REMark

April 1991

The Official Zenith Data Systems Computer Users Magazine

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.

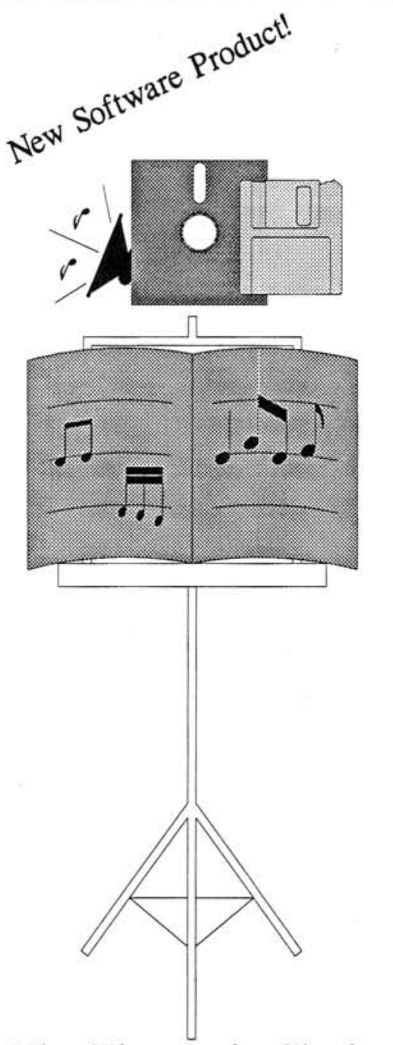
April 1991

REMark®

The Official Zenith Data Systems Computer Users Magazine

Advertising

Reader Service No.		Page No.
250	B.U.D. Unlimited	16
104	FBE Research Co., Inc.	19
136	Lindley Systems	41
149	W.S. Electronics	28



The Electronic Clavier
P/N 885-6016

Managing Editor
Jim Buszkiewicz
(616) 982-3837

Software Engineer
Pat Swayne
(616) 982-3463

Production Coordinator
Lori Lerch
(616) 982-3794

Secretary
Lisa Cobb
(616) 982-3463

COM1 Bulletin Board
(616) 982-3956
(Modem Only)

ZUG Software Orders
(616) 982-3463

Contributing Editor
William M. Adney

Printer
Imperial Printing
St. Joseph, MI

Contributing Editor
Robert C. Brenner

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

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

* U.S. Funds

Limited back issues are available at $2.50, plus 10% shipping and handling - minimum $1.00 charge. Check 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

Software

H8 - H/Z-89/90

PRODUCT NAME	PART NUMBER	OPERATING SYSTEM	DESCRIPTION	PRICE
ACTION GAMES	885-1220-[37]	CPM	GAME	20.00
ADVENTURE	885-1010	HDOS	GAME	10.00
ASCIRITY	885-1238-[37]	CPM	AMATEUR RADIO	20.00
AUTOFILE (Z80 ONLY)	885-1110	HDOS	DBMS	30.00
BHBASIC SUPPORT 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 PROCR	35.00
SCICALC	885-8027	HDOS	UTILITY	20.00
SMALL BUISNESS 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)

PRODUCT NAME	PART NUMBER	OPERATING SYSTEM	DESCRIPTION	PRICE
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

PRODUCT NAME	PART NUMBER	OPERATING SYSTEM	DESCRIPTION	PRICE
H/Z-100 (Not PC) Only				
CARDCAT	885-3021-37	MSDOS	BUSINESS	20.00
CHEAPCALC	885-3006-37	MSDOS	UTILITY	20.00
CHECKBOOK MANAGER	885-3013-37	MSDOS	BUSINESS	20.00
CP/EMULATOR	885-3007-37	MSDOS	CPM EMULATOR	20.00
DBZ	885-8034-37	MSDOS	DBMS	25.00
DUNGN & DRAGONS (ZBASIC)	885-3009-37	MSDOS	GAME	20.00
ETCHDUMP	885-3005-37	MSDOS	UTILITY	20.00
EZPLOT II	885-3049-37	MSDOS	PRINTER PLOT UTIL	25.00
GAMES (ZBASIC)	885-3011-37	MSDOS	GAMES	20.00
GAMES CONTEST PACKAGE	885-3017-37	MSDOS	GAMES	25.00
GAMES PACKAGE II	885-3044-37	MSDOS	GAMES	25.00
GRAPHIC GAMES (ZBASIC)	885-3004-37	MSDOS	GAMES	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	ATRONOMY 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

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.

Now Available!
ZUG software is now available on 2" disks. Just put a "-90" at the end of the part number (i.e., 885-6014-90). Also add $3.00 to the purchase price of the software (i.e., $20.00 + $3.00 = $23.00).

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

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.

ZENITH data systems

Groupe Bull

REMark Magazine Subscription & ZLink/COM1 Bulletin Board Information

Your subscription entitles you to receive REMark, our monthly magazine containing articles specific to Zenith Data Systems computer and generally to other PC Compatible computers. All articles in REMark are submitted by readers like you. We welcome YOUR articles, and will pay you for any we accept!

A REMark subscription also allows you full access to the ZLink-COM1 bulletin board system (COM1, for short) described in detail in the brochure. There are many, many megabytes of free and shareware software available for downloading to registered COM1 users. Full access also lets you order products from the "Bargain Centre" section of COM1. The money you can save in the Keyboard Shopping Club will pay for decades of REMark subscriptions.

Last, but definitely not least, your subscription puts you in touch with thousands of other Zenith Data System computer users, from whom invaluable information can be exchanged.

REMark subscriptions, currently $22.95, can be obtained in one of three ways. First, by ordering one on the COM1 bulletin board (see the Keyboard Shopping Club section); second, by phone with VISA, MasterCard, or American Express; and third, through the US Mail using a credit card, money order or check made payable to: Zenith Data Systems. Our address is:

Zenith Data Systems Users' Group
P.O. Box 217
Benton Harbor, MI 49023-0217
(616) 982-3463

Once you receive your ID number, registration on the COM1 BBS is NOT automatic. It requires that you log on, enter your first name and last name EXACTLY as they appear on your REMark mailing label, and then enter your ID number as your password. The FIRST time you access the board, you must elect to start a NEW ACCOUNT and answer the various questions. Once you've done this, our automated scanner will compare the system's database against the subscription database. If you made no mistakes, you will be verified and given full access, within 24 hours.

Once you've been authorized as a full member, several important things happen. First, you're given full downloading privileges of up to one megabyte per day. Secondly, you'll have full access to the message boards. And finally, you'll be able to take full advantage of the Bargain Centre product savings.

Detach this form, enclose your check, money order or credit card information (no cash please).

REMark Subscription / Renewal Form

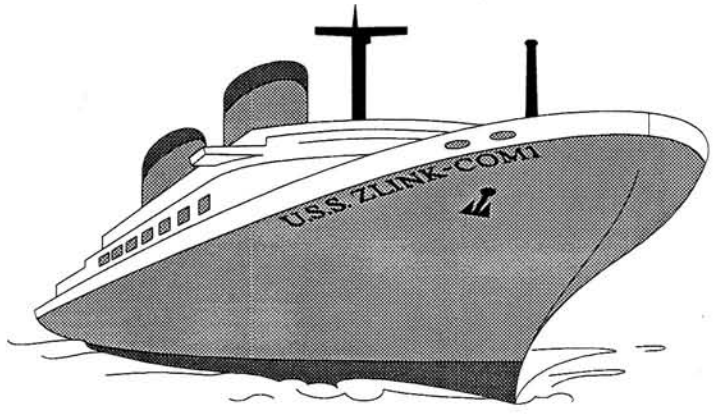
New Member: ☐ Yes ☐ No   Credit Card # ____________________

ID Number: ____________________   Exp. Date ____________________

Address Change? ☐

		Renew	**New**
Name: ____________________	U.S. Bulk Mail	☐ 19.95	☐ 22.95
Address: ____________________	U.S. First Class	☐ 32.95	☐ 37.95
City, State, Zip: ____________________	APO/FPO Surface Overseas	☐ 32.95	☐ 37.95
Daytime Phone #: ( ) ____________________	Air Printed Overseas	☐ 52.95	☐ 57.95

Port-O-Call: COM1



Laura White
759 Polfus
Benton Harbor, MI 49022

If you've been wanting a faster computer without purchasing an entirely new machine, the Heath Computer Upgrade kit is for you. Model HUG-386-25 converts your H-248, H-386 or H-386-3 computer into a 25 MHz, 32-bit, 80386-based computer.

The kit, includes a four-drive chassis, 32-bit 80386 replacement main board, 32-bit input/output board a 32-bit 16K cache memory board, and is used in conjunction with your computer's present power supply, drive controller and video boards. Because the new main board is larger by design, old video boards like the Z-409 CGA video cards will not fit into the slots — so plan on purchasing a newer video board if you're upgrading. In addition, the new board is compatible with either the Intel 80387 or Weitek WTL-3167 numeric coprocessor and specific instructions are included in the kit to insure proper installation.

The main board is the real key to getting more speed out of your machine. Equipped áwith two, 32-bit 1M single in-line memory modules (SIMMS), the board also has the fast "Enhanced Page-Mode" RAM technology. When these features are combined with the advanced paging controller and AT-superset 32-bit memory bus, memory access time is significantly reduced. In addition, to decrease any remaining memory access wait the kit includes a 16K cache memory board.

For those of you who are in constant need of more memory, the upgraded system has a maximum capacity of 64M. You need to be informed however, that there is an admitted bug which will be corrected in the eight slot version of the motherboard. The motherboard in the upgrade kit will not recognize a two or four meg SIMM in the first four sockets. It will however recognize these larger modules in the last four sockets giving you a maximum total of 20M on the motherboard and 44 on memory expansion cards. You can expand the system memory in either 1M, 2M or 4M increments.

Still looking for more? If you utilize applications which require more than the 640K limit of MS-DOS, the upgraded system has the Lotus-Intel-Microsoft Expanded Memory áSpecification (EMS). In addition, the main board's EMS 4.0 hardware implementation makes up to 256K of memory available for those software packages, like Windows 4.0, which are written to take advantage of expanded memory.

If you are wondering how all this will interact with you computer's present equipment—namely your power supply, your worries are all for nothing. Your computer's original 200-watt switchiing-type regulated power supply provides enough reserve power for the maximum 64M of system memory, two floppy disk drives, and two hard disk drives.

You do need to be aware however, that because the new main board is so much larger than the old one, you can only install one full-height, three inch high, hard disk drive in your new upgraded computer, unless you are willing to give up your second floppy drive. You see, the new motherboard occupies the same real estate that is required by a full-height drive on the inside cage in the lowest position. The maximum configuration allowed may include three half-height devices and one full-height device or, two full-height and one half-height devices.

Even though you are limited when it comes to your full height hard disk, the number of other accessories you can install more than makes up for it for the limitation. You have your choice of one or two hard disk drives model numbers HWD-4028 (5-1/4", 40M, 28ms drives) or ZD-800 (5-1/4", 80M 40ms drives). You can also install any combination of one or two of the following drives: 5-1/4", 1.2M (ZD-12); 5-1/4", 360K (Z-207-7); or a 3-1/2" 1.4M (ZD-144 installed in a 5-1/4" bracket. As was mentioned before, you can add memory by putting in a memory expansion board, model number ZA-3600-MQ, which holds up to four 2M SIMMS or 4M SIMMS when they become available.

You not only get a wide variety of choices in drives, but you also get a choice when it comes to monitors. With a 31 kHz dual-port VGA video board you can drive VGA, EGA, CGS, MDA, or Hercules monitors, Model HVB 550. That includes everything from a 31 kHz Monochrome Monitor, model number ZMM-149-A or P, to a flat technology monitor, model number ZCM-1492.

Installation of the upgrade is relatively quick and easy. After removing the circuit boards and the backplane board, you can then remove both disk drives. Next, disconnect the cables from the hard drive labeling cables appropriately. (You want reassembly to go as smoothly as possible when it comes to cables and connections.) Finally, you need to remove the power supply, the lock assembly and the card guide. That's all there is to it.

Take note however, to use caution when removing circuit boards. Do not let go of a

Continued on Page 32

...the other cats get to sing along!

And now, HEPCAT does Windows!

HEPCAT, the powerful, versatile pop-up calculator from ZUG, is now even better. HEPCAT not only can pop up over just about any DOS program you are likely to use, but now version 2 can also pop up over Windows™ 3, even when it is running in the standard (286) or 386 enhanced modes.

What Is HEPCAT?

HEPCAT (Handy Engineer's and Programmer's CAlculation Tool) is a floating point calculator with several scientific/engineering features built in, and a binary (programmer's) calculator combined into one tiny, powerful program. HEPCAT is a memory resident program that "pops up" on your screen whenever you activate it by typing a special "hot key" sequence.

The Other Cats Can Sing Along

Unlike other pop-up calculators, HEPCAT is concurrent. That means that when you pop it up over a running program, the program can continue to run. For example, if you pop it up while Lotus™ is busy loading a huge spread sheet, it will continue loading while you perform your calculations. And HEPCAT always pops up in the current video mode, rather than forcing the screen into a text mode like other pop-ups do. HEPCAT can pop up in any standard CGA, EGA, VGA, or Hercules™ graphic mode, as well as in any text mode. It can even pop up in some non-standard graphic modes (but it may not clear its window when you exit).

HEPCAT Works Harder

The floating point calculator in HEPCAT includes the following built-in functions: powers, pi, factorial, square root, sine, arc sine, cosine, arc cosine, tangent, arc tangent, log (natural and base 10), e^X and 10^X, and it does rectangular-to-polar and polar-to-rectangular coordinate conversion. It also includes several built-in US-metric and metric-US conversions. The binary calculator works in these number bases: binary, tetral (base 4), octal, split octal, decimal, and hexadecimal; and it supports these operations: MOD, AND, OR, XOR, SHL, SHR.

The HEPCAT floating point calculator supports 8 significant digits and can display numbers four ways: floating point, fixed point, scientific notation, and engineering notation. Numbers are handled internally in BCD format to eliminate binary round off errors in addition and subtraction.

HEPCAT Eats Less

HEPCAT uses less than 18k of memory – less than any other pop-up calculator that we know of. It also uses less than 14k of disk space, so you don't have to worry about where to put it on a small system. HEPCAT is easier to learn, too, with commands that make sense.

If you are tired of pop-up calculators that can only sing solo, or calculators that can do DOS but not Windows (or Windows but not DOS) give HEPCAT a try. HEPCAT is available from ZUG as part no. 885-3045 for $35.00 (plus S/H). It works on any Zenith Data Systems computer that runs MS-DOS or Z-DOS (including the Z-100 series), and on most PC-compatibles. If you have HEPCAT version 1, send in your original distribution disk and $10 to upgrade to version 2.

New and Improved

HARVARD GRAPHICS

Earl R. Zimmerman, Jr.
78 Wells Drive
Dayton, OH 45431

Introduction

Back in May 1989 I reviewed Harvard Graphics 2.12 for REMark. The review talked about how easy Harvard Graphics was to use, the large variety of charts that can be created, symbols libraries, and other features such as the spell checker, slide show capability, and using macros to speed up your work. I thought it was a top notch program then, and the newest version, Harvard Graphics 2.3, hasn't changed my opinion. While many upgrades to a basic program are relatively minor in nature, that can't be said about this upgrade.

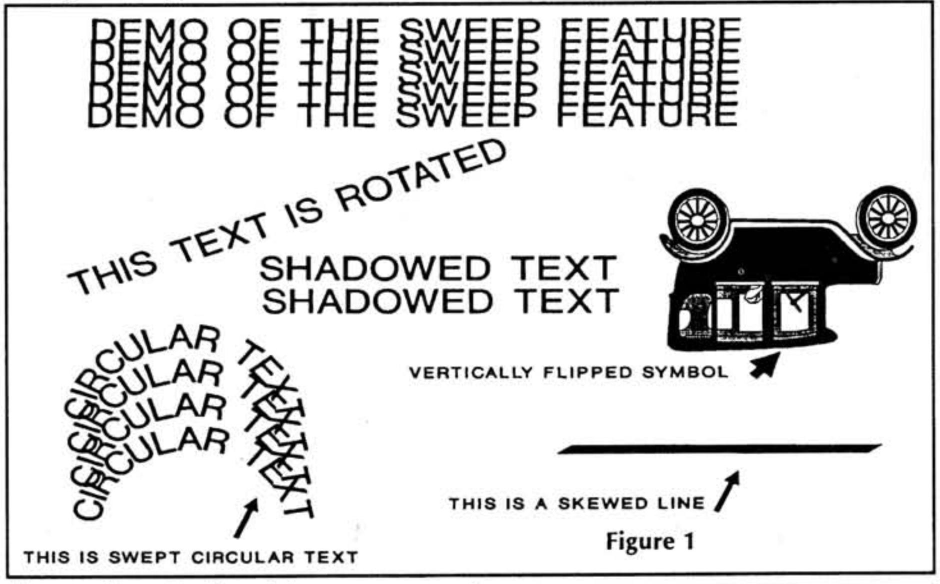
This upgrade has taken some of the accessory packages that Software Publishing Corporation (SPC) sold separately and combined them into one program without changing the feel and operation of previous versions. For instance, version 2.3 contains the Draw Partner and Chart Gallery accessories. Even with additional features version 2.3 only uses 420K of RAM, unless you want to use the on-line tutorial or macro program. You will need 640K to run these programs.

