs into all the pin 1 locations of the RAM sockets. It's really an easy mod. Both of these mods come complete with instructions. No soldering or trace cutting is required. In both cases, you have a lot of RAM to work with after the modification.

There is also the same SMARTCLOCK available for this machine as discussed below, but you also require an additional \$2 spacer kit.

We don't want to leave here without

mentioning the ZPC software written by Ace Software Engineer Patrick Swayne of the Zenith Users' Group. Probably the most phone calls we receive on the Z-100 is: "Is it PC compatible, and if not, can it be made to be?". There were, long ago, two companies, Gemini and UCI Corporation that made hardware mods which would make the systems PC compatible. They were both expensive (around \$500) and have since been discontinued. With ZPC (available from the Zenith Users' Group software library for \$60), you can run a lot of PC software on these machines. Contact ZUG for more on this. For \$60 to get PC compatibility, you can't go wrong. (Note: ZPC requires 768K of RAM.)

General PC Enhancements

We have so far discussed enhancements specific to various computers. We will now go into some neat stuff that can be used on most computers. First, we'll cover some neat stuff for the PC/XT line. This includes computers such as the '151/'161 series, '148, '157, '158 and '159 series.

System Clock. The next most common request we receive is for a system clock. Easy. Unlike the older generation clocks which took up a PC 8-bit slot, the SMARTCLK for \$33 simply plugs into the monitor ROM socket on the CPU board of a PC/XT, and the BIOS ROM then plugs into the clock. It has a sealed lithium battery contained in the "IC", which is said to last 5-10 years. Just use the supplied software to set the clock and place the time commands in AUTOEXEC.BAT, and you'll never have to enter time and date again. If you have version 3.3 plus of ZDS DOS, you don't even need the supplied software for the clock as the DOS already supports the smart clock.

Accelerator — Speed Up the CPU. The next most common request is how can we speed the beast up? If you recall, that was H/Z's first PC design, and it was only a 4.77 MHz system. There have been several speed mods out for these systems, but they have been too pricey for what you get. Enter the Sota 286i and 386i accelerator cards. They work in the H/Z systems, while most other third party accelerator cards do not.

The 286i (our number is EXP12) for \$295 is a 12.5MHz 80286 processor board with 16K on-board cache RAM which plugs into a slot and cables over to the 8088 CPU socket. The 8088 CPU then gets installed on the EXP12. Install some software and you've got a system which will rival the original IBM AT computer. You should see a 200-600% performance increase, depending on what you are doing. In very CPU intensive things, like spreadsheets, CAD, publishing, etc., there is a very noticeable speed increase. There is even a 16-bit daughterboard memory option which actually turns your system into a true '286 or

'386 machine allowing you to run Windows, OS/2, etc. However, after purchasing the accelerator, the memory card (about \$260 w/o RAM, RAM about \$60/meg - up to 4 meg can be used) and RAM, you begin to be better off just buying a '386 SX clone or something. Anyway, on both of the accelerator cards, you can also add a co-processor.

With this enhancement, you must be certain you have the MONITOR BIOS ROMs located on the CPU board of at least a -16 level for the '151/'161 series. There is a 444-229-xx and a 444-260-xx monitor ROM on the CPU board. The 'xx' is the revision level. It is currently up to a -18. As a rule, this is a good part to have upgraded, especially if you are at a low-level like a -4 or -5 or something ridiculous like that, since it's where your PC compatibility comes from. For video upgrades, speed enhancements and some other upgrades, a -16 or higher number should be used. The new BIOS ROM also supports the 101-key keyboards.

For other computers like the '148, '158, '159, check to see what BIOS ROM is needed for the speed change. We can supply you with that information if required.

The Sota 386si is similar, but is a '386 SX 16MHz accelerator. It also works with all the H/Z PC computers. It used to be \$395 from us, and probably will eventually fall back to that price. As I write this (May), Intel has created a severe shortage of 80386 SX processor chips, and thus added a \$75 surcharge per chip, making the board \$469.

Hard (Winchester) Drives. You can easily add a hard drive to any of the H/Z PC/XT computers simply by adding a hard disk MFM controller card (or RLL, although I never really believed in RLL), cables and hard disk. The drives available today are small half-height drives, either 5" or 3.5" in 5" frames. They are fast drives and have a low power consumption. While you normally remove the B: floppy and install the hard drive directly in its place in the full size desktops, such as the H/Z-151, '157, '158 and '159 series, you can usually sneak one in under the B: floppy if you wish to retain the B: floppy.

If you have one of the earlier laptops that did not have a hard drive installed, you can still add an external hard drive to those systems. This is especially useful for the old '181 and '183 series of laptops, and is also well suited for the MinisPort and EaZy PC computers.

We are presently evaluating some of these units so I can't say very much about them at this time. However, they interface via the parallel port and you can still use the parallel port for the printer. Two lines are added to the CONFIG.SYS file for the hard drive driver from software supplied. That's basically it. Connect up, configure the software and go. The units work remarkably well and fast. I tried my evaluation unit on

my MinisPort, a Z-248 and a Z-159 all with good results. The unit we are evaluating is from UCI, but several others also have them available. The 20MB external unit has a price under \$500.

Additional Floppy Drives. Other options common to laptops and general PC/XT/AT desktop computers include modems, hard drive controllers and floppy controllers which support 1.2 and 1.4 meg floppy drives. We will briefly discuss the Compaticard floppy controller and the hard disk upgrades. Since modems are straight forward, we will not devote much time to them, except to say, be certain that the COM port address of the modem does not conflict with any COM ports in the system. Remember, the computers originally had two serial ports, COM1 and COM2, later to be changed only to COM1, with COM2 as an option.

If you have DOS 3.21 or higher, you can replace your B: floppy drive in any PC/XT with a 720K 3.5" floppy. Since these come in a 5" frame, the 720K 3.5" floppy is a direct replacement. Screw holes and cables are the same as for the 5". Just be certain to consult your DOS manual for DRIVPARM. If replacing the second 360K floppy, you'll have to include a line in your CONFIG.SYS file telling the system the B: drive is now 720K. This line will do the trick:
DRIVPARM = /D:1 /F:2

The 2 in F:2 tells the system the drive is 720K format, and the 1 in D:1 tells the system it's the 2nd drive.

Although Zenith Data Systems and Heath Company do not seem to realize it, you can add 1.2 meg 5" floppy drives, 1.44 meg 3.5" floppy drives, and even 8" drives to any PC/XT/AT computer. There has been some really interesting things happening with disk drive add-ons of late. Anybody with a PC/XT/AT compatible or a laptop can now add high-density drives, such as a 2.8MB or 1.4MB 3.5", or a 1.2MB 5" floppy drive to their systems. There is both a card version which requires an 8-bit slot (laptops with the expansion chassis can go this route), and a parallel version. The parallel port version still retains that port to drive a printer. These products are both excellent if you want to add one or more drives to your system to get better disk compatibility with another system, say laptop to desktop and vice versa. Interested?

The first product from Microsolutions, makers of the famous Compaticard is the Megamate. It comes in two 3.5" flavors right now. You can get it with a 1.4 meg drive or a 2.8 meg drive. For the small price difference, it's foolish not to start with the 2.8 meg version. The 1.4 meg version will handle both 1.4 meg and 720K 3.5" floppy disks. The 2.8 meg version, however, will handle the new 2.8 meg high-density format, and the 1.4 and 720K formats. Thus, getting the 2.8 meg version for a small price

increase gives you the best of all worlds.

The Megamate is an entirely self-contained system including a Compaticard 4, cable, cabinet, drive, software and documentation. All you need to install it is a screwdriver. It's easy to use, just like your existing floppy drives. The 2.8 megabyte version will store almost 3 megabytes of information on one disk. The only problem is, at this time, the disks are rather expensive, but they are coming down in price. They originally sold for \$10 each, and we are now selling them at \$7.40 each. (I tried a standard 1.4 meg disk and it worked fine at almost 3 megs, but I would not suggest doing that.) The unit itself is very small, just a little larger than the size of a diskette, thanks to the unit taking power from the computer. Thus, it will fit on top of your computer, or easily on your desk.

Installation consists of removing the cover from your computer, installing the half-card in an 8-bit card slot, plugging in the cable of the drive unit to the card on the rear of the computer, and installing the driver software in your CONFIG.SYS file. Boot up your system, and go. The drive will assume a drive designator letter after your hard disk letters. That's all there is to it.

We are currently handling these products. They have a one year warranty and the 2.8 meg version is \$295. We don't stock the 1.4, but it's even less expensive.

Similar units, but not as complete, are the Compaticard family. This is simply a controller card which we have tested as a secondary card for the H/Z PC/XT line, the laptops via expansion chassis, and AT computers. There is a Compaticard which will control up to four of any kind of drive, including the 8" drives. It can be set to one of four addresses, so it will work in all the H/Z systems and not conflict with the primary controller. The DOS version being used does not matter, as it comes with its own driver software. Simply configure the card, plug it in, configure the software, connect any floppy, and away you go. It will assign the new floppy drive to a letter after the system drives. It has a DB37 connector on the rear for two floppy drives, and has two 34-pin male connectors on the card itself for floppy cables. Cost for this one is just under \$100.

There is a Compaticard II, which is similar to the above, but only handles two floppy drives and does not have the DB37 connector on the rear. It's about \$80.

The one the Megamate uses, is the Compaticard IV. This one is quite different in that it will also support the 2.8 megabyte floppy drives, whereas the others will not. It also has an on-board boot ROM allowing it to be used as the primary controller BOOT device in cases where this is possible. But it will handle up to four floppy drives like the standard Compaticard. The price is about \$125.

In each case, each connector will con-

trol up to two floppy drives, but if more than one floppy drive is used per connector, the cable must be of the twisted variety.

If you don't have a slot, such as with a laptop, then what? An alternative to taking up a slot is the Microsolutions Backpack units which simply plug into a parallel port. This makes it excellent for connecting to computers which do not have an expansion slot to spare, which of course, immediately targets the laptop computers.

The Backpack comes in many flavors. You can get 360K, 1.2 meg 5.25", 1.4 meg 3.5", and 2.8 meg 3.5" drives. Thus, if you have a laptop with 360K disks, a good choice would probably be the 1.4 meg or 2.8 meg units. Then, you would have both 5" and 3.5" capability. If you have a '286 or '386 laptop, a good choice would be the 1.2 meg unit which adds the lacking 5" format to your 1.4 meg 3.5" present format. For a computer like the MinisPort, which does not support a hard drive (new units do), the 2.8 meg unit would be an excellent choice to give you the 3.5" format, and also almost 3 meg of storage capacity. That would start making the limited capacity MinisPort a much more valuable computer. It should be pointed out here, that distribution media for the Backpack is on 5.25" and 3.5" disks. Since the MinisPort requires 2" disks, you'll have to have the distribution software converted to the 2" format before you can use it for the MinisPort. We can do this conversion for an additional \$10, so be certain you specify MinisPort when ordering this for that computer so we can send along a 2" disk for you. For those with desktop computers, you can easily add any format to complement what you already have simply by plugging the cable into your parallel port and adding a driver to your CONFIG.SYS file. If you want to use a printer, it can then be plugged into the rear of the Backpack unit.

If you want to add more than one drive or one type of drive, you can daisy chain several Backpack units together.

Special software is provided which will allow you to format disks in the background while your system is doing other things. One year warranty.

If you are interested, here are our order codes:

Model No.	Drive Size	Price
BPACK2.8	3.5" 2.8MB version	\$319
BPACK1.4	3.5" 1.4MB version	\$269
BPACK1.2	5" 1.2MB version	\$269
BPACK360	5" 360K version	\$269

Other Goodies. Here's a real neat product. Want to have two parallel ports from one? There is a new product out by XIRCON called the PARALLEL PORT MULTIPLEXOR. This has been a blessing, especially for laptop owners who require a printer and a lan adapter or something similar, since you can't add another parallel

port to a laptop. No more need to lose your printer port. Installs directly on the parallel port of your laptop or PC allowing you to connect both a pocket LAN adapter and a local printer to the same parallel port. Or need to access two printers? Eliminate the need for switch boxes or another card. With the multiplexor you can access them both, even at the same time. And there are no switches to mess with. The multiplexor actually adds an additional industry-standard parallel port to your existing PC configuration without using a slot. Plugs into the parallel port. LPT1 turns into LPT1 and LPT2. LPT2 turns into LPT2 and LPT3. The 3 x 2.3 x 1.4 inch device includes a small driver that comes both in a .SYS and .COM format, making it easy to install. Comes with 8" extender cable to make installation easier. The only things this will not work with is programs, such as FastLynx, which bypass BIOS and DOS calls. The cost of this

is under \$100.

ZDS Computer Current ROM Versions

I don't know how many calls I get asking what a current ROM version is for a certain computer. Since we're covering upgrades and in many of these upgrades ROM upgrades are essential, I'll also present the latest (as of May) ROM versions for the ZDS PC series of computers. This listing of the computer and associated ROM part number and version, both in revision level and part number revision is Figure 1.

That's the end of this article for now. Believe it or not, I started on this early in 1990. I kept wanting to submit it, but continued to find new things to add to the article, so it continued to grow. But all good things must come to an end, and so does this article.

I'll cover other upgrades and newer products in a future installment.

Computer	Date	Ver	p/n
Z-148	05/11/89	3.1E	444-380-08
Z-150	05/11/89	3.1E	444-260-18 444-229-18
Z-157	05/11/89	3.1E	444-561-04
Z-158	05/11/89	3.3C	444-358-07
Z-159	05/09/89	3.1E	444-494-07
EZPC 1,2,3	N/A	2.0B	444-613-04
Z-181	08/16/88	3.1D	444-502-04
Z-183	08/16/88	3.1D	444-656-01
Z-184	08/16/88	3.1D	444-657-01
MinisPort	06/13/90	3.4C	444-804-03
MinisPortHD	09/25/90	3.5	444-120-03
SlimsPort	10/24/90	3.6E	444-883-04
SupersPort286	09/05/90	3.4E	444-671-05 444-672-05
SupersPort286e	06/19/90	3.3F	444-806-05 444-807-05
SupersPortSX	04/05/90	3.3F	444-808-05 444-809-05
TurbosPort386	09/13/90	3.4C	444-651-04
TurbosPort386e	03/15/90	3.3D	444-788-02
Z-241/248	12/13/89	3.5	444-423-12 444-424-12
Z-248/12	11/16/90	3.5	444-643-11
Z-248SX	11/05/90	3.6B	444-793-05 444-794-05
Z-286LP/8	11/16/90	3.5	444-643-11
Z-286LP+	11/05/90	3.6B	444-793-05 444-794-05
Z-386/16	02/06/90	3.2C+	444-549-10
Z-386SX/16	12/12/89	3.6B	444-832-02
Z-386SX/20	11/19/90	3.6D	444-989-03
Z-386/20	12/05/89	3.5	444-684-07
Z-386/25	12/05/89	3.5	444-684-07
Z-386/33	12/05/89	3.5	444-684-07
Z-386/33E	11/13/90	3.5A	444-847-06
Z-486/25E	10/18/90	3.6C	444-886-02

Figure 1



Getting the Most From Your Computer

Part 4

John P. Lewis
6 Sexton Cove Road
Key Largo, FL 33037

One of the most thought provoking aspects of writing a series such as this one is the prospect of finding new material to write about. Quite a bit of research must be done to ensure that the article is as error free as is reasonable to expect and that the material is of interest to the readers. Several other factors must also be taken into account, such as matching the complexity level and the size or magnitude of the project to the projected audience.

Fortunately, I have a fairly large inventory of programs that I have written to draw from, but I'm constantly looking for new material. Several years ago I wrote a database program that proved to be quite popular and that I still use in several different configurations. It has evolved into a very useful utility, but one rather large difficulty presented itself toward using it as a future part of this series. It was written in "C".

Shortly after writing the "C" version I wrote another in Pascal but it was incompatible with the database created by the other program. In other words, it fell short of being as useful and versatile as its predecessor written in "C". I'm happy to report that I've just completed another database program that is even more versatile than its "C" predecessor and is also compatible with the existing database. Needless to say, the new creation is written in Turbo Pascal. It meets or exceeds the performance parameters of the "C" version. To say that I'm pleased with the result would be a masterpiece of understatement and yes, I do intend to include it as part of this series.

We will save the database program for a bit later in the series though — for several reasons, not the least of which is the increased code complexity as well as the

involved program logic. Quite a few new concepts will be presented and the best approach, in my view, is to learn programming techniques in bite sized pieces. If you follow this series closely you should find that software creation will soon become a very interesting, enjoyable task. There will be some stumbling blocks along the way, in addition to the frustration encountered in debugging a particularly stubborn routine, but — at least in my opinion — the feeling of euphoria after a successful computing session is difficult, if not impossible to obtain in any other field of endeavor.

The program presented in this month's listing is a calendar utility. The user must supply only two parameters, month and year, and the program will print an accurate calendar for the given month with the first day of the month falling on the correct day of the week. Think about that for a minute. A bit of magic? How does the computer know which day of the week that March 1, 1995 (or any other date) falls on? I would like to take credit for such a clever bit of logic but I'm afraid that it has been around a lot longer than I have. This logic gem is from "Zeller's Congruence", a scholarly work published in 1887. As you may have surmised, Mr. Zeller was a nineteenth century mathematician.

This program has gone through an evolutionary cycle, too. Its first implementation was written in "C" and was not capable of a hard copy printout. I rewrote it for the Turbo Pascal compiler some months ago, for inclusion in another, larger program which had its genesis in Pascal. In an effort to produce some material that could be distributed, advertising my photographic business, I added the hard copy printing module to the latest incarnation of

the program, which was then used to produce some appointment books. Aldus Pagemaker was employed to produce an attractive cover and some nature photos were added to embellish the facing page for each month. The resulting product was a big hit, gleaning quite a few compliments (also producing a lot of work). One of the books is presently sitting next to the computer on my desk, a handy reference.

Other, equally significant uses of our new program suggest themselves to me. We may delve into one or more of these (so far hypothetical) uses at a later date but for now let's do some ground work.

In today's world of really sophisticated software, no program will get more than a second glance unless its screen output is attractive. With that in mind, we'll create a grid to house the dates and use colors to dress up the presentation. The grid itself will be constructed using a variation of our friend "DRAWBOX.PAS" from part one of this series.

Our next consideration will be the actual design parameters. No mysteries here, just a few differences that occur from month to month that will present some contingencies to be dealt with by the code. To wit: The months may comprise 28, 29, 30 or 31 days, with February using 29 days every fourth (leap) year. We'll use a "look up table" for assigning values to the various months and a bit of more sophisticated logic to account for leap year.

The actual math involved in computing the week day (Sun - Sat) corresponding to the first day of any given month is a little involved but, like anything else, is not too tough if we take it a step at a time.