I'll begin by talking about the major improvement - Draw Partner, and then cover time saving features, improved printing capability, new symbols, slide show presentation improvements, and other miscellaneous features.

Draw Partner

Draw Partner is a versatile drawing program that lets you create more complicated charts, as well as logos for businesses or clubs, churches, etc. It allows you to add special effects to your charts. Figure 1 illustrates just some of the features you can use to create interesting charts. Figure 2 is an example of a logo I created using Draw Partner. In addition, you can fine tune drawings by using the zoom feature and point editing features. You can also use different fonts on a chart as well as perform the same features found in Draw/Annotate such as drawing lines, circles, and boxes as well as adding buttons. The button feature is discussed in more detail later in the article.

The most welcome improvement is that you can access Draw Partner from Harvard Graphics as well as return to it without leaving Harvard Graphics. Previously you had to access it from the DOS Prompt. You can now access it by using a speed key or through the Application Menu which is


DEMO OF THE SWEEP FEATURE
DEMO OF THE SWEEP FEATURE
DEMO OF THE SWEEP FEATURE
DEMO OF THE SWEEP FEATURE
THIS TEXT IS ROTATED
SHADOWED TEXT
SHADOWED TEXT
CIRCULAR TEXT
VERTICALLY FLIPPED SYMBOL
THIS IS A SKEWED LINE
THIS IS SWEPT CIRCULAR TEXT
Figure 1

discussed later in this article.

Chart Gallery. Anyone who presents briefings knows there often isn't time to prepare charts. Keeping this in mind, SPC integrated the Chart Gallery feature into Harvard Graphics 2.3. Chart Gallery contains 66 different pre-created charts in nine different categories. The briefer has a choice of text, pie, horizontal and vertical bars, and line charts, six area and high/low/close charts, five organizational charts, and four combination charts. This feature is activated from the main menu by selecting Create Chart and From Gallery. The briefer then selects the type of chart and the choices for that type chart appear. The briefer then enters the data and saves the chart. There is no need to spend time changing colors, sizes, etc.

Application Menu. The Application Menu is a simple DOS shell for Harvard Graphics. It only uses about 15K of RAM. Like Chart Gallery, it saves you some time. It allows you to run up to eight application programs, run Draw Partner or other Harvard Graphics Accessories, or work from the DOS prompt. However, you can't run

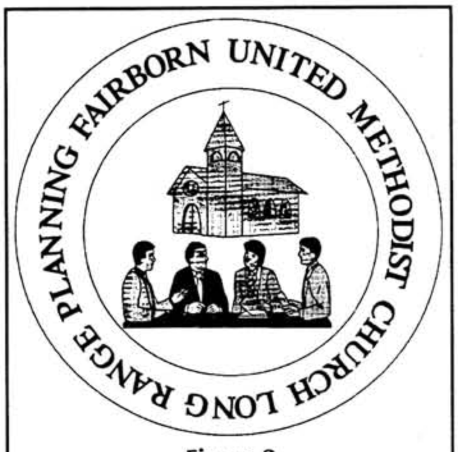
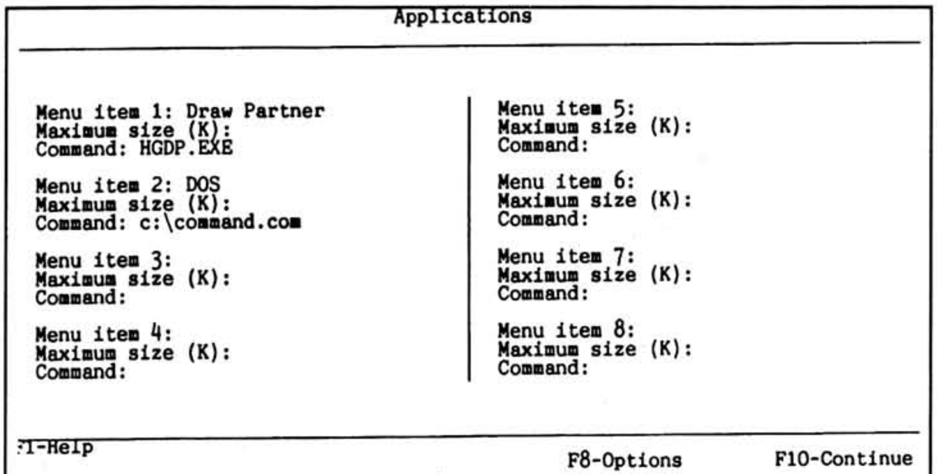


Figure 2



```
                               Applications

  Menu item 1: Draw Partner                Menu item 5:
  Maximum size (K):                        Maximum size (K):
  Command: HGDP.EXE                        Command:

  Menu item 2: DOS                         Menu item 6:
  Maximum size (K):                        Maximum size (K):
  Command: c:\command.com                  Command:

  Menu item 3:                             Menu item 7:
  Maximum size (K):                        Maximum size (K):
  Command:                                 Command:

  Menu item 4:                             Menu item 8:
  Maximum size (K):                        Maximum size (K):
  Command:                                 Command:

F1-Help                                 F8-Options       F10-Continue
```

Figure 3

the Harvard Graphics Macro program or the Harvard Graphics tutor or any other memory resident utilities. If you want to run the macro program or any other memory resident utilities you must load them before you start Harvard Graphics.

Memory limitation is often a big draw-back in trying to run two programs at a time. Harvard Graphics allows you to take advantage of EMS or a ram disk when you set the Application Menu options. A word of caution - before setting Harvard Graphics to use EMS or a ram disk ensure you have at least 600K of EMS or 600K of disk space available on your RAM disk. If you set the options to use either of these and don't meet the requirements you will not be able to run any additional programs. You'll just get an error message telling you there isn't sufficient disk space available. I know this because I made the mistake of setting Harvard Graphics to use my 384K ram disk to store temporary files.

Another limitation to this feature is, unless the program you want to run is on the DOS path, you must create a batch file with the commands necessary to run the program. The batch file name is typed on the command line (see Figure 3) at the Applications menu. SPC should improve this feature in future versions by allowing multiple command lines for each program.

Speed Keys. Another handy time saver is Speed Keys. You can save keystrokes to perform common operations. There are seven speed keys available at the main menu or at the chart data forms:

Ctrl G - Get a chart
Ctrl X - Import Excel Data
Ctrl S - Save the chart
Ctrl L - Import Lotus Data
Ctrl P - Print a chart
Ctrl R Go to Draw/Annotate
Ctrl D - Go to Draw Partner

This feature mostly benefits the inexperienced user. Experienced users are aware they can create one-key macros using the Macro program. In my opinion, SPC should expand on this feature in future releases as there are still many routine tasks not assigned a speed key.

Showcopy Utility. This utility is performed from the DOS prompt. It's now easier to copy slide shows, check a directory for all the files needed to copy a file show, and create a slide show from all the charts in a directory. It's no longer necessary to copy a slide show file by file. There are three commands:

Showcopy - Used to copy a slide show, templates (.TPL files), palettes (.PAL files), and bit-mapped graphics files (.PCX files) to another directory or disk.

Showcopy Verify - Used to verify that all the files, palettes, and bit-mapped graphics files are present on the disk before a copy is made.

Showcopy Create - Used to create a slide show from a particular disk or directory. Unlike the previous commands, Showcopy Create only uses the chart (.CHT) files to create a show. Charts will appear in the slide show according to where they are located on your disk. Harvard Graphics will assign the name NEWSHOW.SHW to the show you create unless you specify otherwise.

Better Computer Graphics Metafiles (CGM) Support.

Previous versions of Harvard Graphics, as well as version 2.3, import CGM images from other programs to create charts. The difference between version 2.3 and version 2.12 is how the imports are handled.

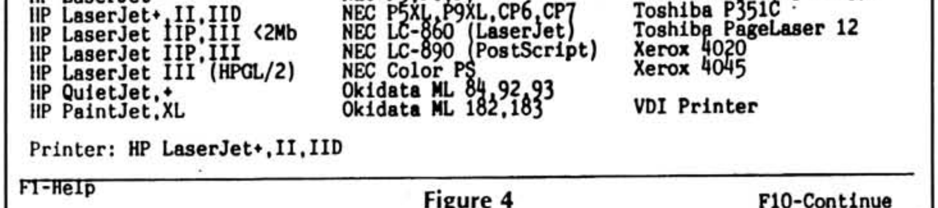
In version 2.12 you used a separate utility call META2HG outside the Harvard Graphics program. You typed META2HG [File name of CGM file] and the CGM file would automatically be saved as a symbol file.

In version 2.3, at the Import/Export Menu, you select Import CGM File, select the CGM file from the Select CGM Metafile screen by highlighting the file with the cursor and depressing F10. The imported file is then displayed on the screen. You can modify it using the Draw/Annotate feature or save it as a chart or symbol.

Improved Printer Output

More and Better Print Drivers. Version 2.3 has eight new print drivers. There are now more and better drivers for Hewlett-Packard Laserjet printers. Data going to the HP printers are now optimized so full size charts can be printed in high quality if the printer has at least 512K of memory. When I previously tried to print high quality charts using a HP Laserjet IID, my chart would not print on one 8 1/2" x 11" page. It printed half of the chart on one page and half on another. Now it prints on one page and it seems to be faster. See Figure 4 for a complete list of supported printers.

Gray Scaling Support for Black and White Printers. To print a pie chart or bar chart in gray scales (16 shades), all the user has to do is select Color as the fill attribute and set the Color to Yes at the Print Chart Options. Printing on an Epson FX-1050, I found this feature to be nice when I printed a chart on standard quality because it was easy to distinguish the different slices of a pie. When I printed the same chart on high quality it was much more difficult to tell the slices apart. However, when I printed the same chart on a Hewlett-Packard Laserjet IID I liked the high quality better. If you don't like the quality of gray scale charts you should select No at the Print Chart



```
                              Printer 1 Setup

 Apple LaserWriter            HP PaintJet,XL (Trans.)      Okidata ML 192,193
 AST TurboLaser               HP DeskJet,+                 Okidata ML 292,293
 Calcomp ColorMaster          HP ThinkJet                  Okidata ML 294
 Canon LBP8 II,III            IBM Graphics Printer         Okidata LaserLine 6
 Epson FX,LX,RX               IBM Proprinter,XL,II         Olivetti PG 108,208 M2
 Epson EX,JX                  IBM Proprinter X24,XL24      QMS ColorScript 100
 Epson MX                     IBM Quietwriter II,III       Qume LaserTEN,+
 Epson LQ 800,1000            IBM Color Printer            Tektronix 4696
 Epson LQ 1500                IBM Color Jetprinter         Tektronix Phaser CP
 Epson LQ 2500                IBM Personal PagePrinter     Tektronix ColorQuick
 Epson GQ 3500                IBM LaserPrinter             Toshiba P1340,P1350,P1351
 HP LaserJet                  NEC P5,P6,P7                 Toshiba P321,P341,P351
 HP LaserJet+,II,IID          NEC P5XL,P9XL,CP6,CP7        Toshiba P351C
 HP LaserJet IIP,III <2Mb     NEC LC-860 (LaserJet)        Toshiba PageLaser 12
 HP LaserJet IIP,III          NEC LC-890 (PostScript)      Xerox 4020
 HP LaserJet III (HPGL/2)     NEC Color PS                 Xerox 4045
 HP QuietJet,+                Okidata ML 84,92,93
 HP PaintJet,XL               Okidata ML 182,183           VDI Printer

 Printer: HP LaserJet+,II,IID

F1-Help                                                    F10-Continue
```

Figure 4

Options and you will get patterns.

Other Printing Improvements. There is a new no-margin option for non-postscript printers. It allows you to get the largest possible output provided you set the chart size to Full when printing. I liked this feature because my images were somewhat larger and clearer. In addition, you can now use A4 paper and to print files to disk.

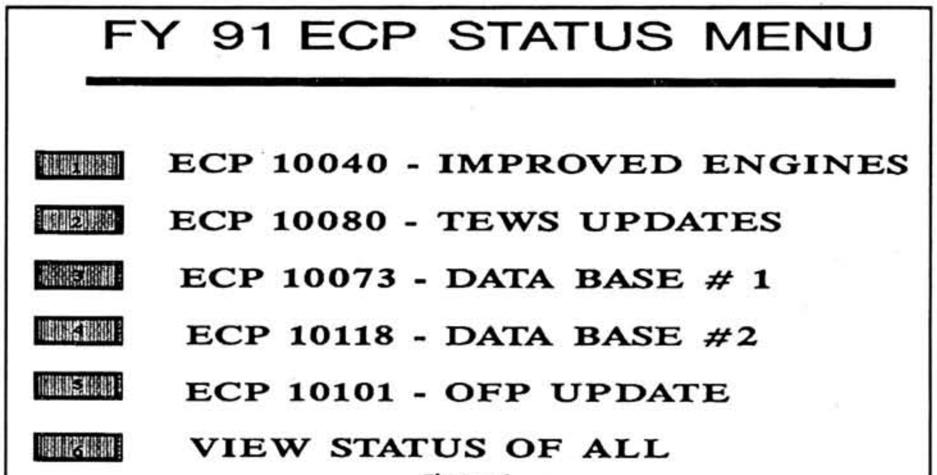
Larger Symbol Library

The new symbol library has over 500 new symbols. This large library corrects one of the deficiencies of previous versions. The symbols are much more detailed than previous symbols and also print in gray scales for users with black and white printers. The library contains: animals and plants, arrows, borders, buildings, buttons (more on this later), calendars, various common objects, computers and equipment, flags of 48 nations, flow chart symbols, Greek alphabet, people, industry symbols and icons, maps, money, signs, transportation, office equipment and items, and presentation symbols. Some of the new symbols appear on Figures 1 and 5.

Slide Show Presentation Improvements

Animated Sequences. Harvard Graphics contains 10 animated slide show sequences to spice up your presentation and grab your audience's attention. They include: a fireworks show, sand going through an hourglass, mouse in a maze, shooting ducks, William Tell, fish eating fish, dartboard target, fish story, birdman, and a cash register ringing up sales. All these shows can be viewed by displaying the ANIMATOR slide show in the HG\SAMPLE subdirectory.

These slide shows are composed of .PCX files and charts created in Harvard Graphics. You can edit the text in the fireworks show or the dollar amount that the cash register rings up or create your own sequences by using an overlay technique which is described in the documentation. The overlay technique draws objects on the screen that appear to be moving.



FY 91 ECP STATUS MENU

1 ECP 10040 - IMPROVED ENGINES

2 ECP 10080 - TEWS UPDATES

3 ECP 10073 - DATA BASE # 1

4 ECP 10118 - DATA BASE #2

5 ECP 10101 - OFP UPDATE

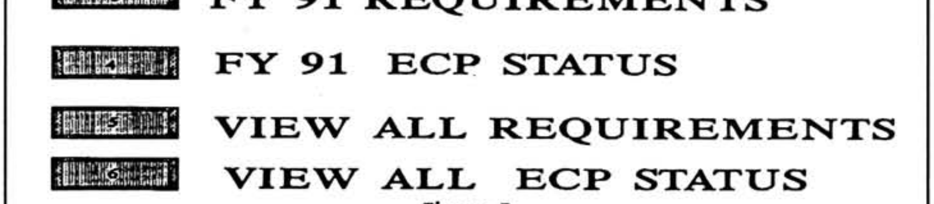
6 VIEW STATUS OF ALL

Figure 6

Hypershow/Buttons Features. This handy feature allows the briefer to show selected parts of a slide show. It replaces the Go-To feature in version 2.12. In that version, you could only go to one different chart from a chart you were viewing. However, in version 2.3 you can go to up to 20 different charts from the chart you are viewing.

Here's how it works. Figures 5 and 6 are master menus (free form text charts) I created to show the financial status of a program I work with. These charts allow me to go to other shows or charts I created previously without having to call up individual charts or separate screen shows from the main menu. The only thing different is the numbered buttons before the six menu items. They are button symbols from the symbols library.

Using the Draw/Annotate or Draw Partner feature I chose Add, selected Button, assigned a button number from 1-6, drew a box with the cursor around the symbols, and saved the chart. Figures 5 and Figure 6 are tied together through the Buttons feature. If I select button 4 (by depressing 4 or clicking on the button with the mouse) on Figure 5, Figure 6 will appear. From Figure 6 I can go to Figure 7 by depressing or clicking on the 1. After all the charts, slide shows, and menus are created they are put in one master slide show and each chart, slide show, or menu is assigned a number sequentially. They are linked through the Hypershow Menu which is activated by pressing F8 at the Screenshow Effects screen. Each button must be assigned a key and a chart/slide show number to go to. After keys and related chart/slide show numbers are assigned, the slide show is saved.



F15-E WST FINANCIAL STATUS

1 FY 89 REQUIREMENTS

2 FY 89 ECP STATUS

3 FY 91 REQUIREMENTS

4 FY 91 ECP STATUS

5 VIEW ALL REQUIREMENTS

6 VIEW ALL ECP STATUS

Figure 5

You can also assign special key numbers that allow you to go to the next chart, previous chart, the first or last chart in the slide show, or escape the slide show. Also, it's really not necessary to use a button symbol to link charts or slide shows. You can place the buttons anywhere on the chart. However, you have to remember where you placed it.

Miscellaneous Features

New Traditional Font and Multiple Fonts on One Chart. A Traditional font was added in version 2.3 for a total of seven fonts. In addition, version 2.12 only allowed one font per chart. In version 2.3 you can select the font you want to use from the Draw/Annotate feature or from Draw Partner. This is illustrated in Figures 5 and 6 where the chart titles are in Executive font and the remainder of the charts are in Traditional Font.

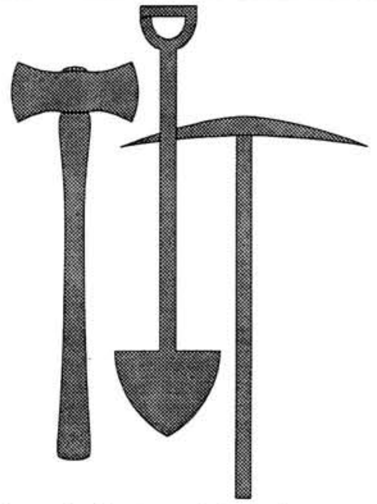
On-Line Tutorial. In addition to getting help by pressing the F1 key you can also get help from the tutor. At the DOS prompt you enter HGTUTOR and a menu appears. The tutor covers the basics of creating the various types of charts. There is also a brief section on slide shows where the Hypershow/Buttons feature is covered. If you are an experienced user I recommend not installing the tutor during installation as the benefit you would receive from it isn't worth the hard disk space it takes up.

Multiple Palettes. There are 11 new preset color palettes. What's new about this is the palette color is saved with the chart. This means each slide could have a differ-

Continued on Page 12

The Programmer's Craft:

Tools Of The Trade



Thomas B. Bing
2755 Carolyn Dr.
Smyrna, GA. 30080

Computer programs liberate people from tedious, mindless tasks. It's an irony, though, that programming itself is often a cottage industry – artisans hand-crafting each program with their own unique methods, using the most basic of tools. People created languages like C and Fortran using an assembler, an editor, and little else. What's worse, the editor was the "line-oriented" variety; using it was like reading your mail through a narrow slot. Debugging meant poring over "core dumps" – files containing a snapshot of the memory image of the program when it went "boom" – to find obscure glitches. That's about as far back as my personal career goes; I have heard stories of programming computers using front panel switches, flipping eight of these to load in one byte, but I'll have to take that on faith. That's an incredible amount of effort, compared to the work the machines were supposed to do.

Fortunately, all that is changing. Not fast enough, maybe – some of you may still use core dumps for debugging. Still, today's tools are miles ahead of what I had when I began programming PC's in 1980. Back then, I was amazed at the "full-screen" editor available on the Heath H-89; it was so much more intuitive and easier to use than mainframe editors. The only other software "tool" I had was the language itself - in my case, Microsoft BASIC for HDOS. You could run and modify a BASIC program without ever leaving the interpreter. This feature led to some quick fixes to code; the interpreter's ability to pinpoint errors on a line was a major boon to programmers. It also led to some bad coding habits; sometimes a programmer needs to back off and rethink the whole program, not just the screenful closest to the first error.

Overview

This article will survey some of the modern programming tools available on PC's, with a few observations on the future of programming. Most of the proper names in this article refer to specific software products or companies, and are therefore the registered trademarks of their respective companies. I try to distinguish between what I've used and products I've simply seen advertised. The PC software field involves rapid change in the companies and products being sold. Consequently, a product I mention is probably not the only tool for a given job, and may not be the best. Specific products are cited merely to give readers a starting point for their own investigation.

Compilers Vs. Interpreters

The first true compilers for PC's gave us a taste of the execution speed and memory savings available through this technique. No longer would we have to give up RAM (32K to 82K) to the interpreter. There was a commercial advantage too; we wouldn't have to distribute our code in source form. There was a trade-off, however; compiling a large program could take several minutes, especially on the early hard-disk 8088 machines. Back then, I didn't have Desqview or Windows, so I couldn't work on another program during a compile; I just waited until the compile succeeded – or failed. I spent ages refining my programs through the "3-C" method: Change, Compile, and Crash. I had to edit the program, save it and exit from the editor, and run the compiler; three separate programs were involved, counting the linker. Still, the pluses of compiling outweighed the minuses. For one thing, I could use pre- written libraries, linked with my own programs, for file management or other routine tasks.

The editor and the language itself were thus my only early tools. After I started using MS-DOS, some of the DOS utilities (and some shareware programs) came in handy for making backups and keeping track of files.

Current Tools: Editors

What's available today? How much of an advantage does a modern programmer have in tools and techniques over the pioneers? Plenty. Let's start with editors. Editors today can bring up two or more files in different windows; you can cut and paste text between them. Many of them can "undo" multiple changes. Color highlights text blocks being copied or moved. Global search and replace can change all occurrences of one name to another - or change only the ones you specify, showing each change as it occurs. Some high-end editors, like Sage Professional, can emulate a broad range of older editors while introducing new and powerful features. The BRIEF editor understands C-language syntax and keeps you from making basic syntax errors. The ability to drop down into DOS and return is a valuable and increasingly common feature. If you really can't live without your mainframe editor, KEDIT and SPF/PC are available for the PC. The ability to move program text easily between the editor, compiler, and other tools, such as version control systems, is rapidly becoming a reality. Granted, the high-end programmer's editors have a price to match, comparable with Wordperfect or Microsoft Word. There are a few good editors available at a modest price. I find that QEdit (SemWare, $59.95) offers multiple windows and a long list of useful features at a modest price. Plus, it's available on many BBS's (including the COM1 BBS; see QEDIT207.ZIP) for a "try before you buy". EMACS is a highly regarded windowing editor from the Unix world, written by Richard Stallman of the Free Software Foundation. There is a PC EMACS version available for $8.00 plus shipping (C Users Group Disk 197). There is also a Z-100 version available on the COM1 BBS; download ZMACSEXE.ZIP.

"Vi" is the editor I use most. If you've read the faint praise for vi in my "DOS And Unix, Part 2" article (REMark, May '90), that may surprise you. Admittedly, vi knows nothing about color. But it's flat-out fast at doing block copies and deletes, and it's available on both MS-DOS and Unix machines. It can be customized a good deal, even to the extent of defining function-key macros. Vi is available by itself for $149 from MKS, but don't buy it that way. It's a much better deal as part of the MKS Toolkit.

My recommendation for a PC editor would be QEdit, vi, or EMACS, at least to start with. However, many folks will be quite at home with the basic Wordstar-style editors that come with compilers like Borland Turbo C and Microsoft Quick C. These integrated editors speed up the cycle of edits and compiles. They also keep you from shuttling back and forth between separate editor and compiler programs.

Version Control

If you only deal with a few short programs that don't change often, keeping track of different versions is easy. However, as time goes by on most programming projects, programs grow and multiply, revisions come more often, and more people get involved in the updates. As a result, an automated system for tracking changes becomes crucial. I use one called RCS (Revision Control System), from MKS. RCS maintains a file for each source code program I develop. This file contains the latest version of each program, its revision history, and "deltas" for each revision back to the earliest. By applying these deltas to the latest version, RCS can retrieve any version of the file I wish to see. Special keywords are used in the RCS source file to identify the version number, author, and revision date and time. In this way, RCS can place correct version numbers and timestamps in each version without the programmer having to change them manually. Two special programs are used to "check in" and "check out" versions. If a version is checked out and locked by one programmer, no one else can change this program (within RCS) until it is unlocked again. The unlock occurs when the original programmer checks it back in or a special unlocking program is run. This diminishes the danger of two conflicting revisions to one program. The program is still accessible for "read-only" use, such as printing, even when locked. RCS can be told to discard the latest version of a program and its revision notes automatically. This effectively "backs out" changes which turn out to be undesirable and makes the previous version the most current one. Any version on file can be deleted in this manner. You can even insert version numbers in .OBJ and .EXE files by simply defining a character array or string containing the version number. Such arrays usually contain a distinctive character sequence that is unlikely to occur elsewhere in the file; this allows the file to be quickly scanned for the version number. This is useful for a programmer troubleshooting on a user's machine; the version number of the software which is actually in use can be verified.

An added benefit of RCS is that only the latest version appears in full-text form; earlier versions are reconstructed as needed from the latest version and the deltas. This technique avoids needless duplication and saves disk space.

These features of RCS are representative of those available in other version control packages. Sage/Polytron offers Polytron Version Control System (PVCS) for PC's. PVCS has been endorsed by Marc Rochkind, the author of SCCS, the original Source Code Control System for computers running Unix. Other PC version control products include TLIB and SMS. These systems can manage other documents as easily as program files. The RCS product for the PC can even control binary file versions, so you could use it for WordPerfect or Wordstar documents.

Selective Compiling: Make

In addition to controlling versions, it's also useful to control the compile process for large programming projects, so that only those source files which have changed are re-compiled. Programs that perform this kind of compilation control are called "make" programs; this was the name of the original Unix version of this utility.

The hot competition between Borland and Microsoft has resulted in fairly capable versions of "make" being included with these two C compilers. Even on today's fast 286 and 386 machines, utilities like these are worth having to eliminate unnecessary compiles (and of course, to save the programmer's time). Opus and MKS also market their own versions of make.

C Debugging Tools

Some C tools that have long been available on the PC are making their way to Unix systems (and vice versa). Users of PC C compilers with integrated editors and debuggers may be surprised to hear that these are fairly new to Unix. In Unix systems with X Windows, we are beginning to see this kind of integration: Hewlett-Packard's Softbench is a fairly sophisticated example. And it's true that Unix programmers can speed up the 3-C cycle by keeping their editor running while they started a new copy of the shell to handle compiles; on systems with lots of memory, you can do that. However, it is only by integrating the compiler and editor that the editor can highlight the exact position of errors in the source file as the compiler finds them.

Gimpel Software's C-terp is an interactive, interpretive style debugging environment for C. It's available for both MS-DOS and versions of Unix that run on Intel CPU's. Gimpel also offers PC- Lint, a source code analysis tool with its origins in the Unix world. Lint was designed to resolve inconsistencies in source programs that the compiler might not spot, such as a function call with two arguments when three are required. I see a tendency for the high-end PC C compilers to include "lint-like" features. For instance, the Microsoft C compiler version I use (5.1) has a /W3 flag that turns on very extensive error checking and spots some of the same errors that "lint" would find.

Cross-referencing to show how variables and functions are used in several program files is a must when developing big projects. It keeps a change in one file from having unintended consequences elsewhere. Programmers who have used "cxref" on Unix can probably find similar capabilities in "C-Analyst" by Cater Software and "CLEAR for C" by CLEAR software. CLEAR also advertises documenting and flowcharting features.

Function Libraries

Often, programmers like to concentrate on the bigger issues in designing a program. "Standard" tasks like file management and screen layout are handled by purchasing function libraries that link with the object modules the programmer creates. When I was developing an inventory management/floor plan package years ago, I chose to move from MBASIC to compiled CBASIC. One factor in my decision was that the Access Manager package from Digital Research would work with CBASIC. Access Manager provided a speedy way to access and update records in large data bases and left me free to design the user interface and error handling. Examples of currently available libraries for C are: c-tree (file management), Blaise C tools (screens and windows), WKS Library (.WKS, .WK1, and .DBF file manipulation), and HALO (graphics). They're usually in the $100 - $500 price range. Often that's worth it, if the package works as advertised and keeps a major software project from going down the tubes.

Database Tools

I've dwelt mostly on C, but the tools for database programmers, particularly in the dBase arena, are worth mentioning. The dBase compilers which turn dBase into .EXE files have attracted attention because they're speedier than interpreted dBase and they don't require a dBase license for each user's machine. I bought Wordtech's Quicksilver a few years back and I remember when a friend brought over a medical practice package he had written in dBase to compile it. It took a little tweaking to get it to compile and link okay, but it ran so much faster he just sat there and giggled.

Because dBase is such a widely used data base manager, several tools are available to aid programmers in using it. There are the dBase compilers already mentioned, such as Quicksilver, Clipper, and Force. There are program generators such as Genifer and UI2 which build complete source code, after the programmer specifies the screen and data file layouts. Having used Genifer, I can say that it takes a little time to build your screens and files correctly, but still far less than with manual methods. Also, once those specs are complete, Genifer will generate hundreds of lines of dBase source code in minutes. The code produced this way can be modified like any other .PRG file. If the initial run points out the need for changes in file and screen layout, they can be easily made and the program can be quickly re-generated from the revised specs.

There are other packages, such as dBEST Toolbox and Mach-4, that add functions like graphics and date conversion to the standard dBase capabilities, and dSALVAGE, to help you recover damaged data files. As mentioned before, there are C libraries to read and write .DBF files. Since other database programs are being marketed as a step up from dBase, many of them have utilities built-in to read data from .DBF files and transfer it into their own data base format.

On The Horizon

One area I've wanted to explore (and haven't yet) is the use of program generators to produce C programs. Tools like Matrix Layout and Charm are advertised as automating the generation of C source code. It'll be interesting to see how much work programmers still have to do with such tools.

I see two trends that will ultimately change the way software is developed for PC's. The first is the emergence of usable CASE tools. CASE stands for Computer-Aided Software Engineering, and, in my opinion, we're not there yet. American companies are aiming for a seamless set of tools that will manage the creating and maintaining of programs over their full life cycle - from the initial bright idea through development and bug fixes, until the program is retired. There are a few products now that help with documenting program specs and new designs, and with figuring out what each program is supposed to do. There are also tools to maintain consistency among the various pieces of large programming projects, to keep the work of a large team in sync. My own view is that such tools must be affordable (about the same price as a PC C compiler), and adapted to the needs of a small programming group, one to five people, before they will make much impact in the PC arena. I also think they should be able to clean up and reverse- engineer existing code to improve its correctness and documentation. The ability to rework existing code is something that is only beginning to appear in the expensive mainframe CASE packages.

The second trend is the emergence of object-oriented languages (Smalltalk and C++, for instance), and use of object-oriented programming (OOP) in traditional languages. The whole idea of "objects" that handle specific tasks by operating on messages passed by other objects appears to be gaining ground. Programmers will develop libraries of objects that can be extended and reused, and these objects will become the interchangeable parts of software manufacture. I believe that C++ will be as commonplace in a few years as vanilla C is now. Whether programmers will have absorbed the OOP philosophy is another matter.

Having all these goodies will not be the millennium or the death knell of programming. Their use should mean that programmers will worry about bigger "chunks" of the programming job; that their creativity will focus at a higher level. I think future programmers will design software more at the "black box" level of whole programs and major functions. To complement the human designer, automated systems will look after lines of code, variable names, and similar low-level issues. For instance, adding a new data field to a whole family of programs and files may require only a single English-like command from the designer. We're not there yet, but from my vantage point, we sure have made some progress.

Sources

To conclude this article, let's look at sources of the products mentioned. Two mail-order houses that cater to the software needs of programmers are:

Programmer's Connection
North Canton, OH 1-800-336-1166
The Programmer's Shop
Hingham, MA 1-800-421-8006

Both these companies publish large, detailed catalogs of their offerings. The catalogs themselves are a basic education in programmer tools; they include capsule descriptions of the products. Just about everything described above is available from one of these two companies. The C Users Group disk containing EMACS is available from the C Users Group, 2601 Iowa Street, Lawrence, KS 66046, phone 913-841-1631. The QEdit editor is also available from SemWare, 4343 Shallowford Road, Suite C-3, Marietta, GA 30062, phone 404-641-9002.

Continued from Page 9

ent background or it could remain the same through the entire slide show presentation.

Conclusion

Version 2.3 is a significant improvement over version 2.12. It is well worth the price of the upgrade unless you previously purchased the Chart Gallery and Draw Partner accessories separately. The strong points of the upgrade are it's improved symbols, improved printing quality, and slide show features.

Don't miss a single issue...

make sure you send in a change of address!