Let's take a look at the formula used for our computations. Our main concern is to

derive a number that yields a result modulo 7 corresponding to the numeric day of the week. We must use integer math for our computations since decimal fractions will not fit our application. Here's the formula:

```
Num = 1461 * F(year, month) / 4 + (153 * G(month)) / 5 + day
```

Further defined:

```
If month <= 2 then F(year, month) = year - 1  
else F = year
```

and

```
If month <= 2 then G(month) = month + 13  
else month = month + 1
```

The constant 621049 is then subtracted from "Num" and the result is operated on with Modulo 7 yielding our target number which can be used to construct a calendar.

Let's take a real world example. January 15, 1991 is a date that most of us will be able to remember for some time to come. We'll convert January to 1 and enter the formula:

```
Num = 1461 * F(1991, 1) / 4 + 153 * G(1) / 5 + 15  
Num = (1461 * 1990) / 4 + (153 * 14) / 5 + 15  
Num = 2907390 / 4 + 2142 / 5 + 15  
Num = 726847 + 428 + 15  
Num = 727290
```

Subtract 621049 (727290-621049) = 106241 Mod 7 = 2 or Tuesday (0 = Sunday). If you look at your calendar you will find that January 15, 1991 did, in fact, fall on Tuesday. The above math works for any date after February 28, 1900.

All that remains is to construct a program which implements this handy little algorithm. Let's take a look at the listing, CALNUNIT.PAS.

Notice the list of constants which contains the characters that make up the grid. The list is much longer than that used for the box from part 1 of this series. The characters include those that form a connection to others, thus adding to the agenda. If you have a catalog to ASCII characters it would be a good idea to look up the references to see how they fit together. Page 422 - 423 of the Turbo Pascal reference manual (Version 5.5) shows the entire gamut of characters.

The procedure "Hprint" and its cousin "Vprint" are both used within the larger procedure "drawbox" to draw the calendar grid. If you wish to dissect this procedure to get a better handle on the actual grid construction, place a "readln;" instruction at judicious points within the procedure and then recompile and rerun the program. The screen construction will stop at each "readln" instruction, allowing you to construct the grid a step at a time. Pressing "enter" or "return" (after the "readln" instruction) will allow the program to proceed. The aforementioned "readln" instructions will have to be removed after your dissection, but that is a very small task and one you should practice anyway; it is a very common course of action when debugging code. Let me add a helpful hint

for adding instructions to a program which will be deleted later. I learned this the hard way. Never "bury" such an instruction within other instructions; rather, create a separate line for the command. It is much easier to

go back later and delete an entire line than hunt for an errant "readln" within a large body of code.

Since there is a tendency for the user to provide alphabetic input when queried for the date, I have included a procedure that precludes getting the wrong kind of data (alphabetic instead of numeric). The function "Getvalue(s:string):Integer;" takes care of this. If the user inputs "March" instead of "3"; then the program aborts,

since it is not able to deal with a string in lieu of an integer. There are better, more sophisticated ways of coping with the wrong kind of data input but we will get to that a bit later in this series. We will create a window informing the user of his/her error and then reposition the cursor for corrected entry.

That brings us to "Procedure Execute_Caln;"; where we will begin construction of our calendar. After obtaining the month information from the user, we use a "look up table" through the implementation of "case mon of". We will assign a string indicating the alphabetic representation of the month and also the number of days in the month. Note that February is assigned a value of 28 with no provision made for leap year. Au Contraire, that is only a preliminary value which will be used for further computations in the actual execution of the algorithm.

After deriving the month information, the user is asked for the year (numeric). Notice that the boolean variable "flag" is set to zero at this point and then we encounter a rather lengthy "if" statement which incorporates a number of "mod" operators. The purpose of this complex "if" statement is to establish the leap year parameters. If all of the comparisons are true then our flag is made equal to one and an additional day is tacked on to February. The code to accomplish this takes the form: Max (days in month) :=Max+Flag; thus incrementing Max in the case of leap year.

Dropping down to the next "if" statement, we are applying one of the conditions which will determine the correct formula for use in our computation. The code fragment "if (mon <= 2) then begin yr:=yr-

1;mon:=mon+13;end else Mon:=Mon+1;" resolves this problem very nicely. You might want to study that line and then apply it to the formula to see how it works; however, we can let the computer do all the math for us by running the completed program with different dates for a check of its accuracy. You might want to run it, using your birthday as the date, to satisfy your curiosity. I have used it extensively and have never encountered any errors.

If you have the Turbo Debugger (integrated or stand alone version) you might want to run the program and examine the "Factor" variable using various input parameters (a quick education). I will cover, in a future part of this series, uses of the debugger, but space does not permit coverage of all the pertinent aspects of programming in one article (that would take the entire contents of several issues of REMark). Perhaps this program would make a very interesting candidate for a debugging session. By using the debugger to display a variable (watch mode) while stepping through the program a line at a time, much can be learned about the way in which a code fragment processes data. I am a very enthusiastic user of both the integrated and stand-alone versions of the Turbo Debugger.

The next task to be attended to is also the easiest to manage since we have completed the "ground work". The grid itself is drawn on the screen and then the days of the week (Sun through Sat) are inserted on the top line of the calendar grid. Notice the rather extensive use of "gotoxy(col,row);" to accomplish this. Next a "For" loop is used, in conjunction with the value assigned to "Factor" to print the day number within the appropriate grid squares. "Max" has already been assigned a value equal to the last day of the month, providing us with a convenient way to terminate the "For" loop.

The balance of the code listing is pertinent to "hard copy", printing only, primarily for the production of appointment books alluded to above. I have left it "as is" for your convenience. It is replete with instructions unique to its use with a Hewlett-Packard LaserJet printer as well as code which works with my daisy wheel printer (an old Radio Shack Mod 210). You might want to dress it up for use with your own printer, use it "as is", or possibly delete it if you consider it redundant.

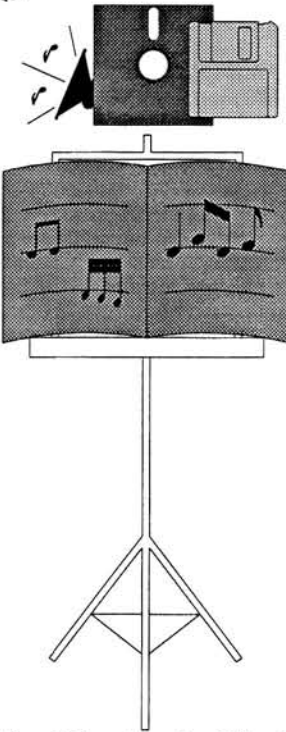
I'm sure you have noticed that the code listing "CALNUNIT.PAS" comprises the source code for a Turbo Pascal Unit, and not an executable (stand alone) program. The intended application of this code is for inclusion within the project program (Projct_1.pas, from part three of this series). A few rather insignificant additions are all that is required to accomplish this implementation. Add to the uses statement: "Calnunit", change the third line

of the menu to "3. Access calendar" and under the "case option of" statement, add "3:begin window(1,1,80,25); Execute_Caln;end;". The ease with which this program is included within its parent should serve to illustrate the reason for configuring the code as a unit rather than an executable program.

If you wish to derive the maximum benefit from this series and you have just started your subscription to "REMark", or for whatever reason you do not have the previous issues containing the first three installments of this series, I would strongly suggest obtaining them. We are building a Turbo Pascal library of source code while learning the basics of programming. Each part of the series builds, to some extent, on its predecessor. There is a lot of ground to cover and a complete inventory of the articles should be very helpful in the pursuit of becoming a proficient programmer in Turbo Pascal.

If you have any questions regarding this, or any of the articles which comprise this series, or even Turbo Pascal programming in general, please feel free to write me at the address given at the beginning of this article. Be sure to enclose a S.A.S.E. if you wish a reply. Feel free to make comments about the series and if you stipulate that your letter may be included in a future article, that would be helpful.

New Software Product



The Electronic Clavier
P/N 885-6016

CALNUNIT.PAS

```

Unit Calnunit;           { Source code for "Getting the Most From Your }
interface               { Computer, Part 4" }
Procedure Execute_Caln;
Implementation
Uses crt, Printer;
Label Again;

Const
  ULFT = #218;           { upper left character in calendar grid }
  URT  = #191;           { upper right }
  VERT = #179;           { vertical }
  LLFT = #192;           { Lower left character }
  LRT  = #217;           { lower right }
  RTCN = #180;           { right connection }
  LFTCN = #195;          { left connection }
  UPCN = #194;           { upper connection }
  BTCN = #193;           { bottom connection }
  HOR  = #196;           { horizontal character }
  MIDC = #197;           { middle connection }
  VSEP = 3;              { vertical space }
  HSEP = 10;             { horizontal space }

Type
  String10 = String[10];

Var
  Rt, Top, Lft, Bot, j, col, StorM, row, max, flag : Integer;
  Mon, Yr, Factor, Storage : longint;
  Month, Year : String10;
  Choice : Char;
  Sp, I, Place, s, Tb, TabLJ : Byte;

Procedure Hprint(col, row : Integer); { prints grid characters across }
Var ColTmp, i, j : Integer;          { the screen (horizontal) }
begin
  ColTmp:=col;gotoxy(col,row);write(LFTCN);coltmp:=coltmp+1;
  for i := 0 to 6 do
  begin
    For j := 0 to HSEP - 1 do
    begin
      gotoxy(coltmp,row);write(HOR);ColTmp:=Coltmp+1;
    end;
    gotoxy(coltmp,row);write(MIDC);Coltmp:=coltmp+1;
  end;
  gotoxy(coltmp-1,row);write(RTCN);
end;

Procedure Vprint(Col,Row : Integer); { prints grid characters vertically }
Var i, j, Coltmp : Integer;
begin
  gotoxy(col,row);coltmp:=col;
  for j:=0 to 7 do
  begin
    for i := row to (row+(VSEP*6)-1) do
    begin
      gotoxy(coltmp,i);write(VERT);
    end;
    Coltmp:=Coltmp+HSEP+1;
  end;
end;

Procedure Tab(Spaces : Byte);        { allows sending spaces to the printer }
Var s : Byte;                        { for formatting purposes }
Begin
  For s:=1 to spaces do
  Write(Lst,' ');
end;

Procedure Lines;                     { This code is used to implement the }
Var s : Byte;                         { appointment book, delete along with }
begin                                  { all references if you wish }
  for s:= 1 to 2 do begin
    Tab(TabLJ);
    writeln(Lst,' _____ ',#10);
  end;
end;

Procedure drawbox(UplftX,UplftY,LowrtX,lowrtY:integer);
Var i, j, r : Integer;                { a modified version of the }
begin                                  { drawbox utility (from part 1) }
  gotoxy(UplftX,UplftY);write(ULFT);
  for j:= 0 to 6 do
  begin
    for i:= 0 to HSEP-1 do

```

CALNUNIT.PAS (Cont'd.)

```
end;
gotoxy(Rt,Top);write(URT);
Vprint(lft,Top+1);Hprint(lft,Top+VSEP);
Top:=top+VSEP;Hprint(Lft,Top+VSEP);
Top:=Top+VSEP;Hprint(Lft,Top+VSEP);
Top:=Top+VSEP;Hprint(Lft,Top+VSEP);
Top:=Top+VSEP;Hprint(Lft,Top+VSEP);
Top:=Top+VSEP;gotoXY(Lft,Top+VSEP);
write(LLFT);
for j:= 0 to 6 do
begin
for i:=0 to (HSEP-1) do
begin write(HOR);end;
write(BTCN);
end;
write(#8);write(LRT);
end;

Function Getvalue(s : String10):Integer;
var num, code : Integer;
begin
val(s,num,code);Getvalue:=num;If code <> 0 then begin clrscr;
gotoxy(4,4);write('Input must be numeric !, aborting program. ');Halt;end;
end;

Procedure Execute_Caln;
Label again;
begin
lft:= 2; top := 2; rt := 79;
Clrscr;gotoxy(4,4);
write('Please enter the month (Numeric) for calendar display. ');
readln(Month);mon:=Getvalue(Month);
case mon of
1:begin Month:='Jan';Max:=31;end;
2:begin Month:='Feb';Max:=28;end;
3:begin Month:='Mar';Max:=31;end;
4:begin Month:='Apr';max:=30;end;
5:begin Month:='May';Max:=31;end;
6:begin Month:='Jun';Max:=30;end;
7:begin Month:='Jul';Max:=31;end;
8:begin Month:='Aug';Max:=31;end;
9:begin Month:='Sep';Max:=30;end;
10:Begin Month:='Oct';Max:=31;end;
11:begin Month:='Nov';Max:=30;end;
12:begin Month:='Dec';Max:=31;end;
end;
gotoxy(4,6);write('Please enter the year (Numeric) ');
readln(Year);yr:=getvalue(year);flag:=0;
if ((yr mod 4 = 0) and (yr Mod 100 <> 0) and (Mon = 2)) or (yr mod 400 = 0)
and (mon = 2) then flag:=1;
if (mon <= 2) then
begin
yr:=-yr-1;Mon:=Mon+13;
end else
Mon:=Mon+1;
Factor:=((1461*yr) div 4) + ((153 * mon) Div 5) + 1;
Factor:=Factor - 621049;Factor:=Factor Mod 7;clrscr;
gotoxy(35,1);write(Month,' ',year);
drawbox(Lft,Top,Rt,(6*VSEP)+4);gotoxy(7,3);write('Sun');
gotoxy(17,3);write('Mon');gotoxy(28,3);write('Tue');gotoxy(39,3);
write('Wed');gotoxy(49,3);write('Thur');gotoxy(61,3);write('Fri');
gotoxy(72,3);write('Sat'); row:=4;Factor:=Factor+1;Max:=Max+flag;
Storage:=Factor;StorM:=Max;
For I := 1 to Max do
begin
if (factor * 10) > 75 then
begin
Factor:=1;Row:=Row+VSEP;
end;
gotoxy((Factor * 11)-3,row);
write(i);Factor:=Factor+1;
Tb:=0;TabLJ:=0;
end;
gotoxy(10,23);write('Do you wish a hard copy y/n ? ');readln(choice);
If Uppcase(choice) = 'Y' then Begin gotoxy(10,24);write('LaserJet y/n ? ');
readln(choice);if Uppcase(choice) = 'Y' then TabLJ:= 4;
again;
Tb:=TabLJ;
Factor:=Storage;write(Lst,#27,#21);{ defeat double spacing on Tandy printer }
Tab(Tb);
Tab(TabLJ);writeln(Lst,' ',Month,' ',year);Tab(tb);
writeln(Lst,' Sun Mon Tue Wed Thr Fri Sat ');
If (factor * 10) > 75 then
Factor:=1;
case factor of
```

Quality Enhancements!

EaZy PC Products

EZM-128: Expand 512K base memory to 640K. Simple, plug-in installation. \$95.00

EZCLOCK: Calendar/Clock. Piggy-back add-on for EZM-128. \$35.00

No Slot Clock/Calendar

FBE SmartWatch: Automatic date/time on bootup. Installs under BIOS/Monitor ROM. Ten year battery. Works with all Heath/Zenith MSDOS computers. For PC's \$32.00, Z-100 \$33.00 Module: \$25.00

H/Z-148 Expansions

ZEX-148: Adds one full-size and one half-size expansion card slot. \$79.95

ZP-148: PAL chip expands existing 640K memory to 704K. CGA/MDA only! \$19.95

Configuration Control

CONFIG MASTER: Menu-select active CONFIG.SYS during bootup. Software for PC/Z-100 MSDOS. \$29.95

H/Z-150 Items (Not for '157, '158, '159)

VCE-150: Eliminate video card. Install EGA or VGA card. All plug in. Includes circuit board, SRAM and RM-150. \$49.95

ZP640 PLUS: PAL chip to expand stand-ard memory card to 640/704K with 2 banks of 256K RAM chips (not included). \$19.95

ULTRA-PAL: Three PAL chips: MR150 for 704K + 512K RAM Disk; MR150T for 640K + 512K RAM Disk; LIM150 for 640K + 512K (32 pages) of simulated v3.2 Lotus/Intel/Microsoft Expanded Memory. With software. Install on standard memory card. No soldering. Needs 45 256K RAM chips (not included) for maximum memory configuration. \$39.95

COM3: Change existing COM2 to COM3. Put internal MODEM at COM2. Don't lose serial port. With software. \$29.95

H/Z-100 Modifications

ZMF100A: Expand "old" motherboard (p/n 181-4917 or less) using 256K RAM chips (not included). No soldering. \$65.00

ZRAM-205: Put 256K RAM chips on your Z-205 board. Get 256K plus 768K RAM disk. Contact us for data sheet before ordering. Without RAM chips. \$39.00

H/Z-89 Add-Ons

H89PIP: Parallel printer 2 port interface card. With software. \$50.00 Cable \$24.00

SLOT4: Add fourth expansion slot to right-side accessory bus. \$39.95

Order by mail, FAX, telephone, or see your dealer. UPS/APO shipping included. VISA/MasterCard. WA residents add 8.1% tax. Hours: M-F 9-5 PST. We return all calls left on our answering machine!