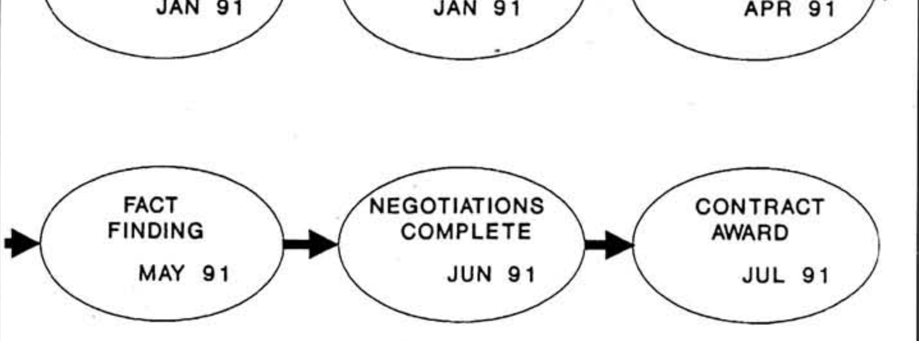

F-15E WST: FY91 3010 FUNDING--$4.2M
ECP 10040: IMPROVED PERFORMANCE ENGINE
ECP RECEIVED JAN 91
FFP COST PROPOSAL JAN 91
TECHNICAL EVAL/AUDIT APR 91
FACT FINDING MAY 91
NEGOTIATIONS COMPLETE JUN 91
CONTRACT AWARD JUL 91

Figure 7

Breaking the Churlish Bounds of DOS

Part 2

David W. Lind
RR 1, Box 3114
Bar Harbor, ME 04609

In the first of these series of articles, the concepts of overlays, expanded memory, and extended memory were introduced. This article completes the discussion of overlays. Parent and child processes were discussed in Part 1. These processes are not overlays in the narrowest sense of the word in that the child programs occupy unused memory released by the parent program for that purpose. The parent program is never overlayed. However, memory previously allocated to the parent program is overlayed. In this article, the processes discussed do overlay the code of the parent program which is called the "root" program. Two different processes are discussed, the use of an overlay linker and the use of the disk operating system Execute Program Function.

The Overlay linker

The easiest way to create overlays is to use the overlay feature of the linker. When a source program is assembled or compiled, a file called an object file is produced. The object file contains some executable code and references to external subroutines which must be incorporated before the code can be made executable. The linker is the program which resolves the external references and arranges the code into an executable program file with the file name extension, EXE. The external references may be other object code files in libraries provided with the compiler, in user generated libraries, or object code files provided to the linker with the main or root program. Assemblers do not normally come with libraries because assembly language instructions are normally translated directly into machine code. Most linkers permit the selection of one or more object files as overlays. The assembler or compilers used to generate the root object module must have an overlay management capability.

The current (Version 5.0) Microsoft Macro Assembler (MASM) does not have an overlay manager. Therefore, one cannot use MASM to generate programs which use overlays created by an overlay linker. However, most newer Microsoft compilers do have overlay managers which will pass appropriate code to the linker to generate overlays. MASM can be used to create object files which are used as overlays by the overlay linker. In general, the root program must be compiled or assembled by a compiler or assembler which has an overlay manager. The overlays can be valid object files created by any means.

The Microsoft Disk Operating System (MS-DOS) LINK.EXE linker program allows overlays to be specified by placing the overlay object files in parentheses when the LINK program is executed. The linker's overlay manager will then determine when and how the overlays will be used. All overlay modules must be code segments or procedures (subroutines). Data is not overlayed. The manuals provided with the linker describe the process in detail. If the applications programmer follows the instructions carefully, the overlay process should work properly without any user intervention when the program is executed. The size of the program code is limited only by the amount of mass storage memory available for the program file. Only one file with the extension EXE is generated. This file contains all the overlays, as well as the root program, and may be much bigger than available random access memory. The file will be reopened as necessary to overlay memory.

If the reader is familiar with the FORTRAN programming language, the following short FORTRAN program can be used to call the procedure "over3" listed as "OVERLAY3.OVL" at the end of this article.

```
EXTERNAL OVER3
CALL OVER3
END
```

This code, to which the file name "OVTEST.FOR" was assigned, should be complied by a FORTRAN compiler which has an overlay manager. The statement "PUBLIC over3" must be added to the procedure source code just before the statement "code SEGMENT." Then the procedure, which was assigned the file name "OVER3.ASM," should be assembled. The FORTRAN object code (OVTEST.OBJ) and assembler object code (OVER3.OBJ) should then be linked with the statement:

```
LINK OVTEST + (OVER3)
```

The reader should note the size of the .EXE file, which for the author's file was 7,432 bytes, and re-link the files without the parenthesis. The resulting file should be much shorter. The author's re-linked file was 4,108 bytes. Thus, there is additional code overhead associated with overlays as would be expected. This overhead must be considered in the design of overlay programs.

Of course, an overlay linker cannot be used to solve the problem which occurs when there is too much data for the program memory space. In that case, the data must be kept in files on mass storage and called into memory as required. Another way to resolve the excessive data problem is to store the data in expanded or extended memory. More about that in future articles. But, even the excess data problem can be handled by the DOS Execute Program function. However, this usage is normally not recommended. In addition, this function permits management of overlays without using the overlay features of an overlay linker.

The programs used as examples in the following discussion are written in assembly language. Many compilers, e.g., C, have features which allow execution of system interrupts. The reader should be able to write programs in these higher level languages based upon the examples. However, there are also library subroutines in these languages which will accomplish the same objectives with less intervention by the programmer. The examples should provide considerable insight into the operation of these library subroutines.

The Execute Program Function

The Execute Program Function was introduced in the first of these series of articles. The Load and Execute Program subfunction, Subfunction 00h, was discussed and demonstrated. Subfunction 03h, Load But Don't Execute, is the subject of this article. Load But Don't Execute overlays a part of the parent or root program with code or data from a file on a mass storage medium. But, the code is not executed. Thus, Subfunction 03h differs from Subfunction 00h in three ways. First, Subfunction 03h does not execute the loaded code or data, but Subfunction 00h does. This difference also means that the overlays for Subfunction 00h must be executable code, but the overlays for Subfunction 03h can be code or data or a combination of code and data. Finally, Subfunction 00h overlays free memory space which was released by the parent program. This space is not in the parent program. However, the memory used by the overlays handled by Subfunction 03h is normally within the parent program.

The Load But Don't Execute subfunction is useful when one wishes to exercise more control over the overlay process or have more direct access to data segments. The overlay becomes a part of the root program, not simply another program in memory. However, Subfunction 03h will overlay any part of random access memory (RAM), including the disk operating system and RAM portions of the Basic Input/Output System. Therefore, the Load But Don't Execute subfunction can destroy important system data or code and even damage video circuits if used carelessly. The likelihood of damaging circuits is not great, but cannot be ignored. As a general rule, one should overlay only memory owned by the root program.

There are generally two types of code overlays, simple code segments and procedures. This article discusses both in detail. A third type of overlay is also discussed, a data overlay. Data overlays are uncommon because it is usually much easier and more efficient to use data files. Before these types of overlays can be used, some preparation is necessary. Keep in mind that the following discussion is a "text book" approach. Alternatives will be presented after the basic discussion.

Memory Allocation Preparation

Before the Load But Don't Execute subfunction is used, memory needs to be reallocated. Recall that the root or parent program is normally allocated all available memory when it is loaded. Although Subfunction 03h overlays do not required free memory, the loader program does. Therefore some memory must be freed for the loader. Generally, it is best to shrink the memory allocated to the program to the minimum required. This feat is accomplished with the Set Block function. The ROOTPGM.EXE listing at the end of this article demonstrates this process. The listing contains an unusual number of comments so that the reader is not forced to recall details from the text. It is best to learn the general concepts from the text and glean the details from the comments in the program. As is no doubt clear by now, this subject is not trivial.

After the program is shrunk, the programmer must determine where the overlays will be located. Usually it is best to establish a separate code segment where the overlays will reside rather than overlay existing program code. The program should be enlarged using the Allocate Memory function, INT 21h-Function 48h. This function adds some of the memory freed by the Set Block function to the root program. Typically, a 64K byte segment is added. The number of paragraphs of memory to be added to the program is placed in register bx. Recall that one paragraph of memory is 16 bytes. In ROOTPGM.EXE, the number of paragraphs is 4096 which is 64K bytes. The Allocate Memory function is called by placing 48h in register ah and executing INT 21h. The major reason for using the Allocate Memory function rather than simply releasing less memory with the Set Block function is that the Allocate Memory function returns the segment address of the allocated memory in register ax. If this additional segment were to be generated by the Set Block function, the programmer would be responsible for determining the address of the new segment. Once one has the address of the new segment, it can be stored for later use. But, keep in mind that the Load But Don't Execute subfunction will place the overlay anywhere in the basic system memory the programmer directs. If one uses a separate segment for the overlays, it is important to keep track of the address of that segment.

The Load But Don't Execute Subfunction

Once enough memory is released for the loader and a segment of program memory is selected/prepared for the overlay, the overlay can be called into memory. This process requires that a parameter block be established in the extra segment. This parameter block contains only two words. The first word is the segment address of the overlay, normally obtained from the Allocate Memory function. The second word is the relocation factor which is normally the same as the overlay segment address for EXE type overlays and is 0 for COM type overlays. Further discussion of relocation factors is beyond the scope of this article. When the subfunction is called, es:bx must point to the parameter block. DS:DX must point to an American Standard Code for Information Interchange (ASCII) string containing the path, file name and extension of the overlay file. This string must be terminated by a binary "0." The ax register must contain the value "4B03h." Then the overlay is called by executing INT 21h.

Simple Code Segment Overlays

For the purpose of this article, a simple code segment overlay is a segment of code which is written as an executable program or part of an executable program, but is not an explicit procedure or subroutine. Subfunction 03h does not examine the overlay to determine if it is data or code and certainly does not determine if it is a simple code segment overlay or a procedure. But, this information is important to the programmer.

Recall that the overlay is not executed by Subfunction 03h. If the overlay is code, it must be executed by a jump or call to the overlay. This transfer of control is to a separate code segment, assuming one has not overlayed the current code segment, and is a "far" jump or call. That is, both the code segment register and the instruction pointer register must be changed. A simple code segment overlay requires some special coding considerations, whereas, a procedure does not.

The problem is not so much transferring to the overlay as it is getting back to the proper point in the original code segment. Either a far jump or call will transfer control to the overlay. The overlay code segment must contain a means to return to the original code segment. If the overlay code segment is accessed by the "call" instruction, the segment:offset address of the next instruction in the original code segment is pushed onto the stack automatically. The code segment overlay can then return control to the original code segment by a far return. If the programmer wishes to use a jump instruction to execute the overlay, the code segment address and the address of the next instruction in the original code segment must be pushed onto the stack before the jump instruction is executed. The return from the overlay code segment is the same as with a call.

The OVERLAY1.OVL overlay file listed at the end of this article is an example of a simple code segment overlay. The execution of such an overlay can be accomplished by pushing the code segment register onto the stack with a "push cs" instruction. Then, the address of the instruction to be executed upon return from the overlay must be pushed onto the stack. This process is most easily accomplished by labeling the instruction to be executed upon return and loading the address of this label into a register with the "lea" instruction and pushing the register onto the stack. Control is then passed to the overlay code by a far jump. The instruction "jmp dword ptr ..." is used for this purpose. The "..." in the instruction is a label in the data segment at which the segment:offset address of the overlay code segment is found. The first word at this label is the offset address. This word is set to 0 for an EXE type code segment and 100h for a COM type code segment. The second word is the segment address, normally obtained from the Allocate Memory function. Note the use of the "dword ptr" modifiers in the jump instruction. The "dword" specifier tells the assembler to treat the operand as a data label at which the double word address is located. The "ptr" operator tells the assembler to override the original type of the label (which, in the example program, is word) and treat it as a double word label.

Some folks may want to use the instruction "jmp far ptr ..." This instruction will not work properly. The specifier "far" is used only with labels. The instruction "jmp far ptr ..." will cause the data at the label represented by "..." to be executed. The only time this instruction can be used is when the overlay code segment is labeled.

The alternate and simpler way to execute the overlay code segment is to use the "call" instruction. The format is "call dword ptr ..." This instruction automatically puts the code segment and return offset addresses on the stack. Again, "..." is a label at which the segment:offset address of the overlay execution point is found.

Returning from the overlay code segment can be simple or esoteric depending upon the version of the assembler in use. Starting with MASM Version 5.0, one may use the return far, "retf," instruction - a simple and straight forward method. It is not necessary to use "retf" with any other instruction or directive.

If one is using a version of MASM that does not support "retf," the method is a bit more complex. The offset address of the original code segment must be popped into a memory location and then the segment address of the original code segment must be popped into the next word. Then a far jump is used to return the code to the original code segment. Be sure to locate the data for the segment:offset address of the original code segment in such a way that the assembler can unambiguously locate the data and that the data will not be executed.

Procedures

Procedures are code segments explicitly written to be called by the root program. In some programming languages procedures are called subroutines. When one identifies a procedure or subroutine to the assembler or compiler, the assembler or compiler ensures that the requirements to return from a call are included in the object file. If the procedure is a part of the root program's original code segment, the procedure is typed as "near." If the procedure is in another code segment, it is typed as "far." The default type is near. Normally, an overlay procedure is explicitly typed as far. One can always type a procedure as far, but the procedure must then be called with a far call.

Once a procedure is overlayed on the root program, the procedure can be called with the instruction "call dword ptr ...". As with the call to a simple code segment, the "..." is the label of two words containing the offset and segment address of the procedure. Again, "dword ptr" must be used because the label is the location of the procedure address, not the procedure itself.

A procedure may also be called by a far jump (e.g., jmp dword ptr ...), but this process requires the programmer to place onto the stack the segment:offset address of the instruction to which to return. The use of the "call" instruction is much easier and less prone to error.

The procedure itself is normally typed as far (e.g., PROC FAR). It is not necessary to use retf if the procedure is defined as far. If one uses retf, no harm is done. Remember, the instruction "retf" was introduced in MASM Version 5.0, so it is not available in earlier versions.

Data Overlays

A data overlay is a difficult means of dealing with data problems. Calling data files through the file control block process is better. However, there may be reasons to overlay data. The data is overlayed on the root program in exactly the same manner as code segment overlays. Of course, the overlay segment address is a data segment address. Also, a data overlay is not executed. The overlay can be on the original data segemnt or in a separate data segment. The program listing at the end of this article shows the data segment overlayed. Keep in mind that the data in the overlay should be addressed by direct numerical means, not by labels, unless one uses a data segment template.

Freeing Memory

When one is finished with the section of the root program used exclusively by overlays, this memory can be freed by the Free Allocated Memory function-Function 49h. This step is not necessary, but it is a good practice particularly if the program is a terminate-but-stay-resident program. To use this function, first push the es register onto the stack because the function uses the es register to point to the segment of memory to be freed. Then, put the address of the segment of memory to be freed into a general register and move it to the es register. Move 49h to the ah register and call the Free Allocated Memory function with INT 21h. This process frees the allocated memory. Then pop the original value of the es register back to the es register.

Assembling and Linking Overlays

Before leaving the subject of overlays, a few words about assembling and linking overlays may be helpful. The assembler will process any valid data segments or code segments individually or in combination. Stack segments should not be included in overlays. Disregard warnings about no stack segments. The root program and overlays should be linked separately. Therefore, labels should not be defined as public and external references must not be included. The linked overlay files should be assigned extensions other than EXE or COM to prevent independent execution of these files. When the loader is asked to overlay existing programs, it ignores the extensions in the overlay file names.

Alternative Approaches

The techniques described above should work in virtually any situation. If the operating system environment is unencumbered and well-behaved, overlay short cuts are possible. The Set Block function may not be necessary. When the operating system releases the loader, the memory that the loader used is also released to the system - not to the executing program. Therefore, if the program does not allocate memory and there are no resident programs which may seize free memory, there will be enough free memory for the loader. That is a rather big "if" given the current glut of esoteric resident programs.

One can define a 64K byte segment in the original program to use as the overlay segment. For example:

```
overlayseg          SEGMENT
                    DD         16384 DUP(?)
overlayseg          END
```

would make a perfectly good overlay segment. The address of this segment would be obtained in ax by:

```
lea      ax,seg overlayseg
```

The difficulty with this approach is that if the program is too big, one has no opportunity to reduce the size of the segment without re-assembling the program. On the other hand, the Allocate Memory function will return the maximum amount of free memory available if it fails to allocate the requested amount of memory because of insufficient memory. At that point, the requested memory and size of the overlay segment can be reduced by the program.

It should be clear at this point that overlays can be versatile and powerful. However, they can be dangerous. One should be forewarned once again of these dangers.

Warnings and Cautions

Overlays can be placed any where in memory. Thus, one can destroy or alter the operating system or other programs and files. Ensure the overlay segment address is not external to the program unless you really wish to change memory external to the program. Avoid changes to segment addresses above E000h where video control memory resides. One can, in this memory area, reset video output to produce Virtual Graphics Array output which can damage video display equipment not designed to accept such signals. This case is one of the exceptions to the rule that one cannot damage a computer system by just programming it.

It is possible to overlay code segments with data and data segments with code. One can execute data segments and use code segments as data. Almost all things are possible, but not all things are profitable. The information in this article opens the door to all kinds of programming tricks. Be careful. Strange combinations of techniques may permit extraordinary capabilities and confuse even the most brilliant programmer, but these techniques may also confuse the originator and become a curse rather than a blessing.

The Demonstration Program and Summary

The program listing following the references demonstrates the techniques outlined in this article. The root program memory is shrunk to minimum size, then memory is allocated for the overlay segment. A simple code segment is overlayed on the overlay segment and executed using a far jump. New data is then overlayed on the data segment. A procedure is overlayed on the overlay segment and executed by a far call. Before the program terminates, the allocated memory is released to the system. These techniques should help the programmer develop powerful programs using overlays. The easiest way to use overlays is to allow the overlay linker to generate them automatically, a feature of the overlay linker which is much appreciated by folks who have occasion to use INT 21h-Function 4Bh.