FBE

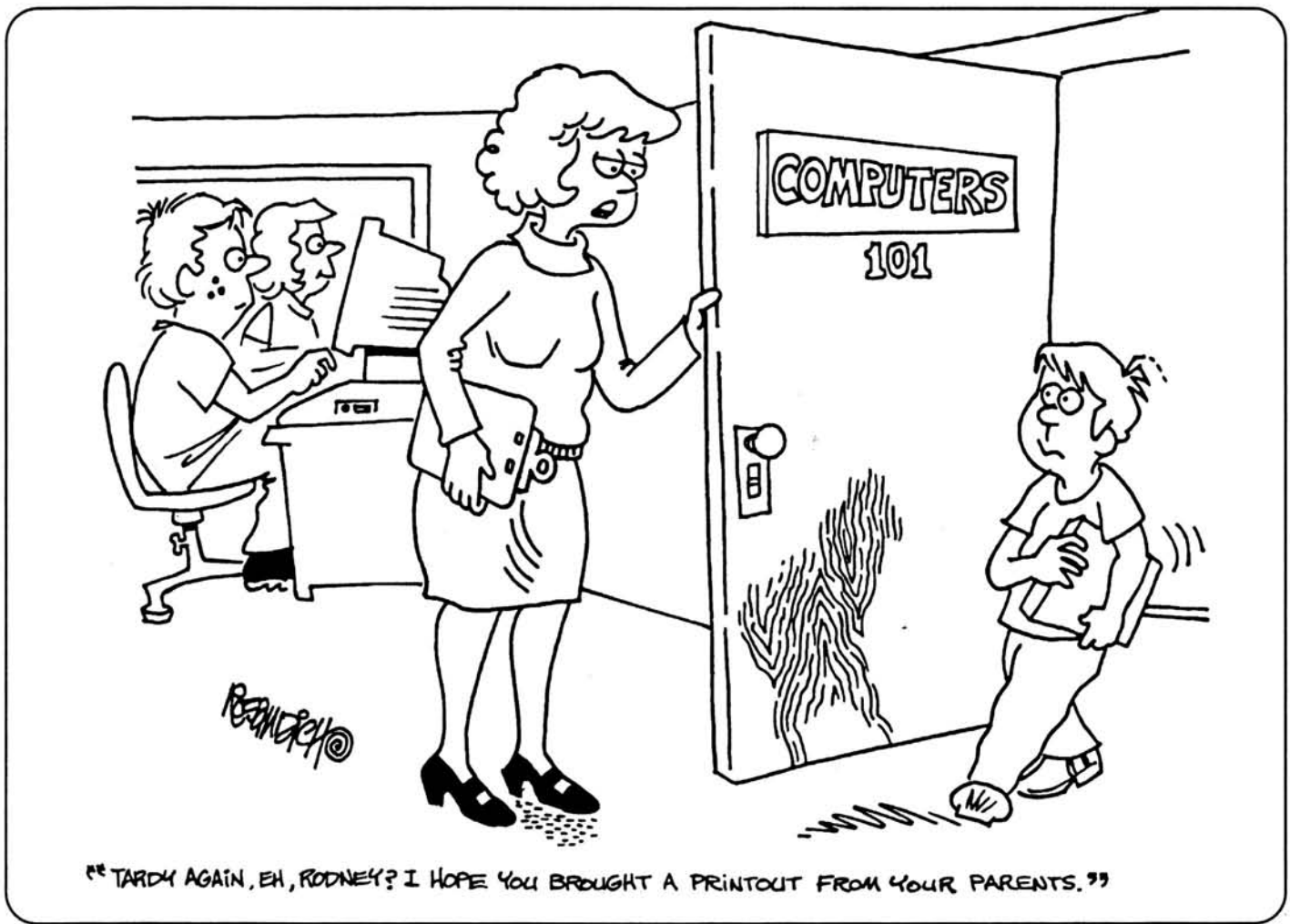
FBE Research Company, Inc.

P.O. Box 68234, Seattle, WA 98168

206-246-9815 Voice/FAX TouchTone Selectable

CALUNIT.PAS (Cont'd.)

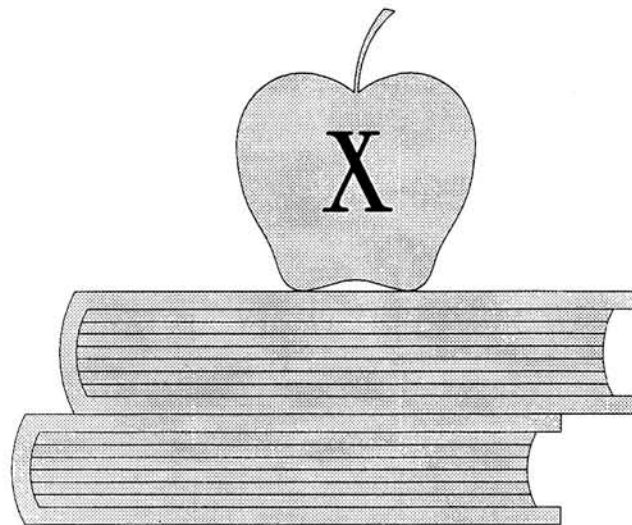
```
1:Tab(0);
2:Tab(4);
3:Tab(8);
4:Tab(12);
5:Tab(16);
6:Tab(20);
7:Tab(24);
end;
J:=1;Sp:=3;Factor:=Factor * 3;
For J:-1 to StorM do
begin
  Tab(Sp+tb);Write(Lst,J);Factor:=Factor+3;Tb:=0;
  if J > 8 then begin SP:=-2;end;
  If Factor > 21 then begin writeln(Lst);Factor:=-1;Tb:=TabLJ;end;
end;
Writeln(Lst,#10,#10);
Tab(TabLJ);writeln(Lst,'Sun _____',#10);
Lines;
Tab(TabLJ);Writeln(Lst,'Mon _____',#10);
Lines;
Tab(TabLJ);writeln(Lst,'Tue _____',#10);
Lines;
Tab(TabLJ);Writeln(Lst,'Wed _____',#10);
Lines;
Tab(TabLJ);Writeln(Lst,'Thur _____',#10);
Lines;
Tab(TabLJ);Writeln(Lst,'Fri _____',#10);
Lines;
Tab(TabLJ);Writeln(Lst,'Sat _____',#10);
Lines;writeln(Lst,#12);
gotoxy(1,24);clreol;write('Copy y/n ? ');readln(choice);
if Ucase(choice)='Y' then goto again;
end;
end;
end.
```



Introduction to C++

Tenth Installment

Lynwood H. Wilson
2160 James Canyon
Boulder, CO 80302



I have always subscribed to the theory that you learn more from trouble than you do when everything is going along fine. You may not like it much, but you learn more. My main project these days involves writing a menu system with a large number of complex and interrelated data entry screens. The specifications are even more meager than usual and the client is busy with other things and can't spend a lot of time on this project. This is not the world of structured programming you read about in the textbooks.

The only approach that I know works in this situation is to write what you think the client needs and get it running as quickly as possible, perhaps with a few of the complex parts replaced by plausible stubs. Then show it to him and see what he thinks. Repeat as necessary. As I go along I am writing what specs I can from what he has told me and from what he has liked about previous versions.

I am finding that it goes a lot easier in C++ than in C. I am also building up a library of classes for user interfaces, menus and data entry screens, that I expect will make it even easier in the future.

Resources

I have switched to the new compiler from Borland which they call Borland C++ Version 2.0. The word is that this will be the professional product from now on and Turbo C++ will be the educational and entry level compiler. Borland C++ will produce Windows code but I don't write for Windows and I don't see much obvious change from Turbo C++ except the price. My advice would be to wait for the next version unless you are working on large programs. In that case Borland C++ seems

to offer some advantages in compile time and such.

Borland has a very active forum on CompuServe staffed with their people to answer questions and discuss problems. If you can't get an answer to your problem on COM1, our own board, try CompuServe.

I've just found a book by Gerry Weinberg (and Donald Gause) that's excellent. It's called "Are Your Lights On?" and it's about figuring out what the problem REALLY is. I think Mr. Weinberg is a genius and I recommend all his books but this one is even better than average.

One of the points which I found particularly relevant is the idea that "...people seldom know what they want until you give them what they ask for."

Structures

Structures are similar to arrays in that they contain more than one data item. Unlike arrays, a structure can contain data items of different data types. They are widely used in their own right as well as forming the basis for objects which we will get to shortly.

Suppose we were writing a program to keep personnel records. It would be most inconvenient to create separate variables for each aspect of each employee. We could not combine them into an array because they would be of different types. So we create a structure to hold the data we wish to maintain for each employee. Here is a declaration for such a structure:

```
struct employee {
    char name[81];
    int emp_num;
    float hr_pay;
};
```

This declaration creates a new data type, the structure employee. It does not

create an instance of such a structure, just the type. No memory is allocated by the declaration. In order to create an instance, an actual structure of this type, and allocate memory for its data, we define a new variable very much like we define a variable of the old types like int or char. In C it's done like this:

```
struct employee emp1;
```

Here we come upon a slight difference between C and C++. That definition of emp1 will work fine in C++, but so will this one:

```
employee emp1;
```

C needs to be told that employee is a struct and C++ does not. In this way (and other ways which we will see) C++ tries to deal with the data types you define just like it treats the built in data types such as int and char.

Examples of the new structure may also be created as part of the declaration of the type, like this:

```
struct employee {
    char name[81];
    int emp_num;
    float hr_pay;
} emp1, emp2;
```

The elements of the structure such as the name are called fields. A field of a structure is identified by the structure name, a period, and the field name. Thus we can say:

```
strcpy(emp1.name, "Bilbo Baggins");
emp1.emp_num = 1;
```

and we have set the name field in the structure emp1 to Bilbo Baggins and the emp_num field to the value 1. The field called name in the structure called emp1, which we address as emp1.name, is an ordinary array of chars and the field called emp1.emp_num is a simple integer variable in all ways except the names we call

them and the fact that they are each somehow connected to some other variables.

We can initialize a structure as part of its definition and then print it to the screen like this:

```
#include <iostream.h>

struct employee {
    char name[81];
    int emp_num;
    float hr_pay;
};

void main(void)
{
    employee emp1 = {"Bilbo Baggins", 1,
1.23};

    cout << emp1.name
        << ", employee number "
        << emp1.emp_num
        << ", makes $"
        << emp1.hr_pay
        << " an hour.";
```

Note that the declaration of the structure type is global. This is normal practice as some other code in the program may need to know about the structure, and there is no reason not to do it that way. The definition of the struct emp1 is local for all the reasons that local variables are preferable. (See installment 5 if you've forgotten.)

The initial values in the curly braces go into the fields of the structure in the order they are written and must be of the correct data types.

The value of a structure may be assigned to another structure of the same type just as though they were simple variables. Each field of the structure on the left gets the value of the corresponding field of the structure on the right. Here is an example:

```
#include <iostream.h>

struct employee {
    char name[81];
    int emp_num;
    float hr_pay;
};

void main(void)
{
    employee emp1 = {"Jonathon Hoag",
1, 1.23}, emp2;

    emp2 = emp1;
    cout << emp2.name
        << ", employee number "
        << emp2.emp_num
        << ", makes $"
        << emp2.hr_pay
        << " an hour.";
```

In this example the structure emp1 is defined and initialized. Its value is then assigned to the structure emp2, which is then printed.

You can see how this would simplify a personnel program, but there will still be a considerable inconvenience in declaring each employee as a separate variable. The solution is to declare an array of employee structures. Like this:

```
#include <conio.h> #include <iostream.h>
```

```
struct employee {
    char name[81];
    int emp_num;
    float hr_pay;
};

void main(void)
{
    employee emp[5];
    for(int i = 0; i < 5; i++) {
        cout << "Name: ";
        cin >> emp[i].name;
        cout << "Number: ";
        cin >> emp[i].emp_num;
        cout << "Hourly pay: ";
        cin >> emp[i].hr_pay;
    }
    int n;
    while((n = getch() - '0') >= 0 && n
        <= 5) {
        cout << emp[n].name
            << ", employee number "
            << emp[n].emp_num
            << ", makes $"
            << emp[n].hr_pay
            << " an hour.\n";
    }
}
```

This program declares an array of employee structures, runs a loop to get data from the user to fill them, and then gets numbers from the keyboard and prints the data on that number from the array.

The only tricky bit is the test for the while loop where a char is read in from the keyboard and converted to an int by subtracting the char '0' from it. If you look for a moment at a table of ASCII values of characters, you will see how this works. If getch gets the char '0' and we subtract the char '0' from it, no matter what number represents the char '0' the answer will be the number 0. Since the ASCII values are sequential, subtracting '0' from '1' will give the number 1 and so forth. (This may not work in systems other than ASCII.)

This program could benefit from being divided into a few well chosen functions, which process could clarify its structure by separating the control from the data input and output. Like this, for instance:

```
#include <conio.h> #include <iostream.h>

struct employee {
    char name[81];
    int emp_num;
    float hr_pay;
};

void get_emp_data(employee emps[], int
num); void
write_emp_data(employee this_emp);

void main(void)
{
    employee emp[5];
    for(int i = 0; i < 5; i++)
        get_emp_data(emp, i);

    int n;
    while((n = getch() - '0') >= 0 && n
        < 5)
        write_emp_data(emp[n]);
}

void get_emp_data(employee emps[], int
num)
{
    cout << "Name: ";
    cin >> emps[num].name;
```

```
    cout << "Number: ";
    cin >> emps[num].emp_num;
    cout << "Hourly pay: ";
    cin >> emps[num].hr_pay;
}
```

```
void write_emp_data(employee this_emp)
{
    cout << this_emp.name
        << ", employee number "
        << this_emp.emp_num
        << ", makes $"
        << this_emp.hr_pay
        << " an hour.\n";
}
```

Here I have separated the input and output from the main body of the program in standard structured programming fashion. Note that I passed an employee structure to writ_emp_data() just like any other variable. And, just like any other variable, the function got the value, not the original structure. That is why I passed the array of structures and the number of the one I was filling to get_emp_data(). Since arrays are passed by reference, the function could put its data into the actual structure in main(). But it is a little awkward to pass the whole array and an index number if we are interested in only one element of the array. How about passing a pointer to the structure we are interested in? Then the function could put data in the structure and it would be available in main().

A pointer to a structure is obtained by putting an ampersand (&) before the name of the structure, just as with other data types. Referencing the fields of a structure is only slightly more complex, and you can do it either of two ways. The first uses the same syntax we are used to.

```
employee emp;
employee *pe = &emp;
(*pe).emp_num = 3;
```

We create a structure called emp and a pointer to it and then fill its emp_num field. *pe is, as always, the thing pe points to so it can be substituted for emp and the dot and field are just as usual. The parentheses around *pe are required because the dot operator has higher precedence than the asterisk. Thus without the parentheses we would be referring to the thing that pe.emp_num points to which is not what we mean at all.

Because the pointer to a structure is widely used there is another way to dereference it which is just a bit simpler. Given the same definitions as above, these two statements are identical:

```
pe->emp_num = 3;
(*pe).emp_num = 3;
```

I think of this arrow notation as saying "the emp_num which is a field of the structure that pe points to".

To pass the structure to the get_data function we change the function call to this:

```
get_emp_data(&emp[i]);
and the function itself to this:
void get_emp_data(employee *emps)
{
    cout << "Name: ";
```

```

cin >> emps->name;
cout << "Number: ";
cin >> emps->emp_num;
cout << "Hourly pay: ";
cin >> emps->hr_pay;
}

```

The function prototype must also reflect this change in parameters, of course.

Classes

We are going to begin encapsulating code with our data, one of the central ideas behind OOP. We could do it with structures but I am switching to classes instead as they are more widely used in C++. The only difference between classes and structures is that in structures the default is that everything, data and functions, is available from outside the structure. In a class the default is that everything is only available inside the class. In either case the access rules can be modified, but the class is a bit more convenient.

We can encapsulate the input and output functions in the class with the data like this:

```
#include <conio.h> #include <iostream.h>
```

```

class Employee {
    char name[81];
    int emp_num;
    float hr_pay; public:
    void get_emp_data(void);
    void write_emp_data(void);
};

```

```

void Employee::get_emp_data(void)
{
    cout << "Name: ";
    cin >> name;
    cout << "Number: ";
    cin >> emp_num;
    cout << "Hourly pay: ";
    cin >> hr_pay;
}

```

```

void Employee::write_emp_data(void)
{
    cout << name
        << ", employee number "
        << emp_num
        << ", makes $"
        << hr_pay
        << " an hour.\n";
}

```

```

void main(void)
{
    Employee emp[5];

    for(int i = 0; i < 5; i++)
        emp[i].get_emp_data();

    int n;
    while((n = getch() - '0') >= 0 && n
          < 5)
        emp[n].write_emp_data();
}

```

I have capitalized Employee because it is my convention, and that of many others, to distinguish the names of classes with a capital letter.

The prototypes for the two functions are moved into the class declaration after the word public. The data, since it is before the word public, defaults to private. That means that no code which is not part of the Employee class can get at the data. The

functions, since they follow the word public, are visible from outside.

The functions no longer take parameters. They automatically use the data in the object they belong to. It's a little like having the advantages of global variables, since the functions can see the data without having it explicitly passed to them. We avoid the disadvantages of globals, however, by being able to control which functions can see the data. And the flow of data through the program can be made quite clear, as we will see.

I moved main() to the end of the program to get the class declaration all in one place. It makes no real difference. Normally, in a large program, the declaration of the Employee class will be in a .hpp file so it can be #included in all the other files that use the class. The functions will be defined in their own .cpp file and the rest of the program, main() in this example, will be in another .cpp file or files.

The function definitions include Employee:: before the function name. This shows that they are member functions in the class Employee. It is possible, and even likely, for there to be functions with the same name which are members of different classes. Incidentally, a few writers call these member functions methods, carried over from Smalltalk.

This completes the declaration of the class Employee. Remember that a declaration describes something and a definition actually creates one and allocates memory for it. We have described a class called Employee, but we have not created one. In other words, we have not yet created an instance of the class Employee.

In the third line of main(),
Employee emp[5];
we create an array of five Employee objects and the array is called emp. Note that the kind of thing is called a class and the thing itself is called an object. This is similar to saying that i in the program above is a variable but the kind of variable it is (integer in this case) is called a data type. In this program emp[2] is an object, of type Employee.

In a sense we have created a new data type, the Employee, and defined the operations which are legal for it.

In main(), when we want to call a function which is a member of one of the objects, we do it very much like we get a field of a structure. The name of the function is the name of the object followed by a dot and then the member function name followed by parentheses, just like any other function.

Note the sixth line in main():

```
emp[i].get_emp_data();
```

We are saying "Go to the object emp[i] and execute its member function get_emp_data()." The function will use the variables of emp[i] of course, instead of those of any other Employee object, so we

do not have to specify them.

Another way of looking at this is that we are saying to the object emp[i] "Go get the data to put in your variables. You figure out how." As you will see, it is very convenient for variables to know how to do things like that.

Constructors

When an object is created, a function called a constructor is executed. If we do not write a constructor for a class of object the compiler provides a default constructor to handle the housekeeping. But if we would like to do any particular thing whenever an object is created, we can write our own constructor.

One of the things a constructor might do is get the data to fill an object's variables. It is good practice, which C++ helps in many ways, to initialize variables at the time they are created. This avoids the common problem of using vars which have no useful data in them.

Here is another version of our program in which we use a constructor to put the data into the objects.

```
#include <conio.h>
#include <iostream.h>
```

```

class Employee {
    char name[81];
    int emp_num;
    float hr_pay; public:
    Employee(void);
    void get_emp_data(void);
    void write_emp_data(void);
};

```

```

Employee::Employee(void)
{
    get_emp_data();
}

```

```

void Employee::get_emp_data(void)
{
    cout << "Name: ";
    cin >> name;
    cout << "Number: ";
    cin >> emp_num;
    cout << "Hourly pay: ";
    cin >> hr_pay;
}

```

```

void Employee::write_emp_data(void)
{
    cout << name
        << ", employee number "
        << emp_num
        << ", makes $"
        << hr_pay
        << " an hour.\n";
}

```

```

void main(void)
{
    Employee emp[5];
    int n;
    while((n = getch() - '0') >= 0 && n
          < 5)
        emp[n].write_emp_data();
}

```

The constructor has the same name as the class itself. It is executed automatically whenever an object of this class is created. Thus when we create the array of 5 Employee objects the constructor for each of them is executed and fills its variables. We

can still call `get_emp_data()` somewhere else in the program and change the data in one of the objects if we like.

The constructor does not return a value and cannot be declared with a data type in front of it, not even `void`.

Constructors can take parameters, and thus can be used to initialize their objects if you have the data at compile time, rather than waiting until run time.

```
#include <conio.h>
#include <iostream.h>
#include <string.h>

class Employee {
    char name[81];
    int emp_num;
    float hr_pay; public:
    Employee(void);
    Employee(char *new_name, int new_num,
              float new_pay);
    void get_emp_data(void);
    void write_emp_data(void);
};

Employee::Employee(void)
{
    get_emp_data();
}

Employee::Employee(char *new_name,
                   int new_num, float new_pay)
{
    strcpy(name, new_name);
    emp_num = new_num;
    hr_pay = new_pay;
}

void Employee::get_emp_data(void)
{
    cout << "Name: ";
    cin >> name;
    cout << "Number: ";
    cin >> emp_num;
    cout << "Hourly pay: ";
    cin >> hr_pay;
}

void Employee::write_emp_data(void)
{
    cout << name
         << ", employee number "
         << emp_num
         << ", makes $"
         << hr_pay
         << " an hour.\n";
}

void main(void)
{
    Employee emp("Tom Paine", 0, 123.45);
    emp.write_emp_data();
}
```

I took the array out of this one and made it a single object to show the syntax for the parameters to the constructor more clearly. Note that the third line in `main()` creates an object and passes to its constructor the parameters in the parentheses even though the constructor is never specifically mentioned.

There are two constructors in the class declaration with different parameter lists. Constructors may be overloaded like most other functions. The one which corresponds to the actual parameters in the program is called.

If we still want an array of employees, here is how it is done.

```
#include <conio.h>
#include <iostream.h>
#include <string.h>

class Employee
{
    char name[81];
    int emp_num;
    float hr_pay; public:
    Employee(void);
    Employee(char *new_name, int new_num,
              float new_pay);
    void get_emp_data(void);
    void write_emp_data(void);
};

Employee::Employee(void)
{
    get_emp_data();
}

Employee::Employee(char *new_name,
                   int new_num, float new_pay)
{
    strcpy(name, new_name);
    emp_num = new_num;
    hr_pay = new_pay;
}

void Employee::get_emp_data(void)
{
    cout << "Name: ";
    cin >> name;
    cout << "Number: ";
    cin >> emp_num;
    cout << "Hourly pay: ";
    cin >> hr_pay;
}

void Employee::write_emp_data(void)
{
    cout << name
         << ", employee number "
         << emp_num
         << ", makes $"
         << hr_pay
         << " an hour.\n";
}

void main(void)
{
    Employee emp[5] = { Employee
                      ("Tom Paine", 0, 123.45),
                      Employee("John Galt", 2, 321.00)};

    int n;
    while((n = getch() - '0')
           >= 0 && n < 5)
        emp[n].write_emp_data();
}
```

There is data here to initialize only two of the objects, so the other three will invoke the version of the constructor which takes no parameters and call for data from the user. Note that in the case of an array of objects with multiple parameters to the constructor, the constructor must be invoked explicitly.

The member functions of a class can be defined inline in the class declaration instead of being declared there and defined elsewhere. Note the first constructor below which contains the body of the function and, therefore, is the definition and the declaration in one. This should be limited to very short functions as such a function will be inline in your finished code rather than being called like normal functions. There is some invisible overhead in a constructor which can make it bigger than you thought. When in doubt, and if it matters, write it both ways and check the code size and speed.

The example below will replace the class declaration and the definition of the first constructor (with no parameters) in the program above.

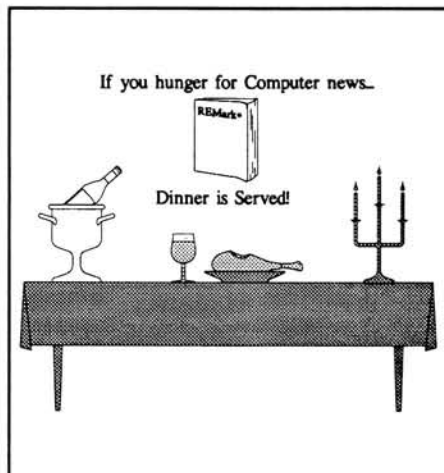
```
class Employee {
    char name[81];
    int emp_num;
    float hr_pay; public:
    Employee(void) { get_emp_data(); }
    Employee(char *new_name, int new_num,
              float new_pay);
    void get_emp_data(void);
    void write_emp_data(void);
};
```

Sources

Borland C++ \$495
 (Includes debugger, profiler, assembler,
 and the Whitewater Resource Toolkit for
 Windows.)
 Borland International Inc.
 P.O. Box 660001
 Scotts Valley, CA 95067-0001
 (800) 331-0877

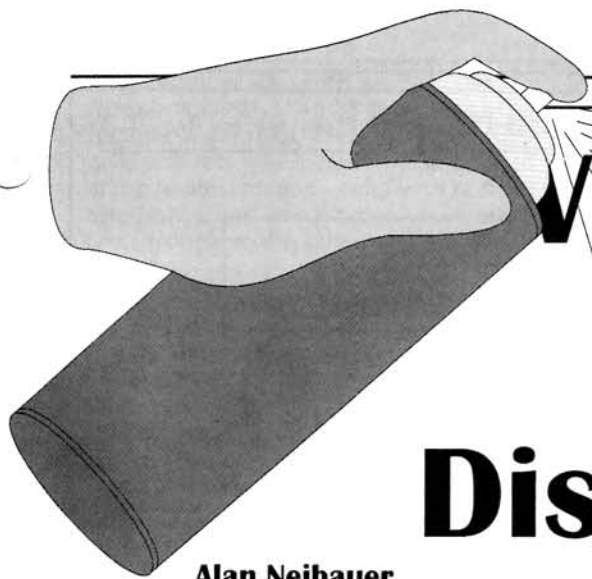
Are Your Lights On?

Donald C. Gause & Gerald M. Weinberg
 Dorset House, 1990 ✱



CAlassified ds

Z-100 LOW PROFILE, 768 KB, 2 5.25"
 Drives, color chips with mono monitor,
 needs power supply. Fortran, Pascal,
 DOS CP/M, much engineering utility,
 and game software. \$500 neg. Call
 Steve Ellingson, (404) 860-3123.



Virus Protection and Disaster Recovery

Alan Neibauer
11138 Hendrix Street
Philadelphia, PA 19116

A hard disk crash is a traumatic event. No matter how prepared you are, recovering from a disk crash is time-consuming and, if your backup system is not perfect, you may never get your system exactly as it was before the disaster.

Some computer consultants warn you to expect a crash, that they are inevitable just like death and taxes. Their warning is aimed at convincing you to plan a disaster recovery system so you are as prepared as possible if the event does occur.

In years gone by, I've had disk crashes on several minicomputer systems. Yet in all of these years I've never had a hard disk crash on a PC. Maybe it's been luck, maybe preventive medicine, but I've taken these warnings to heart and looked at as many disaster recovery techniques as possible.

The goal of disaster recovery is to return your hard disk to its original pre-crash condition as quickly and as accurately as possible. Chances are, unfortunately, that you might lose something. If you're lucky, it will be just whatever you're working on when the crash occurs. But on the dark side, you could lose a lot more if your recovery plan is inadequate, or if you fail to implement your plan correctly.

In this article, I'll discuss some disaster recovery methods, including protecting your system and cleaning it of computer viruses. I'll also outline my own disaster prevention methods that have worked so far.

Types of Protection

There are four levels of protection that you can employ:

1. System file (boot sector and partition table) recovery
2. Virus protection
3. File Backup
4. Preventive disk handling techniques

Each method serves its own purpose and by itself is not a completely effective

system. Some combination is needed for optimal performance.

System File Recovery

Some disk crashes only damage the boot sector, partition table, or directory. By backing up these sections of your disk, you may be able to recover the hard disk without restoring all of your files. This can save a great deal of time because you'll have access to the exact files that existed on your disk before the crash.

While DOS doesn't offer this type of protection, many third-party utility programs do. PC TOOLS, for example, comes with the programs MIRROR and REBUILD. MIRROR makes a backup copy of the File Allocation Table (FAT) and directory on a hidden, unused, portion of your hard disk. The section is marked as used so DOS doesn't overwrite it in normal operations. By running MIRROR regularly, such as just before you turn off your computer, you have an up-to-date record of these sections of the disk. There is also an option that saves the partition table on a floppy disk for more serious recovery.

If you accidentally reformat the hard disk, REBUILD may be able to restore the disk to its original condition. It searches the disk for the hidden file containing the FAT and directory, then writes them to the proper location. If you have a more serious crash, you can reboot from a floppy disk and use REBUILD to restore the partition table as well.

MACE Utilities takes a similar approach. Its program RXBAK makes a backup copy of your system's boot sector, FAT, and directory in a file called BACKUP.M_U. The next time you run RXBAK, it renames the file OLDBACK.M_U, and saves the current information in the BACKUP.M_U file. This always gives you two copies of the critical files for safekeeping.

Now suppose you accidentally refor-

mat the hard disk. Just boot your computer with a floppy disk, insert the Mace Utility disk that contains the program UNFORMAT, then enter UNFORMAT C:. That's all there is to it. The boot sector, FAT, and directory are automatically restored.

Unfortunately, some versions of FORMAT erase more than the FAT and directory — they actually erase the files on the disk. To get around this, MACE includes a replacement FORMAT program called FORMATH. FORMATH leaves the files intact so they can be restored using the UNFORMAT program. (A version called FORMATF is also supplied to format floppy disks so they may be restored as well.)

Why are two copies of the backup file needed? Let's assume only one is created. Now suppose you erase an entire subdirectory then reboot your system, or a disk crash occurs that damages parts of the disk but allows it to boot. If AUTOEXEC.BAT runs RXBAK automatically, the BACKUP.M_U file will now include the directory after the files were deleted, or a copy of the damaged FAT.

With the backup file, UNFORMAT or MACE's UNDELETE programs will search for OLDBACK.M_U to restore the disk completely.

Of course, if a system crash or virus damages an application file itself, restoring the boot sector, FAT, partition table, and directory will not restore the file. So in addition to system file recovery, you should also have a file backup that we'll discuss later.

Virus Protection

Backing up the system files may not help if they have been infected by a computer virus. If an infected boot sector, FAT, or partition table is saved and restored, the system will only be reinfected and crash again. You'd have to install a fresh unaltered system to correct the situation.

The same applies to application files that have been infected. The only safe way to recover an infected application file is to delete it and reinstall the program from a known uninfected source, such as the original distribution floppy disk.

On one hand, let's not panic about computer viruses. Using common sense, you are likely to never infect your system. But viruses are real and can cause damage.

Viruses enter a system in a variety of ways. The most common routes are through networks and infected programs on floppy disks.

A virus is a computer program that attaches itself somewhere in your system. Using a variety of mechanisms, it gains control to do the damage or mischief for which it was intended.

A boot sector virus, for example, infects the boot sector of your hard disk and loads itself into memory when you start your computer. Some newer versions of these have the ability to bypass DOS itself making themselves very difficult to detect.

ARAM virus loads itself in your memory and usually watches the interrupt calls to DOS. Unknown to you, they transmit their own requests to DOS causing strange I/O operations.

Program viruses attack application programs. They add code onto programs or alter their operations.

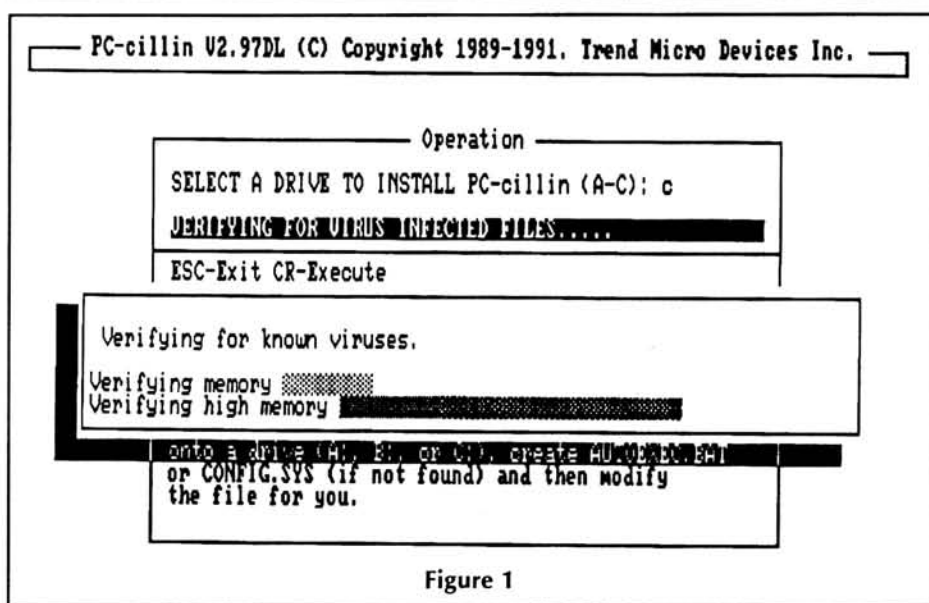
Because of the way viruses operate, you can unknowingly spread them to other users. Most work in the background, silently infecting your programs. At some point in time the virus is triggered, or activated, to cause its damage. But you may not realize a program is infected when you copy it to share with another individual.

Luckily, there is a wide variety of virus protection solutions on the market. Some detection programs will scan a suspect disk looking for known viruses. Many of these are themselves available in shareware or public domain form. As a safeguard, you can scan all floppy disks before loading the programs to the hard disk or running them the first time. Detection programs, however, may not protect a system that's already been infected. It may also be defenseless against new viruses developed after the scan program itself.

An alternative is a TSR monitor that watches requests to DOS interrupts. These look for known viruses as well as out-of-the-ordinary I/O requests. They usually watch for requests that write code to the application itself, which might indicate a virus that is trying to attach itself to the application.

Unfortunately, there are applications that routinely write to themselves, changing configuration or installation parameters, for example. When the write occurs, the TSR is triggered and warns of a possible infection.

There are also programs that check



the integrity of individual files using the checksum or similar method. Checksum was originally developed for confirming the accurate transmission of data over communication lines. In checksum, an algorithm is applied that computes a unique number based on the bits that make up the file. The detection program calculates the checksum of your programs and saves them in a disk file.

You then run a TSR that computes the checksum of a program that you execute and compares it with the stored checksum. If the numbers do not match then the TSR warns that the program might have been infected. These types of programs, however, are susceptible to the same problem as TSR monitors. That is, a program might write new code to itself which would result in a different checksum the next time the program was executed.

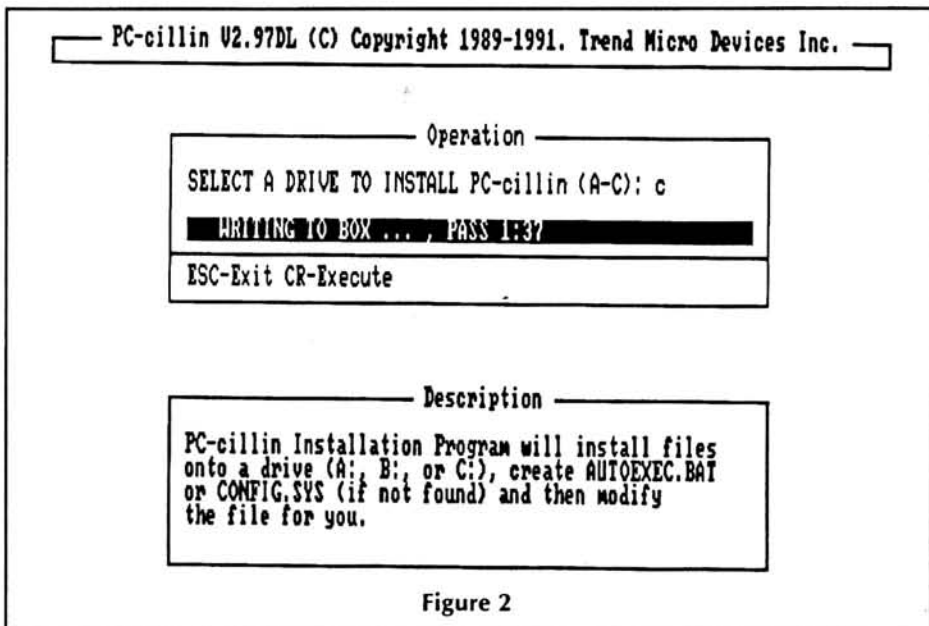
Each type of detection program serves

a useful purpose, although none are fool-proof. A sneaky boot sector virus, for example, can install itself into memory before any TSR detection program is loaded. As such, it can trick DOS into ignoring the warnings triggered by the detection program. Anyway, once the boot sector, FAT, and partition tables are effected, viral detection programs won't be of much use because the damage has already been done.

The TSR detection programs take up useful memory, may slow down system performance, and cause panic if triggered by false alarms.

PC-cillin

Trend Micro Devices has tackled virus detection with a combination hardware/software approach that combines the detection methods. In addition, it saves a copy of your boot sector, FAT, and parti-



tion table to help recover from a virus that has attacked the system files, or to reconstruct the system after a general hard disk failure.

The system, called PC-cillin, includes a small hardware device, called the Immunizer Box, that connects in-line to the parallel port. The box contains non-volatile EEPROM that stores a copy of your disk's boot sector and partition table. The Immunizer also includes code that checks the PC-cillin system files each time the computer is started.

When you start your computer, the PC-cillin program compares itself with a version in the Immunizer box. If it detects that its disk file has been infected, it downloads a clean copy from the EEPROM. It then checks the disk's boot sector and partition tables with the version stored in the Immunizer. If either have been altered, they are immediately downloaded.

The key to using the Immunizer is the PC-cillin installation program. When you first load the system, it checks system memory for any RAM viruses using what it calls "intelligent viral traps" (Figure 1). Instead of merely checking for currently known viruses, it examines the disk using IA techniques which detect general viral behavior. Trend claims that this insures detection of any new virus which may have been developed after PC-cillin.

If memory is clear, it checks the boot sector and partition table for any viruses. If these are clear, the vital information is written to the Immunizer box (Figure 2). PC-cillin then scans all of the programs on your hard disk for infection at a rate of 7 MBytes per minute. If it finds an infected program, it will report the file and give instructions. PC-cillin doesn't attempt to remove any virus but suggests deleting the file and replacing it with a clean copy.

Once installed, the PC-cillin TSR is loaded and checked every time you start your computer. After confirming that your boot sector and partition table are clean, it displays the message

```
PC-cillin HAS EXAMINED YOUR SYSTEM.  
NO VIRUS HAS BEEN SENSED AT THIS TIME.  
PLEASE PROCEED WITH NORMAL OPERATION
```

The program then works in the background, checking every program you execute for signs of infection. If any are detected, it stops the program execution and displays a warning message and instructions on the screen.

For example, after installing PC-cillin I intentionally ran a program with a known virus. As soon as the program was executed, PC-cillin took over, halted execution, and reported that a virus has attempted to take up residence in my RAM. I was told to turn off my computer and reboot, then to delete the offending program from the disk.