The next article in this series will discuss the use of expanded memory.

References

1. Duncan, Ray, Extending DOS, 1990, Addison-Wesley Publishing Company, Inc., Reading Massachusetts.

2. Microsoft Corporation, Microsoft Macro Assembler 5.0 Microsoft CodeView and Utilities, 1987, Microsoft Corporation, Redmond, Washington.

3. Microsoft Corporation, Microsoft Macro Assembler 5.0 Programmer's Guide, 1987, Microsoft Corporation, Redmond, Washington.

4. Microsoft Corporation, Microsoft MS-DOS Operating System Programmers Reference, 1984, Microsoft Corporation, Bellevue, Washington.

5. Phoenix Technologies Ltd., System BIOS for IBM PC/XT/AT Computers and Compatibles, 1989, Addison-Weseley Publishing Company, Inc. Reading, Massachusetts.

6. The Waite Group, The Waite Group's MS-DOS Developer's Guide, second edition, 1989, Howard W. Sams & Company, Indianapolis, Indiana.

Trademarks

Microsoft and MS-DOS are trademarks of the Microsoft Corporation.

OK. give me the story one more time, you're reading a borrowed REMark?

RAPID INDEX: A computer program that lists titles of magazine articles by subject or in alphabetical order.
☐ CPM or ☐ PC/MS-DOS 2.1 or later.

COMPUTER:	DRIVE:
☐ Z-100 Models	☐ 5.25"DD..$21.00
☐ IBM Compatable	☐ 5.25"HD..$23.00
	☐ 3.5"DD.....$23.00

Magazines currently available:
Elect. Service & Tech. (1984 - 89)
REMark computer magazine (1985-89)
Calif. residents add 6.75% sales tax.
Send check or money order (US funds) only to:

B.U.D. Unlimited,
P.O. Box 503, Elverta, CA 95626.

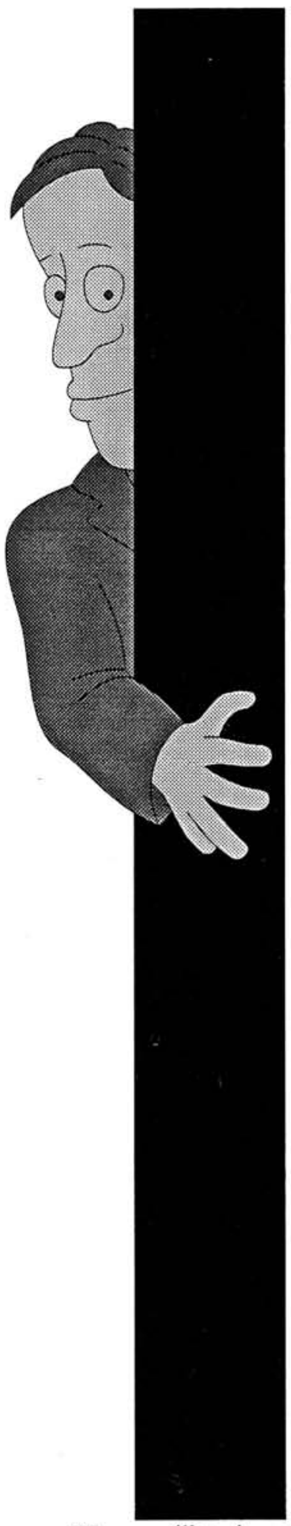
REMark is a trademark of Heath. Zenith User's Group.

Reader Service #250

Listing

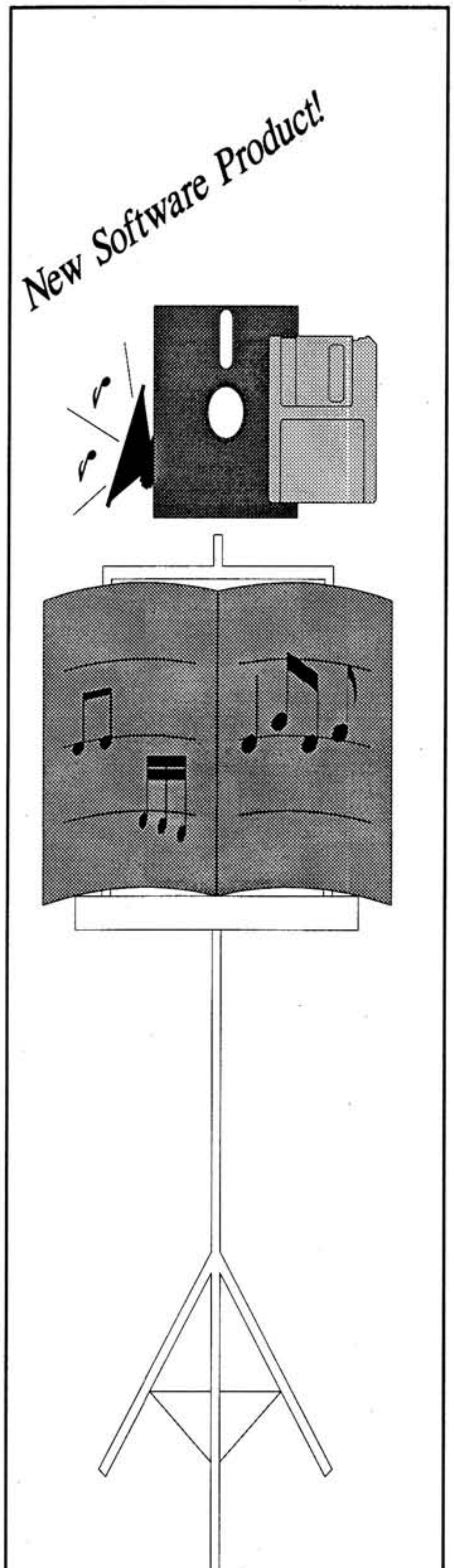
```
         COMMENT * PROGRAM: ROOTPGM.EXE

                   This program demonstrates the overlay process controlled by
                   the Execute Program Function, Subfunction 3.  Three over-
                   lays are called by this program. One overlay is another
                   code segment, the second is a data segment, the other is a
                   procedure.

                   This program and the overlays are for pedagogical purposes
                   only.  No expressed or implied warranties are made for this
                   program with respect to merchantability or fitness for a
                   particular purpose.  The user assumes all responsibility
                   for any damage resulting from the use of this program or the
                   concepts.  WARNING:  Overlaying some parts of memory could
                   cause damage to data or equipment.  Ensure that the segment
                   address of the overlay is within the program memory space
                   and that the overlay is able to fit within the allocated
                   memory. *
data     SEGMENT

         COMMENT * The following message is displayed by OVERLAY1.OVL. *

ovlay1   DB       ' This line is displayed by the new code segment.',10,13,'$'
         DB       100 DUP(0)           ;Allow room for overlays

         COMMENT * The following data are used by ROOTPGM. *

ovad     DW       0                    ;Far address of overlay, offset portion
overseg  DW       ?                    ;Far address of overlay, segment portion
name1    DB       'OVERLAY1.OVL',0     ;File name of first overlay
name2    DB       'OVERLAY2.OVL',0     ;File name of second overlay
name3    DB       'OVERLAY3.OVL',0     ;File name of third overlay

         COMMENT * The following data are messages produced by ROOTPGM. *

mess0    DB       ' ERROR - could not shrink program memory.',10,13,'$'
mess01   DB       ' ERROR - could not allocate memory for child process.',10,13
         DB       '$'
root1    DB       ' This line is displayed by the root program.',10,13,'$'
xitmess  DB       '****Normal Program Termination.','$'
data     ENDS

esdata   SEGMENT

         COMMENT * The following data is the parameter block for the Execute
                   Program function. *

segadd   DW  ?
relfact  DW  ?
```

```
esdata  ENDS

code    SEGMENT
        ASSUME   cs:code,ds:data,es:esdata,ss:stack  ;Identify segments
start:
        COMMENT * The following section of code initializes the data segment
                  and shrinks the program's memory space to that which it
                  actually requires.  If the program space cannot be reduced,
                  the program terminates.  This process is required to free
                  memory for the program loader which must be called into
                  memory to load the overlays. If the program indicates that
                  memory could not be released or the released memory is not
                  sufficient to contain the loader (most unlikely), remove
                  resident programs (e.g. Sidekick, Windows, etc.) to free
                  memory space. *

        mov      ax,data           ;Put the segment address of data in ax
        mov      ds,ax             ;Move the address of data to ds
        mov      cx,es             ;Put PSP segment address in cx
        mov      bx,seg pgmend     ;Put address of last segment in bx
        sub      bx,cx             ;Find length (in paragraphs) of this program
        mov      ax,4A00h          ;Put Set Block function number in ax
        int      21h               ;Shrink memory allocation
        jnc      memok             ;If carry flag is clear (no error), continue
        mov      ah,09h            ;Else, put Display String function number in ah
        lea      dx,mess0          ;Put message offset in dx
        int      21h               ;Call display string subroutine
        mov      ax,4C00h          ;Put Terminate Process function number in ax
        int      21h               ;Terminate program
memok:

        COMMENT * This block of code initializes the extra segment, displays
                  a message, and allocates 64K bytes of memory (4096
                  paragraphs) to ROOTPGM.  If there is insufficient memory
                  to allocate 64K, a message is printed to that effect.  Then,
                  more memory must be obtained by releasing resident programs
                  or the amount of requested memory must be reduced.  This
                  process is necessary because the program was shrunk to its
                  minimum size.

                  To avoid confusion and possible conflicts, an additional empty
                  code segment in the program is generated as a place for the
                  overlays.  By convention, a 64K byte segment is established.
                  The Allocate Memory function adds this 64K byte segment to
                  the program and places the segment address in ax which one
                  can use to locate the segment.  The shrunk program has grown
                  by 64k bytes.  It is possible to shrink the program to its
                  minimum size plus 64K bytes or to overlay existing parts of
                  the program.   But, such techniques require carefull
                  management by the applications programmer.  It is usually
                  better to use the Allocate Memory function. *

        mov      ax,esdata         ;Put the segment address of esdata in ax
        mov      es,ax             ;Move the address of esdata to es
        lea      dx,root1          ;Put the offset address of root1 in dx
        mov      ah,09h            ;Put the Display String function number in ah
        int      21h               ;Call Display String
        mov      bx,4096           ;Put number of paragraphs in bx
        mov      ax,4800h          ;Put Allocate Memory function number in ax
        int      21h               ;Locate allocated memory
        jnc      allok             ;If carry flag is clear (no error), continue
        mov      ah,09h            ;Else, put Display String function number in ah
        lea      dx,mess01         ;Put message offset in dx
        int      21h               ;Call display string subroutine
        mov      ax,4C00h          ;Put Terminate Process function number in ax
        int      21h               ;Terminate program
allok:

        COMMENT * At this point, the first overlay of the additional code
                  segment can be called.  The parameter block for the Execute
                  Program function must be initialized.  This block is located
                  in the extra segment and consists of two words.  The first
                  word (segadd) contains the address of the segment (paragraph)
                  where the overlay is to be loaded.  The second word (relfact)
                  is the relocation factor (i. e., the segment address of the
                  module which begins execution).  For EXE type overlays, the
                  segment address and relocation factor are normally the same.
                  For COM type overlays, the relocation factor is 0 because
                  only EXE programs are relocatable.

                  The American Standard Code for Information Interchange
                  (ASCII) string at name1 contains the file name of the
                  overlay.  The offset address of this string must be in
                  register dx when the Execute Program function is called.
                  Also, register bx must contain the offset address of the
                  parameter block.  The Execute Program function is called
```



We are still trying to get caught up on our issue.

We would like you to know how much we appreciate your patience. Thank You!



The Electronic Clavier
P/N 885-6016

```
                 by loading 4B03h in ax and executing INT 21h.  The overlay
                 is then loaded at the specified loaction in memory. *

        mov     overseg,ax        ;Store the overlay segment address in overseg
        mov     segadd,ax         ;Put the segment address in segadd
        mov     relfact,ax        ;Put the relocation factor in relfact
        lea     dx,ds:name1       ;Put offset address of overlay file name in dx
        lea     bx,es:segadd      ;Put offset address of parameter block in bx
        mov     ax,4B03h          ;Put function and subfunction numbers in ax
        int     21h               ;Call Execute Program function

        COMMENT * Although the overlay is now loaded, execution proceeds
                 with the next instruction in the root program.  This
                 differs from subfunction 1 of Execute Program which
                 executes the child program. Therefore, the overlay
                 generated by subfunction 3 of Execute Program must be
                 executed by a FAR jump or call (remember, the overlay
                 is in a separate code segment). In order to return to
                 the root segment after the overlay is executed, the
                 current code segment must be saved on the stack and the
                 offset address at which one wishes to resume execution
                 must be pushed onto the stack as well.  Then one can jump
                 to the overlay *

        push    cs                ;Save code segment address on stack
        lea     ax,xit1           ;Get offset address of resume point
        push    ax                ;Save offset address on the stack
        jmp     dword ptr ovad    ;Far jump to overlay with double word address
xit1:                             ;Point at which to resume execution

        COMMENT * The following code overlays the data segment.  This process
                 is not recommended and is included for instructional purposes
                 only.  Data segments can be more safely overlayed by data
                 file processes.  This code demonstrates that overlays can be
                 loaded anywhere in memory.  The process is the same as
                 loading a code segment overlay except that the overlay is
                 loaded at the beginning of the data segment and, of course,
                 no attempt is made to execute the overlay, although, that
                 could be done. Note that only a part of the data segment is
                 overlayed. *

        mov     ax,ds             ;Put the data segment address in ax
        mov     segadd,ax         ;Put segment address in parameter block
        mov     relfact,ax        ;Set relocation factor
        lea     dx,name2          ;Put offset address of file name in dx
        lea     bx,es:segadd      ;Put offset address of parameter block in bx
        mov     ax,4B03h          ;Put function and subfunction numbers in ax
        int     21h               ;Call Execute Program function

        COMMENT * The new data overlayed on the data segment is displayed
                 by the following code.  Note that the data is not
                 referenced by a label.  One should use an offset number,
                 in this case - 0.  Labels are not changed by the overlay
                 process, but they are not transferred from an overlay
                 file either. *

        mov     dx,0              ;Put offset address in dx
        mov     ah,09h            ;Put Display String function number in ah
        int     21h               ;Call Display String function

        COMMENT * The following code overlays the second code segment with
                 a procedure (subroutine).  The first overlay occupies the
                 second code segment until the third overlay overlays it.
                 The process is the same as the first overlay process except
                 for the name of the overlay file. *

        mov     ax,overseg        ;Put overlay segment address in ax
        mov     segadd,ax         ;Put segment address in parameter block
        mov     relfact,ax        ;Set relocation factor
        lea     dx,ds:name3       ;Put overlay file name offset address in dx
        lea     bx,es:segadd      ;Put parameter block offset address in bx
        mov     ax,4B03h          ;Put function and subfunction number in ax
        int     21h               ;Call Execute Program function

        COMMENT * The procedure is called with a far call to the segment:offset
                 address of the procedure.  One cannot use the procedure name
                 because external references are not resolved by the loader.
                 It is not necessary to save cs and the instruction pointer
                 because the "call" instruction does so. *

        call    DWORD PTR ovad

        COMMENT * The memory allocated to the second code segment is freed
                 by the following code.  This process is optional, but is
                 good practice. *
        push    es                  ;Save extra segment address
        mov     ax,overseg          ;Put segment address in ax
        mov     es,ax               ;Move it to es, i.e. redefine extra segment
        mov     ah,49h              ;Put Free Allocated Memory function # in ah
        int     21h                 ;Call Free Allocated Memory function
        pop     es                  ;Restore extra segment address

        COMMENT * An exit message is now displayed to indicate control has
                  passed to the root program. *

        lea     dx,xitmess          ;Put message offset in dx
        mov     ah,09h              ;Put Display String function number in ah
        int     21h                 ;Call Display String function

        mov     ax,4C00h            ;Put Terminate Process function number in ax
        int     21h                 ;Terminate program

code    ENDS

stack   SEGMENT stack
        DB      256 DUP(?)
stack   ENDS

pgmend  SEGMENT                     ;Dummy segment to mark program end
pgmend  ENDS
        END     start               ;"start" must be placed in this line

        COMMENT \ OVERLAY1.OVL
                  This program displays an "*" and the message at offset
                  0 of the root program's data segment.  Note that this
                  program has no data or stack segments.  The instruction
                  "retf" resumes execution at the location in the root
                  program which was stored on the stack. \

code    SEGMENT
        ASSUME  cs:code
start:
        mov     dl,'*'          ;Put "*" in dl
        mov     ah,02h          ;Put Display Byte function number in ah
        int     21h             ;Call Display Byte function

        COMMENT * Labels are not transferred, i.e. externals are not resolved,
                  by the overlay process.  Therefore, memory locations in the
                  root program must be referenced by offset numbers. *

        mov     dx,0            ;Put offset 0 in dx
        mov     ah,09h          ;Put Display String function number in ah
        int     21h             ;Call Display String function

        COMMENT * The following line must be deleted for MASM versions less
                  than 5.0. *

        retf                    ;FAR return, MASM Version 5.0 or above

        COMMENT * Use the following code for returns from MASM version less
                  than 5.0. *
        jmp     continue        ;Jump around data
orad    DW      ?               ;Offset pointer
orads   DW      ?               ;Segment pointer
continue:
        pop     orad            ;Pop original code segment offset address into orad
        pop     orads           ;Pop original code segment address into orads
        jmp     DWORD PTR orad  ;FAR jump to original code segment
code    ENDS
        END  start

        COMMENT * OVERLAY2.OVL
                  This program consists only of a data segment containing
                  one message which will be overlayed onto the root
                  program's data segment.  *
data    SEGMENT
mess    DB  '**This line is displayed after the data overlay',10,13,'$'
data    ENDS
        END

        COMMENT * OVERLAY3.OVL
                  The following procedure displays a message and returns
                  to the calling program.  The message to be displayed is
                  in the procedure's code segment.  Therefore, the data
                  segment is made identical to the code segment so that
                  the Display String function can access the message.  Of
                  course, the original data segment address must be saved
                  before the data segment address is changed and restored
                  before the procedure returns control to the root program. *

code    SEGMENT
```

Continued on Page 36

Quality Enhancements!