PC-cillin also supports two additional functions, Quarantine and a Rescue Disk. You use the Quarantine option to check

new programs or disks before running or transferring to the hard disk. The program lets you specify the disk, subdirectories, and files to be checked (Figure 3). If it detects a virus, it tells you the type and gives you the option of deleting the infected program (Figure 4).

The Rescue Diskette is an added bonus that provides support if a virus or other problem causes a hard disk crash that leaves the system unable to boot. It stores its own code and a copy of vital system information on a bootable floppy disk. If for some reason you are unable to boot from the hard disk, you use the rescue disk to boot and automatically restore the boot sector and partition table from files on the disk and in the Immunizer.

This makes PC-cillin do double-duty as a virus detection system and as a boot sector/partition table recovery program.

Hard Disk Backup

If a program has already been infected when you run a viral scanning program, then you'll have to restore it from a clean copy. You'll also need copies of your files if a hard disk crash damages the disk in a way that recovery methods cannot help.

Backing up the hard disk to floppy disk, external hard disk, or tape is effective in most cases. However, it is the most time consuming recovery method and requires periodic, if not daily, implementation. It will also not protect you from computer viruses. If you unknowingly back up an infected file, you'll re infect your entire system when you restore it to the hard disk. That is why it is important to secure your original distribution disks.

Numerous articles have been written about using DOS's BACKUP and RESTORE programs for safeguarding your files. There are also a variety of third-party programs designed just for backing up your disk. But no matter what program you use, it is your technique that is critical to good disaster

recovery.

For example, suppose you completely backup your system to floppy disks in January. You tuck them away in a secure location with a safe feeling that you're protected. Three months later your hard disk crashes and you restore the backed up files to the hard disk.

Do you have an effective recovery? Not really since you've lost three months of work. The only way to utilize disk backup for hard disk recovery is by regularly making incremental backups. An incremental backup records your most recent changes — new and updated files that were not saved in the last complete backup. If you make an interim backup weekly, the most you'll lose is one week of work.

Many users improve their chances of recovery by adding the commands for making an incremental backup to their AUTOEXEC.BAT file. This way, an incremental backup is made every time they start their computer. I know of others who add the commands to batch files that they use before turning off their system.

Daily (or even more frequent) backups such as this, of course, can take up quite a bit of time and a large number of disks. If you only update a few files each day, however, the time may not be that noticeable. If you backup to high-speed tape or some sort of external hard disk, it will be even less painless.

Many general-purpose DOS shells can be used to perform full or incremental backups using their COPY command. QDOS II, for example, can make an incremental backup of files. DOS Partner's copy command will even format the destination disk for you and prompt you to insert a disk when one becomes full.

Independent backup programs are designed to make backups faster and more convenient. Some backup programs, such as Perfect Backup, save files in their own special format that requires their own res-

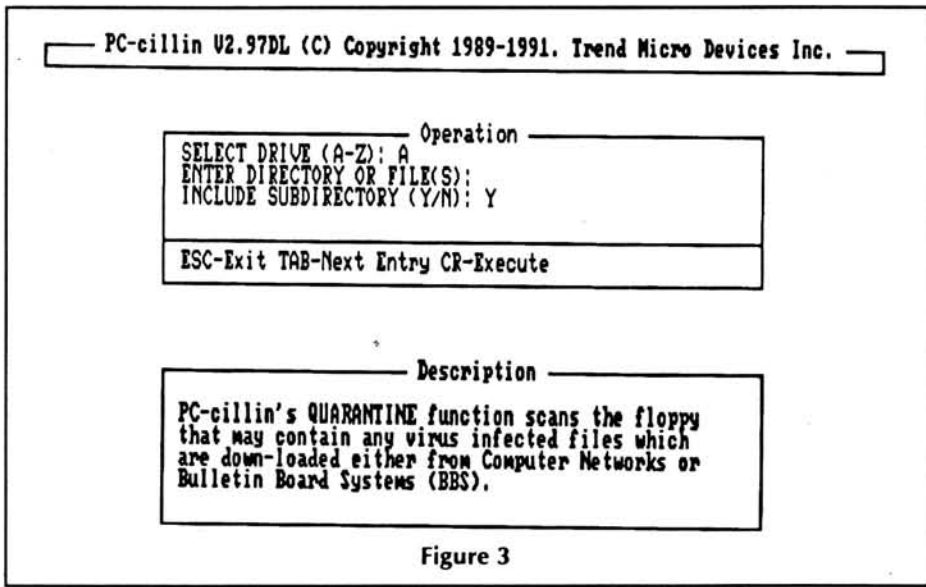


Figure 3

```
Operation
SELECT DRIVE (A-Z): B
ENTER DIRECTORY OR FILE(S):
INCLUDE SUBDIRECTORY (Y/N): Y
DELETING FOR VIRUS INFECTED FILES.....
ESC-Exit TAB-Next Entry CR-Execute
```

```
Verifying B:\README.1ST
Verifying B:\1701.COM

Found Virus 'Cascade(01)' in B:\1701.COM,
Delete this file(y/n)?
```

```
THEY MAY CONTAIN EQU VALUE INFECTED FILES WHICH
are down-loaded either from Computer Networks or
Bulletin Board Systems (BBS).
```

Figure 4

toration programs. Other programs, such as BackEZ and Backup, save files in DOS format so they can be restored using COPY or XCOPY.

Both methods have their advantages. Most propriety backup formats save files compressed, meaning that you can store files on fewer floppy disks. However, you must have the program available to decompress and restore the files. Backup programs that save files in DOS format give you the option of working with the backup files using standard DOS commands.

All backup programs, however, are only as good as your technique. You have to remember to run the program for incremental backups and you have to develop good floppy disk storage techniques. Having a pile of unlabelled backup disks scattered around the house may be no protection at all.

As an example of backup programs, let's take a closer look at two that use different approaches.

Intelligent Backup

This is a menu-driven program from Sterling Software. When first run, it will prompt you to make a complete backup. Later, however, you have the option to make an incremental backup or to consolidate your backup disks.

Intelligent Backup reports its progress during the actual backup, giving an estimate of how long the backup will take and the number of directories, files, and characters have been backed up and how many remain. You can cancel the backup at any time.

Restoring files is just as automatic. You designate directories or files to be restored using the ? and * wildcards, or tag specific

files in a restore directory. Several advanced options let you customize your restore even further.

If you want to reduce the number of floppy disks required for backup, you can select between two methods of data compression. Using standard compression you reduce file size up to 40%. You can also select super compression that reduces size up to 80% but increases backup time by about one half.

Sit Back

If you want the security of backups but don't want to spend the time making them, consider Sit Back from SitBack Technologies.

Sitback is a TSR program that automatically makes incremental backups during periods of keyboard inactivity. When Sit Back senses that you haven't pressed a key for about two minutes, it scans your disk to backup new or updated files. It will even continue the backup in the background as you work with your application, although you may notice some slowdown in system response.

When you install the program, you designate the disk drive to use for the backup. As you work, you have to keep a backup disk loaded and ready. Sitback will only backup files to a disk with a volume label beginning SB, so it will not destroy an application or data disk that you have inserted in the target drive.

As with most backup programs, you can tell Sitback to exclude certain files and you can customize it in a number of ways. For example, you can inactivate the keyboard delay option or set it to an interval between 20 and 60,000 seconds. You can also designate specific times of the day to

backup files, to activate Sitback during lunch or dinner breaks, for example.

Preventing Disaster

You can greatly reduce the chances of needing a recovery system, but not totally guarantee it, by using careful technique and common sense.

First, take care of your hard disk:

1. Before turning off your computer, use a ship or park program that moves the heads to a safe location. This greatly reduces strain on the disk surface when you turn on your computer.
2. Vacuum the area around your computer, and in and around the disk drive bays, regularly.
3. Don't smoke around your computer.
4. Run programs such as CHKDSK periodically.
5. Consider MACE Utilities, PC Tools, or another system that safeguards the system files.

Second, try to avoid viruses as much as possible. If you only use legal copies of commercial programs — not public domain, shareware, or illegal copies of programs — then you greatly reduce your chances of infection. If you use other types of software, then use these precautions:

1. Get public domain/shareware programs from reputable sources, not second-hand.
2. When using a new disk, boot from a floppy disk not the hard disk. If an infection occurs, it may only infect the floppy disk not your hard disk.
3. Make critical files read only using the DOS Attrib command.
4. Beyond that, consider a virus protection or detection program of some sort. When you download a file from a BBS, download to a floppy disk first. Log off, then use a detection program, such as Quarantine, to check the file for any infection. In fact, check all software — commercial, shareware, and public domain programs the same way.

(Only once did I find an infected file — in an obscure public domain program I received at a computer users group meeting. The virus, I later discovered, was a rather harmless one that just displayed a greeting at bootup. But since the virus could have been more serious I was certainly happy that Quarantine was on my side.)

Third, use proper storage and backup techniques:

1. Carefully store your original distribution disks. Write protect them, never use them on your system, and never lend them to another user.
2. Backup the files on your hard disk. Make incremental backups as needed.

✱

Getting Started With . . .

Laser Printer Font Tools

Alan Neibauer
11138 Hendrix Street
Philadelphia, PA 19116

If you have a laser or ink jet printer that can accept downloaded fonts, then you know how powerful a printer can be. When combined with the right application programs and printer drivers, the right printer bridges the gap between word processing and desktop publishing.

In several previous articles, I discussed using fonts with both PCL printers, such as the Hewlett-Packard LaserJet family, and Postscript printers. I also discussed generating fonts-on-the-fly, or scaling fonts to almost any size as characters need to be printed.

In this article, I'll look at specialized tools for creating unusual effects with softfonts and manipulating graphic images for output by a laser printer.

In light of new font scaling (font-on-the-fly) technology, programs that deal with bit-map fonts may seem out-dated. But while fonts-on-the-fly programs are available for Windows, WordPerfect, Microsoft Word and other major applications, there are many programs that still rely on bit-maps. To get the most out of your laser printer with these applications, you must still generate font bit-maps in the sizes and effects that you want.

In addition, there are some effects that scalable fonts just do not offer. While Glyphix can generate special effects using patterns, for example, most font-on-the-fly software can generate only normal, bold, and italic fonts. If you want to custom design a font, or combine effects, then you'll have to rely on bit-maps and programs that manipulate them.

Each of the programs discussed here offer special features that make them valuable additions to a laser printer's arsenal.

Font Solution Pack

This product from SoftCraft, Inc. is a collection of useful programs integrated

through a main menu and a central setup environment. The programs use either bit-mapped fonts or Bitstream outline fonts designed for the Fontware generation program, not fonts-on-the-fly. Graphic images include those in the PCX or TIF formats.

Let's take a look at each of the program modules.

Fontware

Like the Bitstream Fontware program, this module generates bit-mapped fonts in sizes from 3 to 240 points from Bitstream outlines. You can slant fonts at any angle and kern character combinations for a real typeset appearance.

The program also allows you to create

custom character sets. For example, normally, if you intend to print line graphics, you'd need a font with the PC symbol set. This includes graphic characters in the ASCII range 176 to 223. But if you want to print certain accented foreign-language characters, you'd need the Roman-8 symbol set. In this set, the ASCII range 161 to 242 is mapped to foreign-language characters and symbols.

To print line graphics and foreign characters in the upper ASCII range in the same document, you'd need available two separate bit-mapped fonts, then switch between them manually. Using Fontware, however, you can combine the line drawing and foreign-language characters that you need

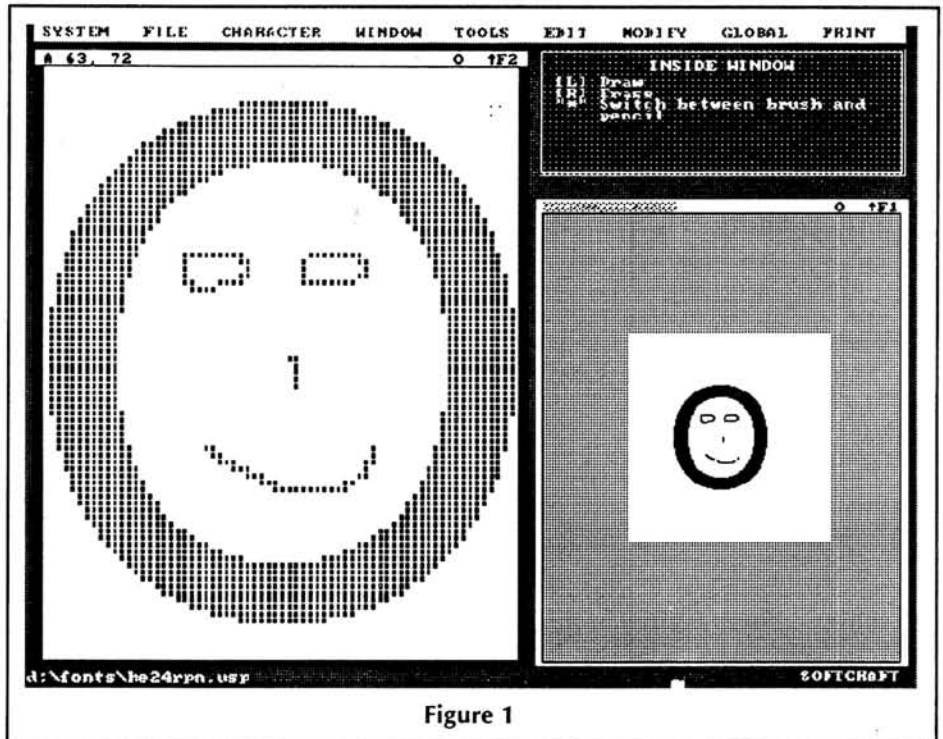


Figure 1

in one font.

Fontware also can create screen fonts for use with Windows applications and for Ventura Publisher. It does so through linkage with a program called WYSIfonts, also supplied with the Font Solution Pack package. (You need version 2.2 of Font Solution Pack if you have Windows 3.0, by the way.)

Edit

Even greater font control is provided by the Edit module. This font editor lets you modify individual characters, create your own symbols or logos, or even create an entirely new custom font.

You edit or create characters by modifying the actual bit-map pattern, on a dot-by-dot basis using pixels on the screen. You can also slant, invert, rotate, and combine characters for unusual effects, and import graphic images created with drawing programs. In addition, you can use the editor to manually smooth jagged lines and arcs in large fonts.

By editing characters and replacing seldom used ones with those of your own design, you create logos and special symbols that are readily available. For example, Figure 1 shows the Edit screen while a smiling face is being added to the uppercase O character. To create the face, I used the mouse as a drawing tool, clicking the left button where I want to insert a pixel, the right button to erase one.

Edit uses windows so you can work with an enlarged version of a character while seeing the final results at the same time. The left window shows the actual dots that make up the character so you can edit it on a dot-by-dot basis. The window on the right displays a higher resolution image to indicate how the printed character will appear.

Figure 2 shows a custom logo being created. (Blame the poor resolution in the right window on my screen dump program, not Edit.) The resulting logo will fit into the standard character width of the font being edited. So, in this case, the logo can be used within a text line for a special effect.

In addition to pixel-by-pixel manipulation with the mouse or keyboard, Edit provides a series of drawing tools. You can draw lines, circles, ellipses, and boxes of various sizes and line thickness, splines to create smoothed curves, and fill bounded areas with solid black or patterns.

Font Effects

While edit allows you to modify individual characters, Font Effects can transform an entire font. You can use an existing font bitmap to create fonts of other sizes, shapes, patterns and effects.

For example, suppose you downloaded an 18-point font from a bulletin board. Using Font Effects, you can create

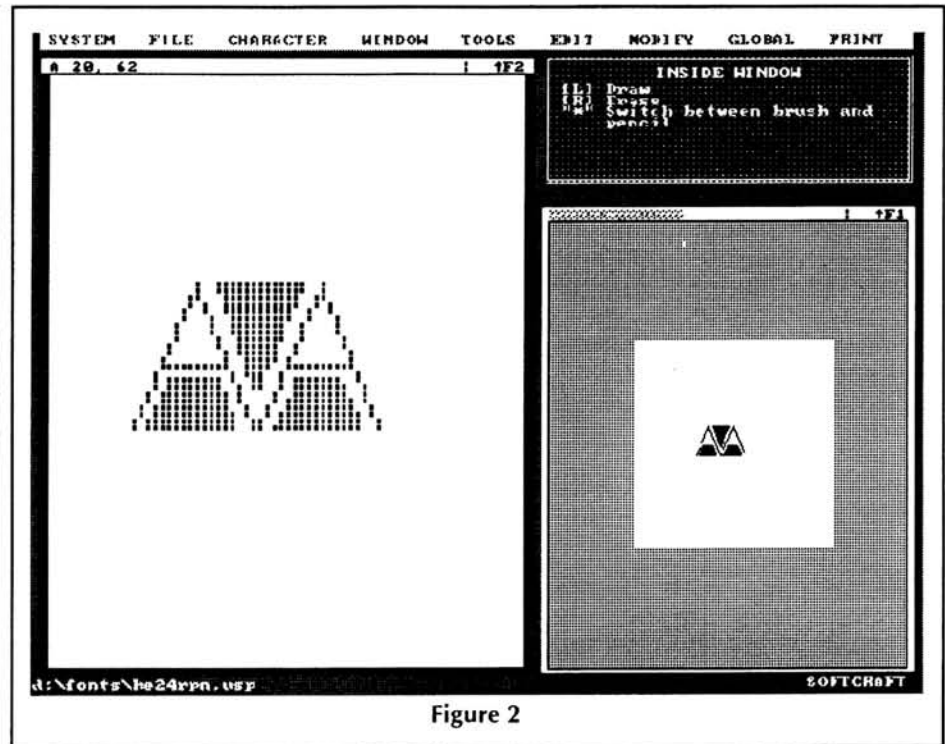


Figure 2

similar fonts in other sizes and effects, such as outline and shadow fonts of the same typeface. Font Effects works with bitmaps, not scalable outlines. So as long as you have one size of a typeface, you can use Font Effects to create an entire type gallery.

The program offers a set of 14 standard effects, such as various fill patterns of outline and shadow fonts, that you can select with a few keystrokes. However, an unlimited range of effects is available using several options on the Font Effects screen. You can, for example, modify the line thickness, outline size, and pattern used to generate the new font, as well as define its background. The background settings can be used to create reverse characters over solid black and various patterns, as well as the types of shadows used for shadow fonts.

The modify option allows you to scale, bold, slant, and fillet the font. Filleting fills the gaps on the edges to create a smoother, and slightly bolder, effect.

Figure 3 shows the transition of a plain Helvetica character to a custom font. (Again, the poor resolution is due to my particular screen capture method.) The uppercase A was converted into a shadow outline, enlarged to twice the point size, then filled with a pattern. A font set could be generated at any stage of the transition for a complete collection of custom fonts.

Spinfont

This is my personal favorite of the Font Solution Pack. Spinfont uses Bitstream outline fonts to create PCX or TIF graphic files that can be merged into your word processor or desktop publishing document.

By creating a graphic file, you can print

headlines larger than the font size your printer can normally handle. You can also create a wide range of visual effects, as shown in Figure 4. You can create graphics of rings, spokes, curved, and rotated text, even combining portrait and landscape text into one file.

After you design the graphic, Spinfont links to Fontware and Font Effects to generate the characters needed in the desired style. Spinfont then creates a PCX or TIF file storing the completed design. You can then load the graphic into your application, such as into a figure box with WordPerfect or as a graphic in PageMaker.

For example, I wanted to send a sample of a newsletter I desktop publish to prospective clients. I decided I wanted the word SAMPLE in 72 point type printed diagonally across the first page. But my LaserJet Plus can not accept a 72-point softfont nor rotate a line of text 45 degrees.

I told Spinfont to create the word SAMPLE in 72-points from a Bitstream Char-

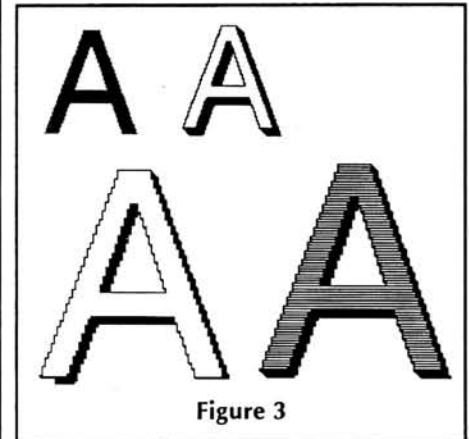


Figure 3

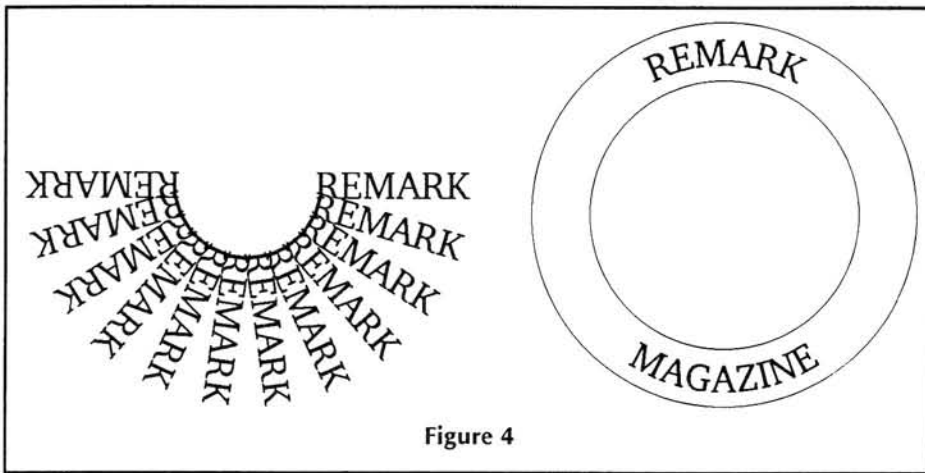


Figure 4

ter outline font that came with the Font Solution Pack. I also told it to rotate the text 45 degrees and create a .TIF graphic file. Then after designing the first page of my newsletter with WordPerfect, I created a graphic box set so text would not wrap around the graphic. When the page printed, the .TIF image printed directly on top of the text just as I had designed.

You can use Spinfont to create attractive logos and eye-catching graphics. For instance, for proposal cover pages I create and use a ring graphic containing the client's name. Most clients are impressed believing that I had an artist manually create the image just for the proposal.

Install Fonts

This module installs bitmap fonts into Windows or application printer drivers. It works with any bitmap font, from any source. If you're using WordPerfect, for example, it adds your fonts to the appropriate .ALL file so you can update the .PRS file from within WordPerfect.

Remember, all parts of the Font Solution Pack are integrated. So if you want, the program will keep track of fonts that you create with Fontware or Font Effects, then automatically install them in your applications.

The module for WordPerfect, by the way, also includes its own options for shadow and outline fonts. When you install a font, you can tell the program to have Font Effects create the shadow and outline fonts at the same time.

Install works with Windows, WordPerfect, and Microsoft Word.

Utilities

Three utility modules are provided on the main menu.

Preview will display a high resolution image of any font or graphic image. Use it to check the appearance of a font before installing it in an application, or of a graphic image before merging it.

Transform converts graphic files between .PCX and .TIF formats.

Print will print a font or graphic image

on any number of compatible printers.

Now let's take a look at another font tool.

Publisher's Type Foundry

This Windows application from Z-Soft Corporation also allows you to create an unlimited number of fonts. While the Font Solution Pack is more graphically-oriented, the Type Foundry takes a more typographical approach. If you want, you can deal directly with character spacing, baselines, underline and other offsets that make up the technical aspects of fonts.

However, the package's most powerful components are a bitmap editor and an outline editor, which make it unique among other font management tools.

The Outline Editor

This editor lets you work with fonts in outline form, similar to the font outlines used for Postscript and font-on-the-fly pack-

ages. You make major changes to a font with this editor, then create individually sized bitmaps that you optimize in the bitmap editor. (According to Z-SOFT you only need to modify two or three dots per character to optimize the bitmap.)

Where do you get the font outlines from in the first place? The outline editor lets you import some formats of Postscript fonts. You can also use the bitmap editor to convert a bitmap font to an outline for use with the outline editor. You convert the bitmap font to an outline, manipulate it with the outline editor, then save it as either a bitmap font or a Postscript font. If you save it as a bitmap, you can then use the bitmap editor to work on a dot-by-dot basis to create the best possible printed appearance.

So, if you have one size and style of a font in bitmap form, you can use the Type Foundry to create an entire family of fonts — in various sizes and styles.

For example, Figure 5 shows the outline editor screen. The outline of a character is displayed in the left window and an illustration of the resulting bitmap or printed image on the right. Using editing tools, you can slant, rotate, scale, and otherwise manipulate individual characters or the entire set. You can also merge characters or create new ones.

But the most interesting aspect of the program is the ability to change the actual outline segments using a series of tools. You can add lines, curves, and points, and move the points that connect segments. Figure 6, for instance, shows the same character after several points have been added and moved. By working with the outline segments, you can make major

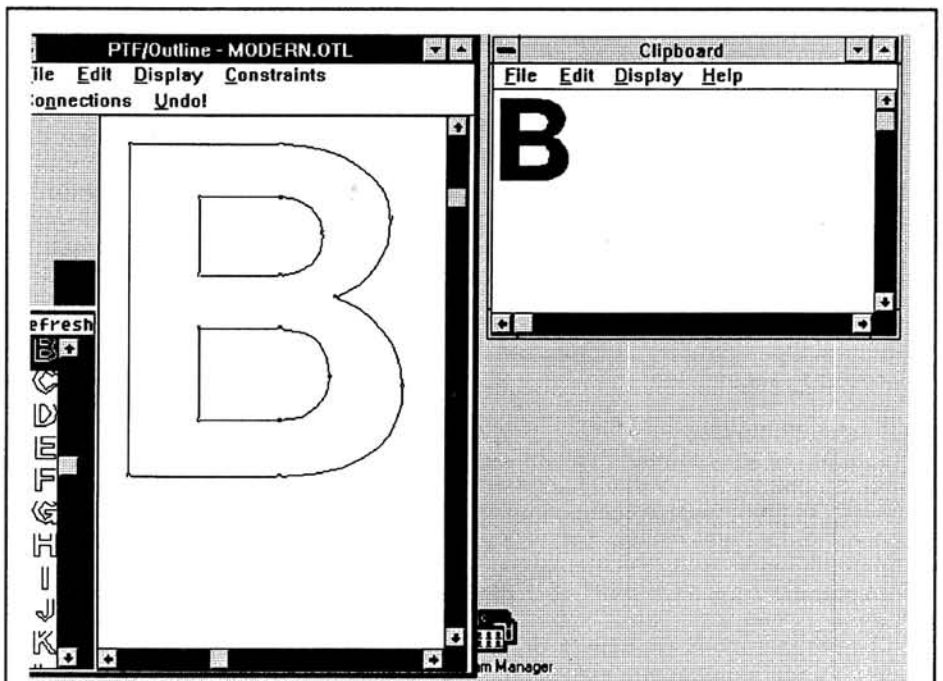


Figure 5

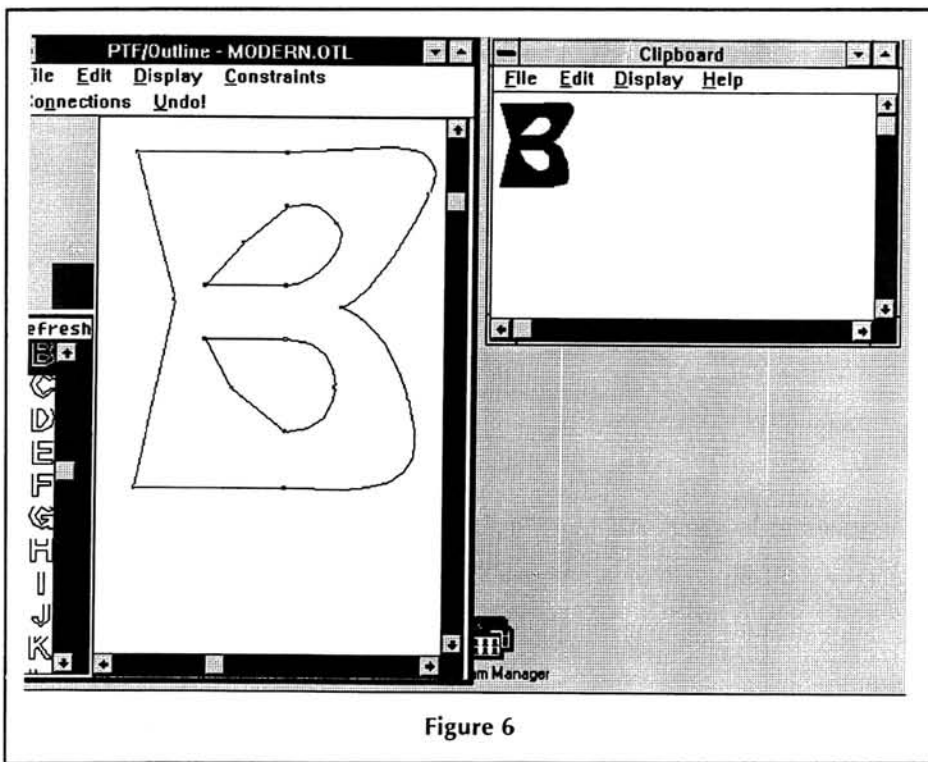


Figure 6

changes to a character without having to draw or erase a long series of individual pixels.

BitMap Editor

The bitmap editor combines some of the features of the Font Solution Pack's Edit and Font Effect modules. You use the editor to modify individual characters on a dot-by-dot basis, or change every character in the font globally. For instance, you can slant characters to create an italic font, add various fill patterns to outline fonts, or use drawing tools to help in modifying characters or creating new ones.

The editor can only use fonts in a special format called .XFR. However, translators are provided for converting fonts back and forth from .XFR to Hewlett-Packard softfont format, or to the outline format suitable for manipulation with the Outline Editor.

Using both editors, you can convert fonts between bitmap and Postscript formats. This is useful if you have both types of printers available or you are changing from one printer to another.

SLED Image Processor

SLED, from VS Software, can also be used as a bitmap editor to alter characters and fonts, using typographical controls similar to those in the Type Foundry. But SLED's real strength lies in its abilities to manipulate graphic images in .PCX, .IMG, and .TIF formats.

The program offers a number of drawing tools, as well as fill patterns and graphic editing capabilities. In fact, SLED offers some commands not found on most draw-

ing programs, such as Hollow, which "whites out" the interior of a black graphic leaving just the outline in black.

Two particularly useful SLED functions are the Fill Black with Pattern and Black to Pattern commands. I've used both to create a "screened" image that I can overprint with text. Fill Black with Pattern replaces the solid black background of an image with a selected pattern. Black to Pattern does the same, but leaves the outline in place by first hollowing the image, then inserting the pattern. I fill the area with a light pattern, load the image into a WordPerfect figure box and set the option to not wrap text around the image.

In addition, SLED makes excellent use of softfonts, allowing you to add text to a graphic using any standard softfont with the .SFP extension. This is especially useful if you're working in DOS, not Windows. While Windows provides its own support for softfont in painting and drawing applications, DOS does not. Most DOS-based graphic programs rely on their own font files, which are often of lower resolution than the laser printer's 300 dpi. SLED, on the other hand, allows you to incorporate characters from standard softfonts in your images.

SLED also offers a unique ability to convert line images to scalable outlines, rather than maintain the bitmap patterns. If you've ever tried to resize a drawing, you probably noticed how the original resolution and clarity it often lost with lines getting a jagged appearance. By converting the image to a scalable outline, SLED can scale an image with more precision.

Also making SLED unique is its ability

to save images as a font file (with the .SFP extension) rather than a bitmapped graphic image. This provides a way to print images much faster than the printer can normally produce a .PCX or .TIF graphic. It also allows you to print graphics with word processors, text editors, and other software programs that do not have their own graphic commands.

This is accomplished through use of a .TSR monitor called SLEDMNTR. The monitor watches output to the printer for special imbedded commands surrounded by the caret (^) symbol. The commands tell SLEDMNTR to print a previously downloaded graphic font file, or download the one named and print it at the position of the cursor in the document.

For example, once you load SLEDMNTR, you can print a graphic from any text editor or using the DOS print command, by entering a line such as :

```
^C:GRAPHIC.SFP^
```

When SLEDMNTR encounters the command in the flow of text to the printer, it checks to see if it already downloaded the image to the printer. If the file is not found downloaded, SLEDMNTR locates the file on the disk and downloads it for printing.

If you have a program that handles graphics, it might still pay to use the monitor, especially if you plan to use the same graphic more than once in a document or during a printer session. For example, suppose you print a scanned signature or company logo with every letter. Save the images as graphic font files and include commands to download and print the file in a document. When SLEDMNTR download files, it assigns them ID numbers in the range from 20000 to 20015.

Now each time that you have to print the graphic in subsequent documents, SLEDMNTR finds that the images have already been downloaded and prints them immediately.

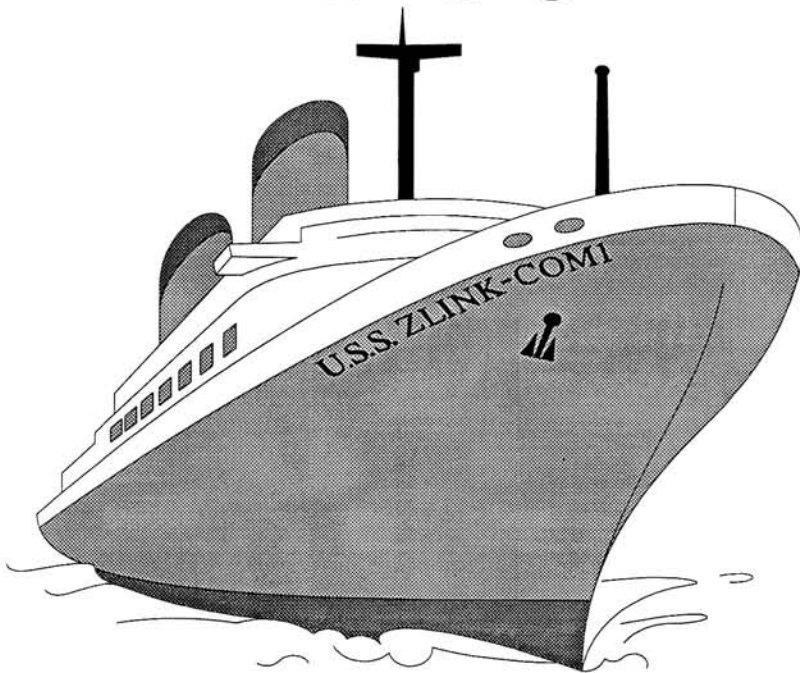
You can include options in the imbedded commands to print a font with a specific ID, assign new ID numbers, or position the graphic relative to the baseline of the cursor in the document.

Just remember that softfonts takes up your printer's memory, and large graphic fonts can be quite an overhead.

While I didn't explain SLED's font editing ability in detail here, it is quite powerful, although it lacks some of the more spectacular features of Font Effects. For example, SLED allows you to quickly create a custom text or symbol font by incorporating up to 94 characters in a single image. You carefully position each symbol in its own square area on the screen. To create a 24-point font of special symbols, for instance, create (or scan) each symbol in a 100 by 100 dot area, then position each carefully on the screen. Then save the

Continued on Page 48

Port-O-Call



COM1

Laura White
759 Polfus
Benton Harbor, MI 49022

Port-O-Call: COM1 is a monthly article designed to give REMark readers a hard copy of useful information that is passed back and forth on the bulletin board. This also gives ZUGgies a chance to benefit from the bulletin board, even if they don't have the luxury of a modem.

This column is not limited to Zenith Data Systems/Heath products even though REMark is published by Zenith Data Systems/Groupe Bull. Its only limit is COM1 and the subjects discussed among its users. Because the nature of this column is to simply act as another form of information transferral that takes place among COM1 users, the information supplied here has the same "guarantee" as that of the bulletin board — a casual exchange of information.

DESKTOP PUBLISHING

Step One: Evaluation of Needs

If you are like fellow ZUG member George Bernegger, you might be contemplating getting involved with desktop publishing, but do not know where to begin. Like most, George was looking for a somewhat user-friendly package that does not take forever to master. He was going to do newsletters, catalogs, news releases and a possible media kit.

Evaluating your needs is a good place to start. Before going out and buying the package that you hear everyone is using, take the time to evaluate your needs. What are you going to do with your system? Are you going to need several different types of fonts? Will you be needing graphics capabilities? Do you plan to run all your copies

with your printer, or are you going to supply just the original and take it to a printer? Will you be using your system often, therefore needing a package that may be more difficult to learn, but more powerful to use once it is mastered?

Or will you be using it to write a semi-annual newsletter and cannot afford to re-learn the product every six months?

Once you have evaluated your needs, the next step would be to evaluate the equipment you presently have and how much you are willing and able to spend. Our soon-to-be desktop publisher was only familiar with the Epson EPL-6000 personal laser printer and did not know about fonts. This is where the second step is essential when buying a desktop publishing system.

Step Two: Education

Once you know what you need the system to do and you know what you are going to produce, you need to become familiar with what is available. You also need to know what the capabilities are of any hardware/software, and what you will need to bridge what you have with what your expectations are.

If, in the past, you have chosen various typefaces (fonts) for your published pieces, chances are you will need either a PostScript laser printer (or a PostScript upgrade for your present printer) or a software package to give your present printer a larger variety of fonts. Either option will give your printer more fonts, which will provide more versatility for your finished documents.