EaZy PC Products

EZM-128: Expand 512K base memory to 640K. Simple, plug-in installation. $125.00

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 standard 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

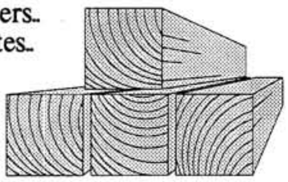
SLOT4: Add fourth expansion slot to right-side accessory bus. $39.95

Order by mail, FAX, telephone, or see your dealer. UPS/APO/FPO 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
Reader Service #104

Splinters..
termites..



With this board
all you have is
a stick in the mud!

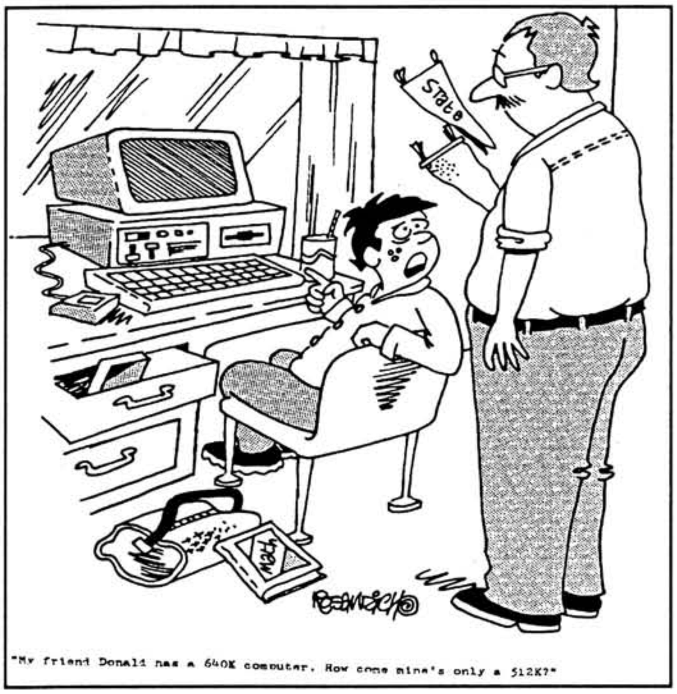
Messages..
Orders..

With this board,
you're really talking!

Which is more useful?

Zenith Data Systems

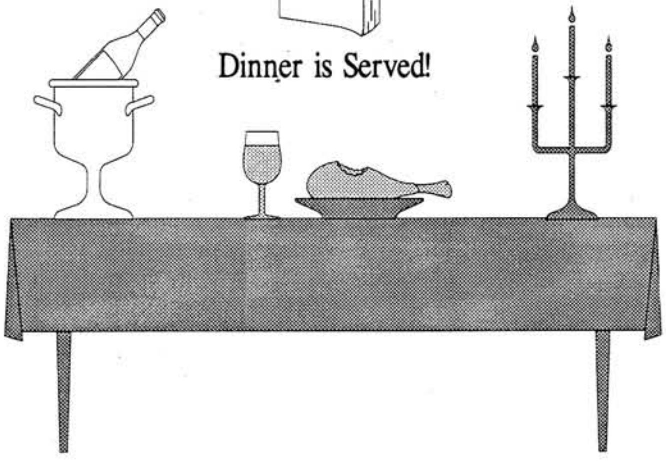
Call (616) 982-3463 for more information



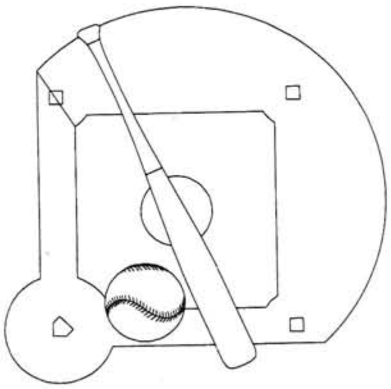
"My friend Donald has a 640K computer. How come mine's only a 512K?"

If you hunger for Computer news...

Dinner is Served!



BY YOUR COMMAND



Chapter 1: Confessions of a Macro User

Richard J. O'Connor
848 Fenske Drive NE
Olympia, WA 98506

Introduction

Computers are a lot like dogs, in that it's often best if you establish your relationship clearly right up front before things get out of hand.

When I began to work on a minicomputer at my office, I found out that the command prompt could be customized to anything I wanted unless it was something useful (like the name of the current sub-directory). This irritated me enough to search around for an appropriate prompt phrase (the default was "OK") that put things into proper perspective. Who's in charge around here, anyway?

Enter the Zylons from the Buck Rogers TV series of several years back (sci-fi fans may correct my memory for names; that's what Buggin' HUG is for!). Those metallic hunks with the room-temperature IQ's were generally only good for harrassing Twikki, worrying Buck, and making the Bettys scream. But their allegiance to The Leader was marvelous, and their response to his every order was the low growl, "By Your Command".

Since that phrase best expresses how I prefer to be addressed by the office mini, that became my prompt. Telling computers (and the occasional human) what to do is a large part of how I earn my living; I expect them to do what I ask and then come back for more orders. This, then is the essence of operating a computer; telling it what you want done, and awaiting delivery of results.

What Do Programmers Know That The Rest Of The World Doesn't?

If this is computer operation (and if you're reading this, you probably have a computer SOMEWHERE that you know how to operate), what, then, is computer PROGRAMMING? A working definition might be: Computer Programming is the art/science/practice of hurling verbs at a central processing unit until you get the desired results. There is a lot more involved in professional programming that has to do with order and logic and insight and design and specific protocols, but the key to getting action from a computer is specifying the right verbs in the right combination.

It occurred to me that it might be enlightening to put together a series-of-the-every-so-often that dealt with helping you tell your computer what to do, even if you aren't classically trained in a programming language. Perhaps your theme song is something like

" Joe Programmer booted up
Feeling somewhat lonely
Didn't know a language, so
He wrote in macros only."

As long as the Poetry Police aren't monitoring REMark, I'll probably get away with that. The point here is that you don't have to rely on Microsoft or Ashton-Tate or WordPerfect Corporation to speak to your computer for you. Sometimes, the best way to get something done YOUR way is to do it yourself. And that's what we're about in this and subsequent articles: giving you examples using macros, batch language, and even BASIC and Pascal that can re-establish YOU as the verb-hurler, and your microcomputer as the box whose only thought between orders is "By Your Command"!

Today's Lesson: When The Standard Output Isn't QUITE Right!

Even a good programmer can't anticipate everything that might be done with the output they generate. And so it was with Cathy's softball statistics manager, a program that converts scorebook entries into box score lines for individual players. Tables are generated for each game as well as for the season to date, and to share this information with our teammates, Cathy would generally print a handful of copies and pass them out at the next game.

Well, you have to understand that this is co-ed recreational slowpitch softball, and we have players representing a wide spectrum of talent, from zero to one, as a programmer might say. This summer, Coach informed Cathy that there were some players who didn't like their individual stats laid out there in black and white for everyone to see. In fact, he would prefer it if she could somehow change her reporting scheme to generate tables for each individual player, which could then be handed out in relative privacy.

```
                    Canyon Country Coyotes vs Suspects
                               Home -- Visitor
                                 12 -- 2

Field:  LBA #2
Date:   6-25-90
Time:   6:00 p

Player               AB    R    H    RBI   2B   3B   HR   BB    K      B.A.    On Base
-----------------------------------------------------------------------------------------
Bonnie Battaball      3    0    0     0     0    0    0    0    0        0          0
Jim Battaball         4    2    3     3     2    0    0    0    0      750       1000
Joan Flyout           1    0    0     0     0    0    0    0    0        0          0
Dan Flyout            1    0    0     0     0    0    0    0    0        0       1000
Denise Homerhitter    1    0    0     0     0    0    0    0    0        0          0
Dan McCannon          4    2    0     2     0    0    0    0    0        0          0
Nadean Mudslider      2    1    0     0     0    0    0    0    0        0        500
Rick Mudslider        0    0    0     0     0    0    0    0    0        0          0
Cathy O'Connor        3    0    2     1     0    0    0    1    0      666        750
Dick O'Connor         2    0    1     2     1    0    0    0    1      500        500
Lynn Singleslapper    1    1    1     0     0    0    0    1    0     1000       1000
John Wellhitball      3    1    2     0     0    0    0    0    0      666        666
```

Figure 1

If you're getting paid to develop software, this is why you get the Requirements List written in ink and signed in the presence of credible witnesses. But the request seemed reasonable, so Cathy and I discussed alternative ways to do what he wanted on the way home that night.

Generally, there are two approaches to a request that comes after the program is already written. You can either add another subroutine to the program, or you can massage the output the program now generates into the desired format. This "massaging" can further take the form of another program, a long and involved session with your favorite editing tool, or perhaps a simple sequence of word processing commands.

Sometimes if you lay the current printout against a sketch of the proposed output, one of your options will jump right up and take over your brain. Figure 1 shows a sample table generated by the program in its original form. (Some of the names of my fellow Coyotes have been changed to protect.you know the drill.) Note that while Jim Battaball might not mind sharing his good fortune with the rest of the world, Denise Homerhitter might have second thoughts. The way Coach *really* wanted the table to look is listed in Figure 2.

Are you thinking what I'm thinking? This problem virtually cries out for a pair of safety scissors and a jar of white paste. In fact, you could create everything the coach asked for with a hardcopy printout of all of the games played to date and a little of that famous cut-andpaste technology we learned in first grade.

If the cut-and-paste metaphor makes sense, use it. Word processors can cut-and-paste electronically, and most of them can repeatedly execute a series of commands, so we chose this option. Remember, there are several approaches to solving problems like this, and you may be led to try a different approach.

```
                         Jim Battaball
                         Oct 23, 1990

Team Name           AB   R   H   RBI  2B  3B  HR  BB   K    B.A.   On Base
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Suspects             4   2   3    3   2   0   0   0   0     750     1000
TCCU                 3   1   3    0   1   0   0   0   0    1000     1000
Les Schwab           5   3   3    1   1   0   0   0   0     600     1000
Brotherhood          3   2   3    2   0   1   0   0   0    1000     1000
Big Tom's            4   1   1    1   1   0   0   0   0     250      250
Taxis Rangers        3   2   1    1   0   0   0   0   0     333      333
Scanners             0   0   0    0   0   0   0   0   0       0        0
Dirty Daves          4   1   2    4   1   1   0   0   0     500      500
Taxis Rangers        4   1   2    2   1   0   0   0   0     500     1000
Gould's Backers      3   1   2    3   1   0   0   0   0     666      666

Cumulative          33  14  20   17   8   2   0   0   0     606      757
```

Figure 2

```
Jim Battaball           Jim_Battaball
Dan Flyout              Dan_Flyout
Denise Homerhitter
Denise_Homerhitter
Dan McCannon            Dan_McCannon
Cathy O'Connor          Cathy_O'Connor
Dick O'Connor           Dick_O'Connor
John Wellhitball        John_Wellhitball
```

Figure 3

```
game_1.90
game_2.90
game_3.90
game_4.90
game_5.90
game_6.90
game_7.90
game_8.90
game_9.90
game_10.90
```

Figure 4

Can You Do It In WatchWord?

A sequence of word processing commands stored in a file for repeated execution is called a "macro." My favorite word processor, PC WatchWord Version 3, supports macros with a language that even allows setting counters, comparing values, and branching to different sections of a macro based on the results of these comparisons. Anything you can do during a PC WatchWord session can be done within a macro, along with these extra capabilities that are more often found in programming languages. One nice thing about WatchWord is that Version 3 of Z-100 WatchWord is so similar to PC WatchWord that the macro we developed will run on a Z-100 with only four command changes!

At this point, I could point you to the macro listing (Listing 1) and leave comprehension as An Exercise For The Reader. Instead, let's learn a few things about macro programming by building it one section at a time.

First, we need to consider the thought processes you would go through if you did this job manually. You would look up the first batter's name, cut it out and paste it on a sheet of paper. Then you would cut out a heading line so that the numbers that followed below made sense to the reader. Next, you would snip out the opposing team's name and the corresponding line of statistics for that player in that game and paste them alongside each other. And back to the next player's name, and so on. Your brain has a better pattern-recognition sense than your microcomputer has, so it's easy for you to see when you've finished the job. The computer has to be taught things like what "done" means, and where the critical pieces of information are, and how to handle the final results. If we break down the chores to be done into single commands, the computer can do the work, and we make the design of the job easier for ourselves.

One way to teach the computer who the players are is to build a disk file listing the name of every player who wants a summary table printed. Since each player's results must be kept separate in the form shown in Figure 2, we could store each table in a separate file. What to name these disk files? In our case, we can build unique filenames using combinations of first and last names of the players without much trouble. Figure 3 shows a file Cathy built for this purpose called ROSTER.TXT. Note that each player's name is listed exactly as it appears in the game summary tables (Figure 1), followed by a version that replaces the space between first and last names with an underscore character. Names in the first column will be used as search strings for WatchWord, while names in the second column can be used to build DOS file names (you can get into

```
 1990 Season (as of 8-14-1990)

Player               AB   R   H   RBI  2B  3B  HR  BB   K   B.A.    On Base
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Bonnie Battaball     29   4   8    4   0   0   0   1   2    275       300
Jim Battaball        33  14  20   17   8   2   0   0   0    606       757
Joan Flyout          29   4  10    8   0   0   0   1   0    344       433
Dan Flyout           16   1   9    3   1   0   0   1   0    562       647
Denise Homerhitter   21   1   5    5   1   0   0   0   0    238       285
Dan McCannon         31  11  14    6   3   0   0   0   0    451       548
Nadean Mudslider     28   9   9    3   0   0   0   2   0    321       400
Rick Mudslider       12   4   6    4   1   1   0   0   0    500       583
Cathy O'Connor       31   6   7    5   1   0   0   2   0    225       333
Dick O'Connor        30  12  14    9   3   0   0   0   1    466       533
Lynn Singleslapper   28   2   9    2   0   0   0   3   1    321       451
John Wellhitball     24   6   6    6   0   0   0   2   0    250       423
*****************************************************************************
```

Figure 5

more trouble than it's worth trying to use DOS file names with spaces embedded within them). The filenames will be constructed with the first 8 characters of the underscored name, and an arbitrary extension (we used .ST, to stand for STatistics).

Entering a header line (actually, two lines consisting of abbreviations like AB, R, H, RBI, etc. and a dashed separator line) into the file we build for each player is much faster if this information is stored in a separate file that WatchWord can read in when needed. Cathy created such a two-liner, and called it PLAYER.SHL.

Now the computer knows who to look for in the game summary tables, and where to store the results of all games. Each game summary table is already stored in its own disk file, courtesy of the original program. But how do we tell the computer what the names of those files are?

If you guessed "another list of names," you're thinking like a verbhurler! Figure 4 shows the contents of a simple file called GAMEFILE.LST, which just lists the names of the game summary files, one per line. At this point, Cathy decided to make use of another file built by the softball program called LIFESTAT.TXT, which keeps a running total of batting statistics across all games through the season. (A portion of this file is listed in Figure 5.) By pulling out the appropriate line from the last table in this file, we can add the cumulative statistics for each of our players without doing any further arithmetic! Now with files ROSTER.TXT and GAMEFILE.LST, the games files themselves, PLAYER.SHL and the LIFESTAT.TXT cumulative statistics file, we finally have all of the props we'll need for our production.

`Part 1: Setting It All Up`

Before starting, there are a few things to clarify. The macro listing we're working with is the one for PC WatchWord, which prints the PGUP sequence as an a-umlaut (two dots above the 'a'), the GDN sequence as a division sign, the key as a graphic that resembles a thin, backwards 'u', and the Shift-UpArrow and Shift-DownArrow sequences as box-drawing characters. If you see odd-looking symbols like these in the listing, refer to nearby comments, which follow semicolons. Hereafter, when I refer to WatchWord, I generally mean both the Z100 and PC versions. However, the testing and timing was all done using the PC version.