Note: Keep in mind your typesetting

needs when it comes to a software package as well; not all packages can meet the requirements of many fonts. We will be discussing software in detail later on.

Suppose you have a PCL printer (Hewlett-Packard LaserJet family or 100 percent compatible printers.) One way you can get more versatility from your printer is through a font package. A font package allows your printer to produce type other than what it is programmed to do, giving you expensive capabilities without the cost.

What is a soft font? To quote text from Aldus PageMaker's fontware package, "A fontware font is a set of characters for a specific model of printer or screen device." You can create that set of characters using the Fontware Installation Kit and a Fontware typeface. Once you create them, fonts exist as files on your system.

To decide which fonts to create, consider which typefaces you use often; which printer and display device you will be using; which set of characters you need; and which type sizes you will need. You might like to know that the more well-known desktop publishing software, such as Ventura and PageMaker, include fontware right in their packages.

Step Three: Software Capabilities

Basically, there are three types of software you may use in desktop publishing: complete "super" packages, such as Xerox Ventura Publisher and Aldus PageMaker; word processors, such as

WordPerfect and Microsoft Word; and graphics/art packages, such as PC-Paintbrush or Harvard Graphics.

The super packages have everything you could possibly need: PostScript support, on-screen layout capabilities, art and graphics and, of course, word processing capabilities. They also have the capability to import (bring in from another package) spreadsheets if necessary for graphs, etc.

A good alternative to a DTP package is a word processing package. Today's packages are becoming more powerful than the original text editors. The newest ones give you a variety of capabilities such as graphic import, limited drawing features, and a large selection of fonts. Often, you will find this will do everything you need at a fraction of the cost of a "super" package.

Graphic and art packages simply add an additional dimension to your publication possibilities. They allow you the versatility of adding one color art work to your pieces. It does, however, help if you have some artistic capability to really get the full effect from them. Otherwise, they do offer basic graphics which can either add or take away from your publication, depending on your eye for art-filled pieces.

Step Four: Summing It All Up:

Printers, software, fontware, what does it all mean? Where do we go from here? Basically, after you have compared all the pros and cons, weighed it all with your pocketbook, the choice obviously is a difficult one.

Remember George Bernegger? He received some very practical advice from some fellow ZUGgies. There was, however, one piece of advice that, in my opinion, was probably the most useful. George was told to sit down and read publications, talk to salespeople, and finally, set up a comparison table and match features against the job(s) you want to do. He was also told to narrow his choice down to two or three packages/alternatives, put an ASCII text file on disk to a dealer, and finally, see what you can do.

Regardless of the actual specifics and how you do your comparison, the important thing to remember is to do your research. Desktop publishing is a big step and one which should take some time. You might also want to remember that by doing your research before purchasing software and hardware, you will be that much further ahead when it comes time to sit down and learn the specifics.

OVERVIEW OF COM1

LED's lighting up; IDE controllers, QEMM 386 with an ESDI drive; adding VGA to a TurbosPort and, upgrading the 386/16 CPU board to 20 MHz.

In this section, we will be discussing a wide variety of topics ranging from LEDs,

IDE controllers, QEMM and its actions with an ESDI drive, to adding VGA to the TurbosPort and speeding up the 386/16 Motherboard.

Diagnostic LEDs Lighting Up

Frank Zinghini called in and wanted to know what to do about his diagnostic LEDs that were lighting up and staying lit. The computer, a 386/25, would never go through the first test and he had to hit <CTRL-BREAK> to stop it before the test went into the AUTOEXEC file.

There were a number of recommendations made. One thing to check for was the power supply voltages, connections, etc. to make certain that there are good clean connections throughout the computer.

Next, it was recommended that he check the six green lights on the motherboard. If the +5 volt (LED closest to the rear of machine) comes up very slowly or flickers, he was told to suspect the power supply. If this was the situation, try and change the jumper on the I/O card (#326) to pins two and three — this will bypass the battery check and allow a boot. It was suggested that all of the LEDs being lit up might suggest that the POST (Power On System Test) cycle was hanging up on the power supply, RAM or CPU.

Finally, it was recommended to check the voltages with a calibrated "O" scope and begin deciphering the problem by focusing in on the power/connection problem.

IDE Controller

Originally, a question was raised whether or not Zenith Data Systems offered an IDE controller with the holes necessary to solder a DB37 plug on the back side for use with an Irwin 445A tape drive. The intention was to upgrade the system to IDE, but not need to purchase an adapter board for using the tape backup system.

Since most ZDS computers have the IDE interface built on the motherboard, the caller had to clarify himself and said that what he was really looking for was an IDE interface board that has a floppy controller and the holes necessary for adding a DB37 plug for his Irwin. No direct answer was given to his question, but an approach to the solution (without the DB37 connector) was to purchase the adapter board (Irwin #4251).

To install it, remove the 34-pin floppy drive cable and connect it to the Irwin board. Then connect the 34-pin ribbon jumper from the Irwin board to where the floppy interface cable is connected. **Note:** Normally, the 445A has a separate DB9 power connector adapter. If the power adapter was not purchased, one can be constructed to supply the +5 and +12 volts from one of the drive power cables.

QEMM With an ESDI Drive

A ZUGgie called in and informed other users that he was not able to get QEMM to run successfully. Whenever it was activated, the ESDI drive died. (Data transfer rates dropped to below XT speeds. Checked using *Coretest* and *PClabs Benchmarks*.) He had been all through the options and found that the problem was not an address conflict problem. Even with QEMM options turned off (all memory excluded and no EMS frame) it still didn't function at its optimum speed. The only way he found his machine was able to regain its speed, was to manually switch off QEMM. The minute he turned QEMM on, the hard drive began running at a crawling speed. He had the following equipment: 386/20, 2MB 16Kcache, DTC ESDI 72 MB, ROM# 444-684-6STB, (TSENG 4000) and VGA.

He was given the following adjustments to make in his software: `DEVICE=LOADHI.SYS SMARTDRV.SYS 512-256`. This apparently speeds up disk access. He was told that he can also include paths which are tailored to fit the situation.

Our ZUGgie with the problem called back and informed us that he had called Quarterdeck and they said that with the hard drive setup he had, they estimated he had approximately a five percent procrastination time and that it was just enough to throw the machine out of sync. They also suggested that the QEMM overhead may be sufficient to foul up the hard drive controller so it was occasionally missing a 1:1 interleave.

The problem was then clarified even more. Apparently, the real problem with the deterioration was in the sequential file and disk transfers. He gave the following example: When loading a program — with QEMM it took 16 seconds; without QEMM, it only took four. He got the same results whether or not he used the cache card. He also pointed out that he was using the latest version of QEMM 386 which is 5.12.

It was questioned whether or not the version was the problem and we were informed that he tried version 5.11 and had the same results. It was pointing to the problem Quarterdeck had at first thought: The system may not be able to handle a 1:1 interleave with a five percent overhead.

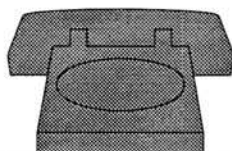
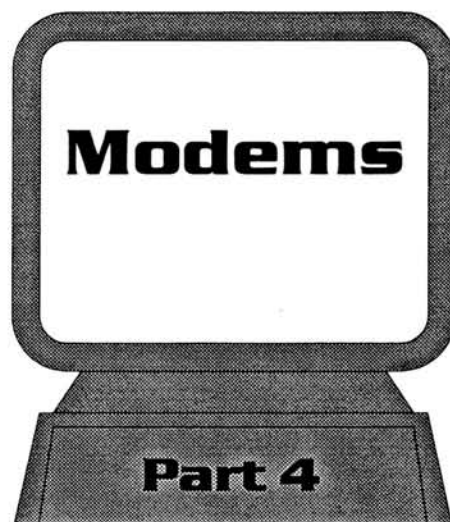
Now, the problem was whether or not there was a way to LLformat (PREP) at 2:1 through the ROM. Finally, our friend called back and informed us that he reformatted 2:1 (by using PREP /1:2) and viola! The hard drive rates had increased considerably.

TurbosPort and VGA

To add VGA to a CGA Turbosport, all that is needed is an expansion box. Model number ZA-3034-EB was recommended. Simply install a VGA card in the expansion box, hook everything up, select enhanced

Continued on Page 48

Robert C. Brenner, MSEE, MSSM
1991 Brenner Information Group
9282 Samantha Court
San Diego, CA 92129



Webster's New Collegiate Dictionary defines a standard as something "setup and established by authority, custom, or general consent as a model or example." It calls a protocol a "code prescribing strict adherence to correct etiquette and precedence." The distinction between standards and protocols is confusing and difficult, even for modem manufacturers.

However, we can agree that, in telecommunications, standards define how data should be handled. Standards originate from three sources: official government-sponsored groups, manufacturers, and the marketplace. From the American National Standards Institute (ANSI) we got ASCII; from the Electronic Industries Association (EIA) we got the RS-232C serial cable definition; and from the Consultative Committee for International Telephone and Telegraph (CCITT) we got the "Vee-dot" modem modulation, data transfer, error correction and data compression standards. There's plenty of overlap between standards-generating organizations. The RS-232C of the EIA is equivalent to V.24 and V.28 of the CCITT. Often standards for one modem scheme are not compatible between two organizations.

The marketplace can also establish de facto standards as customers adopt certain methods to perform a function. Hayes Microcomputer Products designed a modem command set that became so popular it was adopted by other modem manufacturers making "Hayes-compatible" a selling feature for non-Hayes modem products.

Twenty years ago there were two primary sources for modem standards, the CCITT and American Telephone and Telegraph (AT&T) Bell system. In the 60s, all U.S. modem standards were set exclusively by the Bell Telephone monopoly of AT&T. Then the 1968 Carterphone court decision opened the market to other modem manufacturers, and standards changed

as the telecommunications world sought design stability, while modem manufacturers sought competitive advantage.

International modem communications are covered by the "V." (pronounced "Vee-dot") series of recommendations from the CCITT. For example, V.32 covers 9600 bps modems. To enhance certain recommendations, the CCITT also issues addendums that are identified by the suffix "bis" (French for "second version") or "ter" (third version). Thus V.42bis is an addendum covering data compression. It compliments the V.42 recommendation on error correction.

Since the Bell breakup, AT&T and the CCITT have attempted to introduce compatible standards, but as our world became more integrated, the CCITT standards have taken precedence. When AT&T developed the modem, they chose not to coincide with CCITT, so all Bell-compatible devices don't necessarily work with their European counterparts. Plenty of confusion still exists in the standards world. Technology has reached the modulation capabilities of a standard telephone line, so current standards are addressing error control and data compression techniques to put more data across in a fixed time and frequency bandwidth.

Today, the increase in confusion matches the increase in data transfer speed. There are at least seven types of 1200 bps modems in

two basic categories of modulation: FSK and PSK. Then, when 9600 bps modems were introduced, no standard existed. So modem manufacturers designed and built products to their own standards while the so-called "standards groups" raced at molasses speed to define the characteristics and performance requirements for this evolving capability. As a result, most early 9600 bps modems can only communicate with modems of the same manufacturer and type.

Evolution is never easy. And the evolution of standards has been a difficult challenge for designers the world over. It's actually amazing that we're able to communicate as well as we do. Yet, the powerful market forces established by the drive to a world economy are pushing technology (and bureaucratic organizations) to develop interface techniques that work for all of us. Table 1 describes the evolution of modem signal modulation standards. The following paragraphs describe the most common modem standards in use today.

BELL 212A is a de facto modulation standard for 1200 bps PSK data transfer

The Standards Evolution

AT&T / BELL	CCITT	INDUSTRY
Bell 103	V.21	
Bell 113		
Bell 201	V.26	
Bell 202	V.23	
Bell 208	V.27	
Bell 209	V.29	
Bell 212	V.22	Radal Vadic
	V.22bis	Vadic 3400
	V..32	
	V.32bis	

Table 1
The Evolution of Modulation Standards.

over common telephone lines. A Bell 212A modem is full duplex, synchronous, and downward compatible with 300 bps Bell 103 FSK devices. Bell 212A operates over 2-wire leased or dial up lines.

CCITT V.22 is a 1200 bps PSK standard similar to Bell 212A for two-wire full duplex synchronous or asynchronous transmission. At 300 bps it stops being compatible with Bell 212A because Bell 212A shifts to FSK, while V.22 remains in PSK. V.22 accommodates synchronous mainframe computers and terminals and asynchronous PCs found in many "hybrid" networks.

CCITT V.22bis is a revision to V.22 that adds the capability to operate at up to 2400 bps doubling the file transfer speed of V.22. It supports 2400 bps ops over two-wire links, at full duplex and with an alternative rate of 1200 bps.

CCITT V.29 is a 9600/7200/4800 bps QAM standard that transmits in only one direction at time (half duplex) over 2-wire networks (full duplex over 4-wire leased lines). For the typical PC user, V.29 modems are half duplex, dial-up, and synchronous, although some V.29-compliant modems incorporate buffers to simulate full duplex (for slow, reverse flow). They transmit asynchronously, use flow control protocols and are compatible with Bell 103, Bell 212A, and V.22bis. V.29 modems plug into the PC's asynchronous serial port and appear like fast Hayes-compatible modems. For synchronous operation on two- or four-wire lines V.29 operates half duplex at 9600 bps. It doesn't add error-handling routines, but relies instead on automatic adaptive equalization to prevent transmission faults, and relies on synchronous data transfer protocols to catch them. The half duplex implementation of V.29 is used by most fax machines to send information in one direction at a time.

Hayes Microcomputer Products added buffers to their 9600 bps V.29 modem to allow asynchronous communications between the modem and the PC while data was being transferred synchronously in the other direction over the telephone line. To make their original 9600 bps modems behave as if full duplex, Hayes implemented ping-pong, in which two identical half-duplex Hayes Smartmodems take turns transmitting at 9600 bps without echo cancellation. Ping-pong works for file transfers with clean lines because data is sent uni-direc-

tional. But error detection and data retransmission are actually faster in full duplex through links that include satellites. Simulating full duplex using ping-pong caused compatibility problems when trying to communicate with a different modem manufacturer's product. Hayes also added trellis coding to improve noise immunity, adaptive data compression, and LAP-B error correction. LAP-B (normally used on X.25 networks) is a block error checking protocol that automatically corrects transmission errors. Hayes further sped operations with V.29 "V Series" modems by sending data in synchronous mode (no start and stop bit per byte overhead). The first Hayes V-series modems required opposite end V-series counterparts so point-to-point communications were the optimum application to access the features of this modem series.

CCITT V.32 is a 9600 bps QAM standard for asynchronous or synchronous, full duplex operation with fallback to 4800 bps and reduced speed capability at 2400, 1200, or 300 bps. V.32 is the most important development affecting dial-up modems in recent years and is the first universal standard for 9600 bps operation on both leased and dial-up lines. It coincides with the rapid acceptance of 9600 bps products and is the standard today for high-speed networks. Modems supporting V.32 must incorporate error control, and echo cancellation. They must operate in full duplex synchronously or asynchronously at transfer rates up to 9600 bps. Beyond these specifications, the differences are value-added features. Full duplex is not possible without "echo cancelling" to filter a modem's own tones out of the line signals and allow both transmitted and received signals to co-exist.

The 9600 bps data rate takes up most of the 3100 Hz bandwidth available on the typical dial-up telephone line, so V.32 modems can't use split-band techniques to maintain full duplex operation. Instead, sophisticated echo cancelling is implemented using digital signal processing (DSP) circuitry to cancel out the strong echo signal from the transmitter so the sending unit can detect the relatively faint incoming signal from the remote modem. Because the telephone company converts two-wire dial-up lines into four-wire lines for transmission to other exchanges, V.32 generates a special tone that temporarily dis-

ables the telephone company's echo-suppression circuits (used during normal voice transmission).

V.32 modems send and receive 4-bit data constellations (QAM) at 2400 baud achieving an effective rate of 9600 bps with a fallback to 4800 bps. Trellis coding (TCM) is used on most (not all) V.32 9600 bps modems to provide error-reduction during signal modulation. With TCM, these modems check for transmission errors using a redundancy bit. This makes the transmissions less vulnerable to data errors over poor telephone lines. V.32 requires that TCM-capable modems still work with non-TCM-capable modems.

V.32 has no provision for error correction, just detection. All V.32 modems handle error-checking, data-compression, and data link communication protocols. A V.32 modem stores data from a PC in a buffer and then collects it in blocks for synchronous transmission. To signal the PC that the buffer is full, the modem uses XON/XOFF and RTS/CTS handshaking signals. RTS/CTS is the choice for high speed modems and binary data transfer.

To maintain continuous communications, users sometimes opt for a V.32 modem that connects to a leased line with automatic dial-up line backup and look-back. If the leased line fails, the modem will automatically switch to a dial-up line. This feature is good for users who need continuous connection (e.g., financial institutions and brokerage houses). The look-back feature is associated with dial-up line backup. After switching to a dial-up line, the modem monitors the leased line that has failed. When line conditions improve, it reverts back to the leased line.

CCITT V.32bis was developed in 1989 (adopted March 1991) to let V.32 QAM modems operate at up to 14,400 bps in full duplex with alternative rates of 12,000, 9600, and 7200 bps. A V.32bis modem can transmit data simultaneously in two directions using two separate data transmission paths. Transmission can be synchronous or asynchronous over two-wire dial-up or private lines. V.32bis increases V.32 speed over 50 percent by defining the patterns and using increasingly robust echo cancellation. This standard eliminates equalizing and echo cancelling during speed changes to improve the process and enable two channels to connect and effectively pass data at 38,400 bps.

SPEED (bps)	300	1200	2400	4800	9600
BELL STD	103J/113D	202S/212A	201B,C	208A,B	209
CCITT STD	V.21	V.22,V.23	V.26	V.27	V.29,V.32
MODULATION	FSK	FSK	PSK	PSK	QAM
DUPLEX	Full	Half,Full	Full	Half,Full	Half,Full
SYNCHRON	Async	Async,Sync	Async,Sync	Sync	Sync

Table 2
Characteristics of the Common Modem Standards Used with PCs.

V.32bis uses the same trellis coding modulation common in standard V.32, but V.32bis requires better echo cancelling and receiver performance. V.32bis features on-line rate negotiation so a modem can automatically increase or decrease its speed during transmission depending on line conditions. These modems also support the CCITT V.42bis data compression standard with its 4:1 data compression ratio. This enables transmission speeds close to 64 Kbps, the speed of current leased lines. Soon, most new modems will support V.42bis and V.32bis and achieve effective throughput beyond 56 Kbps.