One thing that both flavors of WatchWord can do easily is work with two files simultaneously, using a horizontally split screen for easier viewing. WatchWord also has variables (named @1, @2, @3,.,@9) and parameters (named @A, @B, @C,.,@Z) that you can use to hold useful items like file names and line counts. We wrote this macro so that the user could specify the name of the roster file and the name of the games list file at execution

Listing

```
;*************************Macro PS.MAC*********************************
;*    Macro Form :  PS/roster filename/games list filename/             *
;*    String Values :                                                   *
;*                       @1 -- Roster filename                          *
;*                       @2 -- Filename containing the list of          *
;*                             game files.                              *
;*                       @3 -- player name from the roster file         *
;*                       @4 -- player filename for saving the           *
;*                             stat file to be created                  *
;*                       @5 -- games list file. Contains the filenames  *
;*                             of games to be parsed into individual    *
;*                             records.                                 *
;*    Variables :                                                       *
;*                       @A -- # of players counter. Obtained by setting *
;*                             @Z using the PARAM command. @Z is set to the *
;*                             number of lines in the buffer.  @A = @Z   *
;*                       @B -- counter to keep track of the number of   *
;*                             completed.  When @A = @B the main loop is *
;*                             finished.                                *
;*                       @C -- line counter in stat file being created. *
;*                       @D -- # of games counter. @Z is once again set *
;*                             using the PARAM command after reading the *
;*                             games list file. When @D = @Z all games  *
;*                             information has been collected for the   *
;*                             current player.                          *
;***********************************************************************
;   Opening Routine
;***********************************************************************
SPLIT1
;read the roster file into the top buffer
READ @1
PARAM
SET A @Z
SET B 0
JUMP i
;***********************************************************************
;   Read_Next_Player Routine
;***********************************************************************
EAD @1
;***********************************************************************
;   Build_Table_For_Current_Player Routine
;***********************************************************************
       ;
        -GUP move to TOP of file
DOWN @B
;get a player's name
ZONE 1 20
SET ZONE ARG 3
ZONECOPY 1
ZONE 25 32
;get a player's filename
SET ZONE ARG 4
;get the shell used to build the statistics textfile
READ PLAYER.SHL
FN @4.ST

        -GUP move to TOP of file
;paste player's name and center it
PASTEOVERLAY
CEN
;paste current date and center it
OVERLAY DATE 1
CEN
;switch to the bottom buffer
SWITCH
SET C 5
;read the games list file
READ @2
PARAM
SET D 0
JUMP h
;***********************************************************************
;   Read_Gamefile Routine
;***********************************************************************
               ;
        -GUP move to TOP of file
DOWN @D
;***********************************************************************
;   Extract_Player_Stats Routine
;***********************************************************************
filename to obtain stats from this game for this player
ZONE 1 80
SET ZONE ARG 5
READ @5
;locate the player's name in the game file
```

```
L/@3/
JUMPF e
;get the player's stats
GOCOL 22
=                              ;  the SELECT command
GOCOL 80
COPY 1
;locate the opponent name by finding "vs"
-L/vs/
GOCOL 46
=                              ;  the SELECT command
GOCOL 80
COPY 0
;switch to the top buffer
SWITCH
                               ;
        -GUP move to TOP of file
ADD C 1
DOWN @C
;paste the name of the opponent
PASTEOVERLAY 0
;The next two strange-looking commands do a Shift-UpArrow and
Shift-DownArrow.
4
9
UP 1
;paste the player's stats
GOCOL 22
PASTEOVERLAY 1
;*************************************************************************
;   Add_YearToDate_Stats Routine
;*************************************************************************
bottom buffer
SWITCH
ADD D 1
COMP @D @Z
JUMPN g
;when finished with the game files get the current cumulative stats
READ LIFESTAT.TXT
v                              ;
        -GDN  move to BOTTOM of file
;locate the player's name, searching backwards through the most
;recent season stats entry
-L/@3/
JUMPF j
;put stats into the paste buffer
GOCOL 22
=                              ;  the SELECT command
GOCOL 76
COPY 0
;switch to the top buffer
SWITCH
                               ;
        -GUP move to TOP of file
ADD C 1
DOWN @C
;Do a Shift-UpArrow to enter screen mode in order to add "Cumulative".
;The following Shift-DownArrow returns to command mode.
4
Cumulative
9
UP 1
;paste stats
GOCOL 22
PASTEOVERLAY 0
;adjust the last two columns to align with the rest of the stats
GOCOL 60
=                              ;  the SELECT command
GOCOL 75
COPY 1
;copy selected text into paste buffer 1
GOCOL 61
PASTEOVERLAY
;switch to the bottom buffer
SWITCH
;*************************************************************************
;   Finish_Player_File Routine
;*************************************************************************
top buffer
SWITCH
SAVE
;check the number of players done
ADD B 1
COMP @A @B
JUMPN f
NOSPLIT
                               ;
```

time for greater flexibility. For example, I start this macro from within WatchWord with the command

```
  MACRO
PS.MAC/ROSTER.TXT/GAMEFILE.LST
```

WatchWord's use of labels (@a, @b, @c,.,@z) adds further flexibility by allowing the computer to branch, or "jump" to a different section of the macro file and continue working. Note that our finished macro has seven labelled sections; let's see what they do.

`Section 1: Opening Routine`

The first thing we do is command WatchWord to split the screen so we can work on two files at the same time later on. This isn't strictly necessary, but it becomes helpful if you want to watch the macro execute during the "debugging" phase. Then the first file named on the macro execution call is read into WatchWord; this happens to be the ROSTER.TXT file. The PARAM command does several useful things; for now, we make use of the fact that it counts the number of lines in the current (ROSTER.TXT) file, and stores that number in the variable @Z. We can use this number as a counter by storing it in a parameter like @A, so this is what the SET command does.

`Section 2: Read_Next_Player Routine`

The main "loop," or sequence of repeated instructions, begins here. The @f label gives us a target for subsequent JUMP commands that will cycle through this loop again. The only action of this routine is to read the first of the files specified when you started the macro into WatchWord.

Note that there were certain things we only wanted to do once in this macro, such as setting variables A and B. This is the purpose of Opening Routines. Since we read the ROSTER.TXT file into WatchWord at the start in order to set these variables, we can skip the Read_Next_Player routine the first time through with a JUMP i command and save a little time.

```
  Part 2: Creating A Player File
Section 3:
Build_Table_For_Current_Player
Routine
```

Each time WatchWord runs this section, a new file will be built for the current player. We start by using the GUP command to move to the top of the file. The parameter @B holds the number of lines we have to move down to reach the name of the next player to summarize; initially this is zero. We set a zone from columns 1 to 20, define @3 as the name of the variable that will be filled with the next zonecopy command, and then place the contents of columns 1-20 (the player's first and last names) into @3. Similarly, the contents of columns 25-32 (first 8 columns of the player's firstname_lastname) are stored in variable @4. WatchWord then reads in the header information, names the current working file with @4 as

```
        -GUP move to TOP of file
;clear the screen
DEL *
;empty the paste buffers used
EMPTY 0
EMPTY 1
;signal the end of the macro, leave macro and quit WatchWord
RING
EXITS/Finished with Print Stats job successfully!/
```

filename and ST as extension, pastes in the player name at the top, centers it, adds a centered date below it, and switches to the lower file buffer to read in the GAMEFILE.LST file. PARAM is used to set @Z to the number of lines in this file, which is simply the number of games played to date.

`Section 4: Read_Gamefile_Routine`

An inner loop begins here, where the game summary table for each game is checked, and all statistics for the current player are copied out into the growing player file. The GAMEFILE.LST file is read in, WatchWord moves to the top of the file, then down "@D" lines to reach the line containing the name of the next game summary file. Initially, GAMEFILE.LST has already been read and @D set to zero, so this routine will be skipped.

`Section 5: Extract_Player_Stats_Routine`

First, the current file name is read into variable @5, and then the file itself is read in, displacing the GAMEFILE.LST information. The Locate command uses the contents of variable @3 (set to the current player's name) to look for a line of batting stats for this player in this game. If the search fails, the JUMPF command drops control down to the next section, where we begin the Are We Done Yet? tests. Otherwise, the stats from columns 22 through 80 are placed in buffer number 1 using the GOCOL, SELECT, and COPY commands. A reverse Locate finds the opposing team name on the line with 'vs' (versus) on it, and places that name in buffer 0.

Then we switch back to the player file under construction, increment the line counter @C, move down that many lines from the top, and paste in the opposing team name. In order to append the stats themselves, WatchWord requires that a blank line be created in the file, which can be done by moving from Command mode into Screen Mode and back again. (The PC WatchWord commands for this are Shift-UpArrow and ShiftDownArrow.) The stats are then pasted in, and a new line has been added to this player's file.

`Section 6: Add_YearToDate_Stats_Routine`

At this point, we need to check if there are any more game files to read. The number we've read so far, @D, is incremented and compared with @Z, the number of lines in GAMEFILE.LST. Until these numbers are equal, control will shift back to Section 4, where the name of the next game file is obtained, and the search for statistics repeats.

When all game files have been searched, the LIFESTAT.TXT file is searched from bottom to top to find the most recent entry for the current player, and this line is copied into the player file through another series of GOCOL, SELECT, COPY, SWITCH and PASTE commands. The word "Cumulative" is used instead of an opposing team name, the columns are aligned, and the player file now looks just like Figure 2.

`Part 3: The End Game`
`Section 7: Finish_Player_Routine`

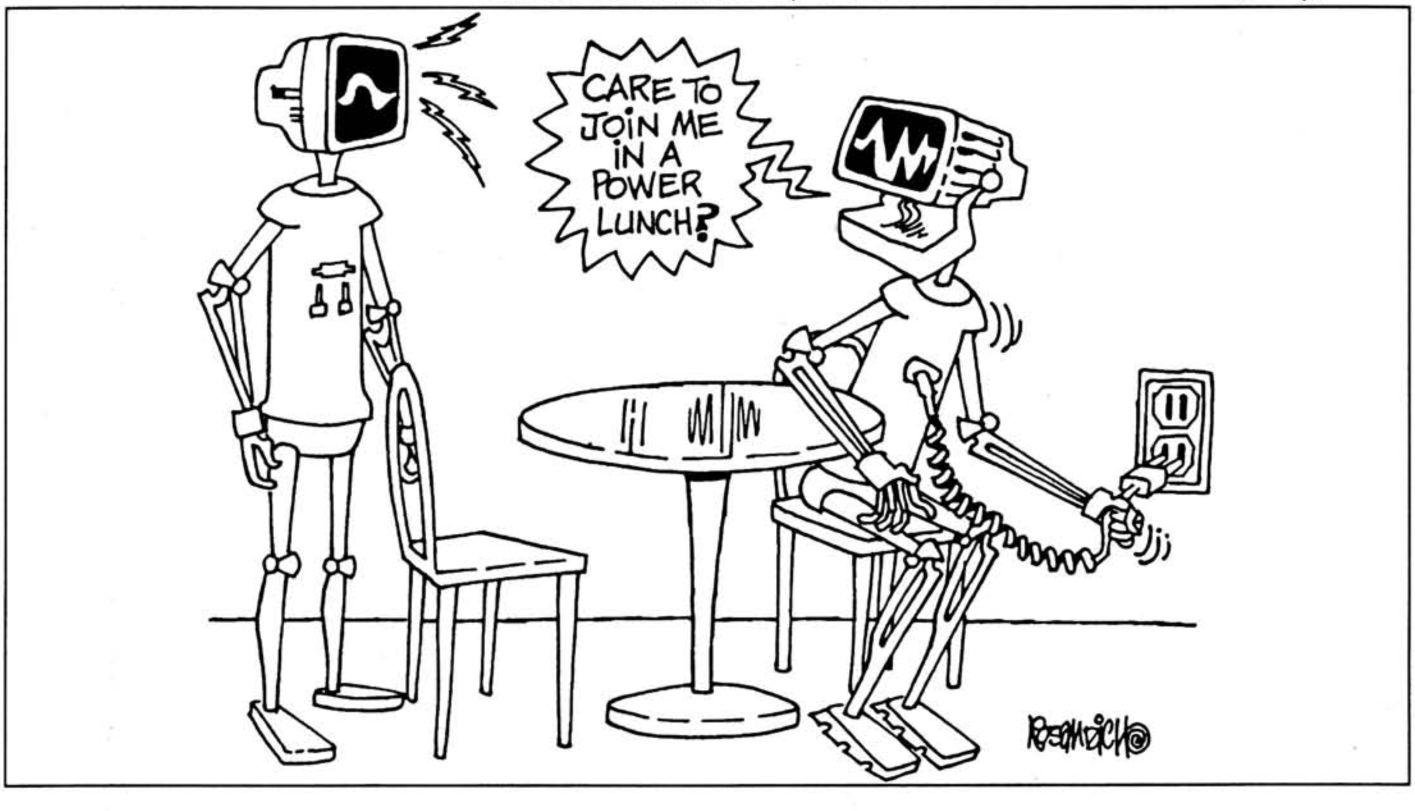
We're done with that player; are there still others to build tables for? In this section, the player file is saved, the number done so far (@B) is incremented, and compared to @A, which was set to the number of lines in the roster file during setup. If there are more players to do, control shifts back to Section 2, where the next player's name is read, and the loop begins again.

Once all players have their summary tables built, the only chore left is to do a little screen and buffer cleanup, and to signal the waiting user to put down the REMark magazine and prepare to print out the files.

On our 12 MHz Zenith Z248, it takes PC WatchWord about 2.5 minutes to run this macro for seven players and ten softball games. There are any number of ways to write a utility to do this faster. But the purpose of this exercise was to see what could be done just using a word processor. It's not "language" programming in the strictest sense, but macro programming is a powerful way for any user who can plow through a manual to start ordering their computer around.

Are you "in command" yet? What are you waiting for???

NEXT TIME: something for BASIC lovers, and others who can type!



ZAP

No More Data!

Robert D. Miller
860 Isis Court
Reno, NV 89512

There have been many articles published in computer magazines about the damage to computer data from power line surges. There has been very little written about the damaging effects of Electro Static Discharge or ESD.

In my job as Quality Engineer, one of my main responsibilities is to create and maintain the company Quality Control procedures. After spending many hours on one procedure (about 30 pages long), I delivered the diskette to the document control administrator. He reached up to take the disk from me and ZAP there was a static discharge. A shock so great that we both jumped.

I decided to check to see if any data was lost, I found the diskette could not be accessed or even formatted. The diskette was completely damaged. Fortunately this was a copy of the procedure, the original was still on my computers hard drive.

The electronics industry has been plagued with ESD since 1960. One aircraft company, Douglas Aircraft Corporation, became aware of ESD when replacing some of their vacuum tube avionics systems with the latest solid state (transistor) technology. Although the new systems were direct replacements, there were four times the number of soft failures or failures that do not reoccur during retest.

Through extensive research, their engineers discovered that static electricity was the cause of these problems. They found there were a number of causes for the generation of static. If a person walks across a carpet and touches a door knob, they can generate as much as 40,000 volts of static electricity (Table 1). To develop a charge that big will occur when the humidity is low (10-20%), but the charge will be only about 1,000 volts with 65-90% humidity.

An electrostatic charge is a build up of static electricity from a process called the triboelectric effect. The triboelectric effect is the rubbing of two dissimilar materials and rapidly separating them. A good example as above, is just walking across the carpet. Another example is lifting a plastic bag off of the work bench.

An electrostatic charge is energy at rest, but is released when you touch a conductive object like a metal door knob (or in my case, even a computer diskette). That ZAP we received when passing the diskette had to be a healthy 20 to 30 thousand volts to arc thru the plastic disk folder and damage the floppy diskette inside.

Table 1 shows the various conditions that will create static build up on your body.

Table 1
Typical Electrostatic Voltages

Means of Static Generation	**Electrostatic Voltages**	
	10-20% Relative Humidity	65-90%
Walking across vinyl floor	35,000	1,500carpet
Walking over vinyl floor	12,000	250
Worker at bench	6,000	100
Common plastic bag picked up frombench	20,000	1,200
Work chair padded with polyurethane foam	18,000	1,500

Further research revealed that a person just begins to feel the sensation of the discharge when the voltage reaches 3,500 volts. A visible arc can be seen at about 4,000 volts with the corresponding crack at about 4,500 to 5,000 volts.

Another hazard with ESD and computers are the effects on the electronic components themselves. Primarily the integrated circuits. An Electrostatic Discharge will cause such an arc, in a brief moment of time, one milli-second (one thousandth of a millionth of a second) that a tremendous amount of heat is generated (as much as 1600 degrees Fahrenheit). That kind of temperature will vaporize a transistor junction or integrated circuit. Many transistors only require a hundred volts or less to be completely destroyed.

Only a few years ago, an integrated circuit contained only a hundred or so transistors, but today integrated circuits like the microprocessor in your new 80386 computer will contain thousands of transistors. This makes the integrated circuit sensitive to even lower voltages.