V.32HDX is a 9600 bps half duplex proprietary standard from Hayes Microcomputer Products. Early Hayes V-series Smartmodem 9600 modems simulated full duplex V.32 operation by alternately sending and receiving data in "ping-pong" fashion. The Hayes V.32HDX label confuses modem users on its true V.32 compliance. V.32HDX incorporates some of V.32, but not all. It implements QAM modulation and transfer speed negotiation, but operates in half-duplex mode. No true simultaneous transmission occurs as in full duplex operation. Because the originating and answering modems take turns transmitting, no echo-cancellation circuitry is used. V.32HDX is acceptable for data transfer over clean lines, but slows down when satellite echoes are present. Because this is not a "true" V.32 standard, the V-Series Smartmodem 9600 with V.32HDX earned a category of its own as a high-speed modem.

Table 2 describes the features of the common modem standards.

CCITT V.42 is a 9600 bps error correcting standard using a 32-bit transfer technique that requires support for Link Access Protocol for Modems (LAP-M) and the Microcom Networking Protocol (MNP) error correction schemes on all information exchanges. V.42 was approved in January 1990.

Trellis code modulation handles errors during signal modulation, while V.42 detects and corrects errors during and after transmission. Since the checking and retransmitting functions are built in, error handling is transparent to the user. This increases data throughput, but when numerous errors are detected, the effective throughput is decreased by data block retransmission.

If a modem offers either MNP or LAP-M at 9600 bps, it's called "compatible." If it offers both MNP and LAP-M, it's called "compliant." Compliant is better because V.42-compliant modems also incorporate data compression. A V.42 compatible modem won't recognize LAP-M, so linking it to a V.42 compliant modem causes MNP error control to kick in, but not LAP-M. A V.42 compliant modem first tries to establish LAP-M error control protocol. If not

successful, it shifts to MNP error control. V.42 specifies RTS/CTS and XON/XOFF local flow control for the data terminal equipment (DTE) interface.

Besides error correction, there are data compression standards that squeeze information to effectively increase the transfer rate. Data compression can save thousands of dol-

lars in telephone charges if huge files are routinely sent over dial-up or leased lines. Data that is designed into patterns, such as forms and spreadsheets, can be compressed so much that 9600 bps can effectively achieve a 19,200 bps transfer. Random data, such as text or computer programs, can be passed over dial-up lines at 15 Kbps - 16 Kbps.

MNP5 is a data compression technique that implements a real-time adaptive data compression algorithm to realize net throughput of 200 percent (on average) more. The algorithm uses both Huffman encoding and run length encoding (RLE) to optimize both interactive terminal data and file transfer data by continuously analyzing the data and adjusting the compression parameters to maximize throughput.

Huffman converts the most frequent characters to 4-bit codes and adds longer codes depending on the frequency of use. Less frequent characters are converted into longer codes (up to 11 bits), while still achieving compression of the complete file.

Run length encoding compresses repetitive character strings including the non-printing characters such as line feed, carriage return, and spaces. When it encounters the same three or more characters in the same row, the RLE algorithm will send the characters and the number of repeats that occur. This is excellent for spreadsheets and other files that have many repeating non-printing formatting characters.

MNP5 can apply 2:1 data compression at 2400 bps to achieve an effective 4800 bps transfer. Microcom claims that its data compression algorithms can obtain a 60% increase in average throughput when the modems at each end of the transmission channel both use the same data compression algorithms.

Software that implements MNP5 must provide data compression and error handling over the entire link. MNP5 provides

Data Form	Approximate Compression Ratio
ASCII text	3.1
Assembly language code	3.4
Images/graphics	2.8
Precompressed (.ARC)	1.3
Program source code	3.3
Random data	1.0
Spreadsheet/database	2.8

Table 3
Data Compression Capability Using V.42bis.

the same error correction found in V.42, but with hardware data compression added. MNP5 was adopted by the CCITT as a standard for data compression.

Telebit and U.S. Robotics implement MNP5 in their modem products. Once active, MNP5 must be manually deactivated. Telebit modems use MNP up to 9600 bps. Above this they use Lempel-Ziv integrated into their own proprietary PEP protocol.

High speed modems with MNP5 data compression can operate at speeds higher than 9600 bps. To aid in this process, the PC to modem connection is set to at least 19.2 Kbps (38.4 Kbps if transferring large ASCII files).

MNP7 enhances the MNP5 data compression algorithm to achieve an average 300 percent speed improvement. MNP7 uses Huffman data compression with a predictive algorithm that represents user data in the shortest Huffman codes. It also includes a Lempel-Ziv data compression option common to European modems to achieve 400 percent efficiency improvement. The MNP7 data compression protocol enables throughput rates of up to 12 Kbps over dial-up lines. MNP7 encodes characters according to the frequency of character pairs. This gives MNP7 the capability for 3:1 compression, so a 2400 bps modem can achieve an effective throughput of 7200 bps.

CCITT V.42bis is a dial-up data compression standard that achieves 3:1 or 4:1 data compression for 9600 bps transmissions. V.42bis was formally adopted in February 1991 as the first CCITT sanctioned method for modem data compression and decompression. With it, users can achieve effective transfer speeds close to 64 Kbps on leased lines and 34,800 bps on dial-up lines. V.42bis is the new standard for data compression and is being incorporated into all new high-end modems. To support V.42bis, 9600 bps modems must

incorporate LAP-M protocol.

V.42bis is based on the Lempel-Ziv data compression algorithm. This is the same algorithm used in the .ARC and .ZIP data compression software. Actually, V.42bis incorporates all four versions of the Lempel-Ziv compression utilities (LZ1, LZ2, BTLZ, and HBTLZ). V.42bis modems process data as it flows from one modem to the other.

Because V.42bis is software intensive and doesn't require extensive hardware redesign, most new modem products incorporate V.42bis. It's a common feature in V.32 and V.32bis modems. The data compression employed is similar to the algorithm found in shareware compression programs, but it's accomplished in real time within the modem circuitry. To use V.42bis data compression efficiently, the comm software must support a streaming file transfer protocol such as YModem-G or ZModem.

With a 4:1 compression ratio, V.42bis

senses when a file is already compressed and doesn't try to re-compress files that are already compressed. The capability of V.42bis can be seen in Table 3 by comparing the amount of compression that can occur on various forms of data.

Software protocols such as XModem, YModem, ZModem, and Kermit perform data compression only during file transfers. Modem designers have implemented MNP5 and V.42bis in ROM (firmware) so these algorithms work all the time. If a modem has MNP5 active, you should avoid running a pre-compressed file through the process. The V.42bis protocol will recognize an attempt to compress a pre-compressed file and not implement compression on that file.

MNP5 provides 2:1 compression; V.42bis provides 4:1 data compaction. This lets a 9600 bps V.42bis modem achieve 38,400 bps effective throughput. However, V.42bis and MNP5 cannot be active without MNP4 asynchronous error correction

also in effect.

It's important to remember that V.42bis senses when a file is compressed; MNP5 does not and must be de-activated if you're transferring pre-compressed files.

The emerging national and international standards are often confusing and conflicting, but efforts are being made to reach consensus. It's really difficult for manufacturers and users to keep track of the many "standards." This is likely to grow even more difficult as newer technologies emerge. Yet standards represent an opportunity to improve communications, and each should be taken seriously if you want to communicate globally. In the next article, we'll look at protocols and investigate those most prevalent in communications today.

This article is paraphrased from *Modems Made Easy* by Robert C. Brenner. Available for \$19.95 from Brenner Information Group, 9282 Samantha Court, San Diego, CA 92129. ✽

Continued from Page 42

image to a graphic font file, specifying a cell width and height of 100 dots.

Sled will save the image assigning each symbol to an ASCII character equivalent. You can then use a program supplied with SLED to print a chart showing the assignments. When you want to use a symbol, change to that font using your word processor's techniques and press the appropriate keyboard character.

Selecting Font Tools

As you can see, each of these three programs offers unique features.

Font Solution Pack is perhaps the strongest of the three if you are interested in special graphics or font effects. The Spinfont and Font Effects modules are particularly useful and entertaining. They can generate fonts and graphics that have a dramatic impact on your readers.

Publisher's Type Foundry excels in providing precise control over font characteristics and techniques of digital type design. It is an invaluable tool if you like to design characters in detail.

SLED is an excellent tool for creating and printing graphic images, for scaling bitmap images as outlines, and printing

with software that does not internally support graphic images. SLED offers capabilities not found in most drawing and printing programs.

There are many other excellent font and graphic tools that can be invaluable if you own a laser printer. I selected these three only because they illustrate a variety of approaches and functionality. In selecting a tool for your own use, consider which aspect of the functions described here are most important to you and what existing tools you already may have available. ✽

Continued from Page 44

graphics in the expansion chassis setup screen and the end result: full color. **Note:** There are no special drivers necessary.

There are, however, some peculiarities with various expansion boxes. One important fact to remember is to turn on the power to the expansion box before you turn on the computer. One caller had a problem after he purchased his Bargain Centre expansion box. His floppy controller was not working and he had to press ESC to continue. Once the VGA card was in default CGA mode, he couldn't access the floppy drive to run configurations for various programs. The fix: A Zenith Data Systems dealer could not find anything wrong with the expansion box so he swapped controller cards.

Things were further explained by another caller. Apparently, some of the laptops use a pull-up on the interrupt line in the expansion box and others use a pull-down. If the polarity is wrong, the machine will give an error message. We were also in-

formed that at one point ZDS changed the backplane card in the expansion box, adding a jumper to select one or the other. Depending on how old the expansion box is, either a new backplane board (in the expansion box) or a change in the jumper may be necessary.

Upgrading a 386/16 Motherboard

To upgrade a Z-386/16 from 16 MHz to 20 MHz, the procedure itself is rather simple. What one ZUGgie did was simply replace the 32 MHz oscillator with a 40 MHz. There are, however, different oscillators, as well as different CPU boards. Problems arose when the upgrade was attempted with an older CPU card, part number 181-7043-20-1.

Apparently, one oscillator is half-size while the other one has a full size, 14-pin socket. With the older card, it was recommended to desolder the old full size oscillator and replace it with a 14-pin socket, removing pins 2-6 and 9-13. After placing the new oscillator in the socket, it appeared

as if the problem was solved.

This procedure seemed to work for one ZUGgie, but the other still had some problems. One advantage of this column is to look at situations and possibly come to a real workable solution for a number of people. If anyone has an alternative solution to the 16 to 20 MHz upgrade problem, please let others know, either by calling COM1 or writing to me directly.

This month, I would like to thank George Bernegger, William Brooks, Jr., Craig Stevenson, Pat Swayne, Robert Cesares, Kurt Erhardt, Brian Hansen, Scott Helmann, Hank Komisarz, Robert LaMoureaux, Brad Martin, Mike Stover, Rick Terrill, Jim Williams, Frank Zinghini and Roland Zuk. If you have anything you want to share or have a problem that needs a remedy, you know what Port-O-Call to use. ✽

A decorative border of stylized orange and yellow flames surrounds the central text. The flames are rendered with thick black outlines and a gradient from yellow to orange. The word "HADES II" is written in a large, red, hand-drawn, scribbled font in the center of the page.

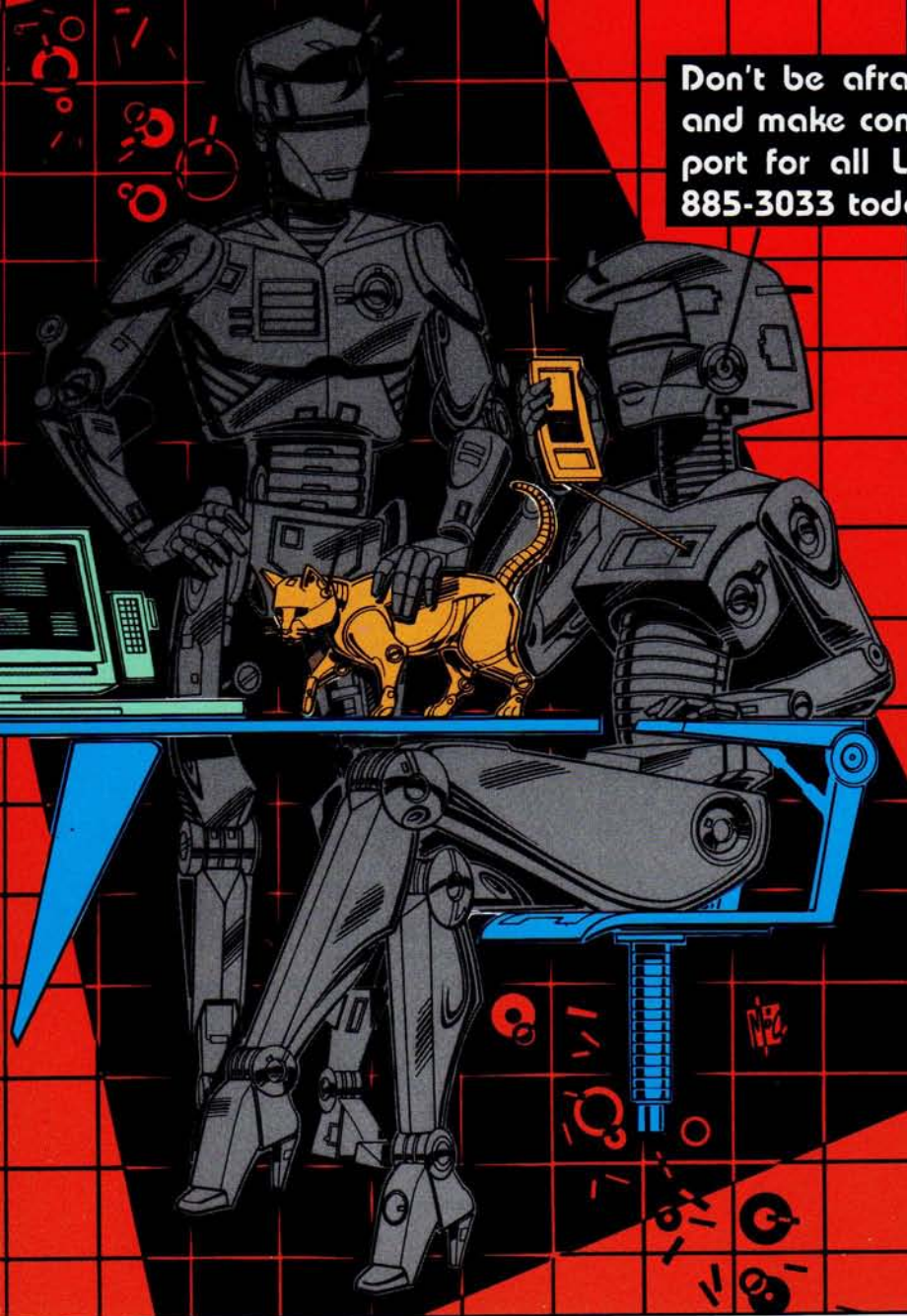
HADES II

It's HOTTER than ever! Jam-packed with new features, HADES II still remains the easiest-to-use disk editor ever! Just look at some of the features:

- Sector Display/Editing
- Sector HEX/ASCII String Search
- File Display/Editing
- Physical and Logical Cluster Display
- File HEX/ASCII String Search
- Drive Parameter Display
- 512 MegaByte Drive Size Limit
- File Attribute Display/Edit
- Automatic Erased File Recovery
- Manual Rebuild File Recovery
- Works with Headerless MS-DOS Disks
- PC-Compatible or H/Z-100

HADES II is still only \$40, and original HADES owners can upgrade their distribution disk for only \$15. Call HUG today at: (616) 982-3463.

Don't be afraid to communicate! Get HUGMCP and make contact the easy way. Now with support for all Laptops, order HUG Part number 885-3033 today.



HUGMCP Commands

- F1 -- Prints This List, Your Storage Buffer Size, And How Many Bytes Are Presently In The Storage Buffer.
- F2 -- Allows Sending A Defined Message, Or Character Sequence. These Messages Are Entered Using The (F3) Setup Command.
- F3 -- Toggles The Storage Buffer On and Off. When The Buffer Is On, The (Buf) On The 25th Line Will Be High-Lighted.
- F4 -- Allows Saving Data To Disk From The Storage Buffer, Or Directly From The Modem By Way Of XMODEM Protocol.
- F5 -- Allow Sending Data From Disk, Using Either XMODEM, Which Optionally Can Be Ignored, Or XMODEM Protocol.
- F6 -- Enters The Setup Mode So This Software Can Be Configured.
- F7 -- Clears Out Any Data That May Be In The Storage Buffer.
- F8 -- Send Data In Storage Buffer To Printer.
- F9 -- Exits Back To MS-DOS.

Storage Buffer : 524288 Bytes
Storage Buffer Usage : 0 Bytes

Select Message (A-0), (F1) To List, Anything Else To Abort --> _

F1:Hp F2:Msg F3:Buf F4:Save F5:Send F6:Copy F7:Clr F8:Print F9:Exit COM

HUGMCP Configuration Menu #1

- A -- This Function Allow The Baud Rate To Be Changed Depending Upon Which Modem You're Using. Normally It Would Be Set To Either 1200, 1800, Or 2400 Baud. Great Connector To A Host. Will Allow Higher Baud Rates.
- F -- This Function Allow You To Change The Word Parity. Normally you Would Change No Parity. This Is Acceptable By Most Remote Systems, But It Is Also Necessary For XMODEM Protocol To Work Properly.
- C -- This Function Allow The Changing Of The Word Length. Normally The Length Should Be Set To 8 Data Bits. This Value Is Acceptable By Most Remote Systems, And Is Necessary For XMODEM Protocol To Work Properly.
- S -- This Selection Allow You To Enter Messages Which Can Be Automatically Sent With The (F3) Key. Up To 14, 32-Character Messages Can Be Saved. Selection 14 Is Special. It Should Contain Your Computer's ID Number And Password. Selection 16 Is Also Special. This Selection Can Automatically Be Sent when This Program Is First Executed By Selecting The Proper Option During Setup.

Type (SPACE BAR) For More Help, Anything Else To Continue

F1:Hp F2:Msg F3:Buf F4:Save F5:Send F6:Copy F7:Clr F8:Print F9:Exit COM

HUGMCP Configuration Menu:

- A --> Modify Baud Rate
- F --> Modify Parity Type
- C --> Modify Word Length
- S --> Modify Or Add Auto-Messages
- M --> Miscellaneous Functions
- W --> Change Screen Color Assignments
- D --> Display Current Configuration
- H --> Make Changes Permanent

Select A-C, (F1) For Help, Anything Else To Quit --> _

Baud Rate: 19200
Parity: NONE
Word Length: 8
Duplex: FULL
Response To Keyboard Disable: NO
Storage Buffer Data Parity Bit: SET TO ZERO
Send Modem Initialization text: NO
Delete Character: W0000L
Modem Port Set to: COM1

F1:Hp F2:Msg F3:Buf F4:Save F5:Send F6:Copy F7:Clr F8:Print F9:Exit COM

ZENITH
data systems



Groupe Bull

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

POSTMASTER: If undeliverable, please do not return.

\$2.50
P/N 885-2140