When a charge of 3 to 4 thousand volts is required before you can even feel the spark, a transistor or IC usually fails completely. The real fun comes when there is an arc that cannot be seen or felt and is presented to that new batch of RAM chips that was just added during your computer memory expansion. The IC is only partially damaged (wounded), it still works, but is not fully functional.

I have built a number of kits over the years and have seen very little information regarding the precautions of ESD. The instructions only talk about not touching the transistor or integrated circuit leads. Another precaution tells you to touch the chassis of the computer before handling the devices, or removing them from their static dissipative containers.

The problem with only touching the chassis is, if you let go, you can build a new charge of up to 6000 volts by moving around at your work bench (refer to Table 1). Now when you get up to answer the phone, forget to touch the chassis again, set the device on something conductive, ZAP you damaged the device.

Below are some quotations from computer manuals that I have in my library. The information pertains to the upgrading the computer by adding an integrated circuit or accessory circuit board.

"CAUTION: Some integrated circuits are electrostatic sensitive and can be damaged by static electricity if they are handled improperly. Once you remove a circuit board from its protective packaging or the computer, do not handle the board unnecessarily."

"CAUTION: Some integrated circuits are electrostatic sensitive and can be damaged by static electricity if they are handled improperly. Once you remove the IC from its protective packaging, do not lay it down or let go of it until it is installed in the board. When you bend the leads of the IC, hold the IC in one hand and place your other hand on the work surface before you touch the IC to the work surface. This will equalize the static electricty between the work surface, you and the IC."

If you decide to add a plug in hard drive or an extended RAM board to your computer more problems can occur. Several components can be damaged by careless handling. Some people believe that if you handle a circuit board by the edges and have the charge on your body equal the charge on the circuit board, the components won't be damaged.

The problem is the board has roughly the same charge as your body and if you set the board down on the metal cover of your computer, the charge dissipates from the board to the conductive computer top, destroying some chips along the way.

Table 2 shows the ESD sensitivity of various semiconductor components found in your computer.

Table 2
Semiconductor Static Sensitivity

Device Type	**ESD Sensitivity Range (Volts)**
VMOS	30-1800
MOSFET	100-200
GaAsFET	100-300
EPROM	100-
JFET	140-7000
SAW	150-500
OP-AMP	190-2500
CMOS (input protected)	250-3000
Schottky Diodes	300-2500
Film Resistors (thickThin)	300-3000
Bipolar Transistors	380-7000
ECL (P.C. Board Level)	500
SCR	680-1000
Schottky TTL	1000-2500

I would like to repeat the "10 Myths of Static Damage" [10] here because I think they are very important to the servicing and use of your desk top computer:

1. Touching the equipment frame to ground yourself is an adequate static-safe procedure.

WRONG. This will quickly drain the existing static charge but fails to prevent regeneration. Even simple body movements can build up enough charge to damage sensitive components. Only by wearing a grounded wrist strap can you ensure the charges will be drained off and new charges will not build-up.

2. Once components are mounted on a PC board, they are safe from static damage.

WRONG. The chance of damage is even greater because the board component interconnections (traces) act to funnel the charge directly to the device. It is very hard to handle a circuit board and not touch leads or circuit traces.

3. Only circuit boards with CMOS devices require static protection.

WRONG. Most PC boards manufactured today have half of their devices of the CMOS variety or have greater sensitivity than CMOS.

4. You must actually touch the PC board or component to cause static damage.

WRONG. Simply exposing a component to the electric field surrounding a charged object (styrofoam cup or plastic bag) will cause a break down of the dielectric layer inside. This will occur without actual contact taking place.

5. PC boards that test "good" after improper handling have not been damaged.

WRONG. There is documented proof that the devices may only be weakened, causing premature failure or erratic operation.

6. Shipping and storing PC boards in pink antistatic polyethylene provides complete and effective static protection.

WRONG. Boards contained in "pink poly" only prevents static buildup. Components can still be "Zapped" by a charge from someone without a grounded wrist strap. The bag must also have a metallic shield impregnated in to the plastic.

7. Only electronic equipment located in a carpeted area requires static protection.

WRONG. Static charges from just walking on a vinyl floor can build sufficiently to damage many semiconductor components.

8. If you keep the humidity high enough around the equipment static will be eliminated.

WRONG. Static charges aren't as high but they are still dangerous. Besides airconditioning usually reduces the humidity in many environments.

9. Topical antistats (sprays, wipes, etc.) are all the protection necessary.

WRONG. Topical antistats are temporary and become less effective with lower humidity. They provide no shielding protection.

The last myth pertains to field service technicians.

10. Technicians will refuse to wear wrist straps or use static precautions.

WRONG. With proper training, technicians realize how static precautions can reduce their work load with a reduction in "NPF" reports (No Problem Found).

To reduce the possibility of losing data on your diskettes or damage to your computer, there are several companies that make anti static products to help protect your work station.

One product is an anti-static strip (fig. 1) that attaches to your keyboard. The conductive strip has a wire with a metal ground lug on the end. The lug is attached to ground at the wall power receptacle with the screw that holds the cover on.

If you touch the strip when you sit down at your computer, any static charge that is built up on your body will be safely directed away to ground. Now when you handle your diskettes there is less chance of damaging the data with static. Some of the older computers could even be damaged from static conducted through the keyboard.

Another device available for discharging static at the computer workstation is an anti-static pad (fig. 2) that the computer sits on and is attached to ground. There is a small box (fig. 3) that sits next to the keyboard and "lights up" when you touch the test point, to show you that the charge has been safely dissipated.

If you are going to upgrade your computer with a RAM board, new RAM chips, a math co-processor chip, a new hard drive or whatever, extra precautions are required here too. Several companies offer a conductive wrist strap (fig. 4) that is attached to ground, at the receptacle cover screw. There is a receptacle test adapter (fig. 5) with three lights that show the receptacle is wired correctly. The test adapter also has a ground jack that the wrist strap wire plugs into.

When you are wearing the wrist strap, you can move around your work bench, remove the RAM chips (or any other electronic component or accessary) from its static safe container and safely install it into your computer. Even better is the additional use of a static dissipative bench pad (fig. 6,7) that is also connected to ground.

When you place a static dissipative container holding the ICs or circuit board on the pad, it will dissipate any charge that has built up. Now you can safely remove the

ICs and place them on the pad until you plug them into the board. Don't forget your wrist strap.

The environment around your working area is just as important in eliminating static. Styrofoam cups, plastic bags, vinyl chair covers as well as the kind of clothing you wear should be carefully controlled.

If you have transistors or integrated circuits that must be soldered in place, another source of ESD is the soldering iron. The soldering iron (fig. 8) must have less than two (2) ohms resistance between the tip of the iron and the ground wire on the power cord. The iron must not have more than two (2) millivolts rms from the tip to ground. Any iron that meets DOD-STD-2000 or MIL-STD-2000 (as indicated on the soldering iron brouchure) are safe for ESD sensitive components.

If you are going to de-solder the devices from a printed circuit board and you use a Soldapullt (that is how it is spelled) by Edsyn, (fig. 9) be sure you only use the "Silverstat" static conductive version.

So for a safe work environment when using your computer, have at least a grounded ESD strip attached to your keyboard. Then touch the strip before touching your keyboard or handling your floppy discs. When upgrading your computer with RAM chips, clock chips, CPU chips, hard drives or whatever, wear a conductive wrist strap.

Remove the devices from their conductive containers, then plug them in to their sockets. Or better yet use an antistatic pad as well as a wrist strap.

These static dissipative items are not expensive when you consider the frustration of intermittent problems with the operation of your computer, or damaged diskette. I consider it good hardware insurance. I consider the wrist strap to be a minimum precaution.

References

D.E.Frank, *"SOFT FAILURES-THE INVISIBLE MODE,"* Annual Reliability and Maintainability Symposium, March 1983.

D.E.Frank, *"ESD CONSIDERATIONS FOR ELECTRONIC MANUFACTURING,"* McDonnel Douglas Corporation, Annual Reliability and Maintainability Symposium, January 1982.

J.R.Huntsman, *"ELECTRIC FIELDS, STATIC DAMAGE, and SHIELDING;"* Static Control Systems/3M,225-4S, 3M Center, St. Paul, MN 55144

D.Yenni, *"BASIC ELECTRICAL CONSIDERATIONS IN THE DESIGN OF A STATIC-SAFE WORK ENVIRONMENT,"* Nepcon West, 1979.

D. Gleeson & B. Russeth, *"10 MYTHS OF STATIC DAMAGE,"* Static Control Systems/3M,225-4S, 3M Center, St. Paul, MN. 55144

Suppliers

RADIO SHACK
500 One Tandy Center
Fort Worth, TX 76102

HUB Material Company
33 Springdale Avenue
Canton, MA 020201.

Technitool
Box 368
Plymouth Meeting, PA 19462

MISCO
One Misco Plaza
Holmdel, NJ 07733

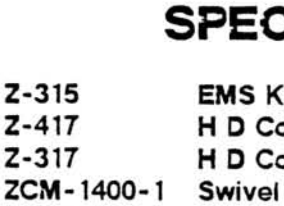
GLOBAL Computer Supplies
2318 East Del Almo Blvd., Dept. 97
Compton, CA 90220 ✻

W S Electronics

(513) 376-4348 ** Since 1975 **** (513) 427-0287**

1106 State Route 380, Xenia, Ohio 45385

YOU AND YOUR BIG IDEAS.



* 300 cps draft speed
* Wide carriage
* 24 pin printing
* Front panel controls

ALPS *Allegro 500XT™*

THE ALPS ASP1600 PRINTER. COSTS VERY LITTLE, JAMS NOT AT ALL.

- Auto tear bar prevents paper waste. With the flick of a button, your fanfold paper advances to the perforation, then automatically returns to top-of-form position.
- Rugged 9-pin head delivers crisp output at 192 cps in draft mode. 38 cps in letter quality.
- Compact 9-lb. body makes for easy portability.
- Printer stand is built-in.
- Prints labels easily.
- Full Epson FX-85 compatibility.

ALPS *AMERICA*
Built by popular demand.

SPECIAL

Z-315	EMS Kit for Z-159	$ 10.00
Z-417	H D Controller Z-248	$100.00
Z-317	H D Controller Z-150	$ 75.00
ZCM-1400-1	Swivel Base for 1490	$ 5.00
CB-4364-39	Diagnostics for 184	$ 5.00
ENABLE 2.0	Word Processing	$ 99.00

Quantities limited to stock on hand

ATTENTION:

Federal Government Offices
We stock ALL ALPS Printer Models and we stock ALPS PARTS and RIBBONS for all ALPS models including your P2000's and ASP-1000's

****GOVERNMENT DISCOUNTS OFFERED****

We are looking for good dealers.
ALPS Authorized Distributor and Service Center

Reader Service #149

Getting Started With . . .

Alan R. Neibauer
11138 Hendrix Street
Philadelphia, PA 19116

Instant Recall (1.2)

Personal Information Managers, or PIM's, are designed to organize your work. They keep track of tasks you and others have to perform, your schedule of activities, and people you work with.

Unfortunately, for all of their value, PIM's have not taken the software world by storm. Perhaps the problem is the targeted audiences. Those managers who could most benefit from using a PIM are not the most likely to be using a computer daily.

After all, using a PIM requires logging of meetings, activities, and people every day, so some amount of time must be spent at the keyboard entering data. If you do not want to spend that time on the computer, then a PIM will be of limited use. However, if you are already using your computer daily, a PIM can be an excellent management tool. And if you have a complex schedule, it is worth your while to get to the keyboard a few minutes each day to use the PIM, even if you use the computer for little else.

In my article Getting Starting with PIM's (Remark, January 1991) I discussed PIM's in general using two examples to illustrate their functions — IBM's Current which runs under Windows, and Primetime Personal, a DOS application. In this article, we'll focus on Instant Recall, a PIM from Chronologic Corporation, to see how it can be used to keep track of time for consultants, lawyers, and others who bill clients based on time spent on projects.

In my own business, for example, I must keep track of each block of time I spend programming or writing, each related phone call, and each visit. If I forget to log an item I probably won't include it in my bill.

Since I spend a great deal of time at the computer, I can keep track of my activities using Instant Recall, or another PIM. I log each call, each section of time I work on a project, then include it in my monthly invoice.

There are programs designed specifically for client billing. However, by using a PIM I have all of the other benefits that this class of software provides. With Instant Recall, this includes, for example, a clipboard for copying text from one application to another, a task and schedule manager, and a telephone-book with a dialer. The telephone book even allows me to copy names and addresses directly into my word processor for addressing letters and envelopes.

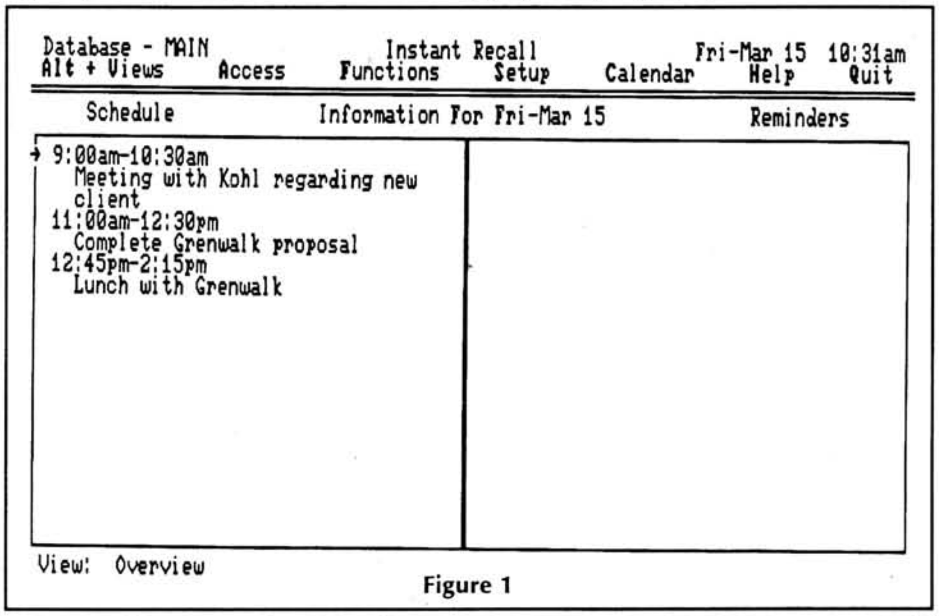
Before showing how Instant Recall can be used for billing, let's look at Instant Recall in general. It offers all of the features you would expect from a DOS-based PIM. As a TSR program, it can be popped up from within any application, even from within Windows 3.0, making it easy to manage your schedule while you are working on your computer.

Interacting with Instant Recall

The class of information stored by Instant Recall is called a view. Each view let's you see information in a certain perspective. Figure 1 shows the main Instant Recall screen in the overview mode. This displays the current day's schedule on the left and any special reminders on the right.

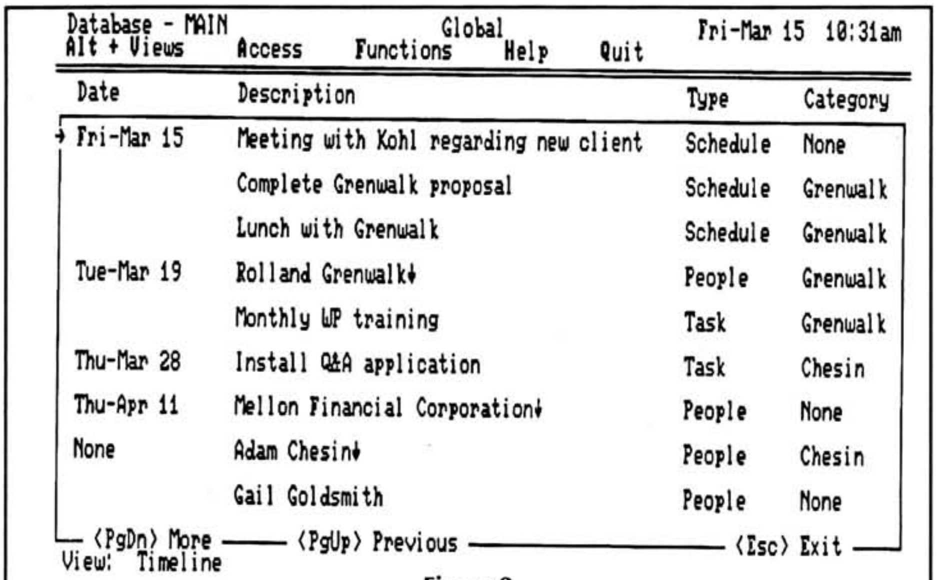
The menu bar along the top of the screen shows the basic categories of operations that you can perform.

View lets you select the type of information you want to display — notes, tasks, appointments, people, schedules, or files. Instant Recall provides two additional view types, Global and Open Time. Global, as seen in Figure 2, shows all of the information in the database sorted by time. Using the global view, you can quickly see how the various types of information relate. The Open Time view, as shown in Figure 3, graphically displays your schedule for the



```
→ 9:00am-10:30am
    Meeting with Kohl regarding new
    client
  11:00am-12:30pm
    Complete Grenwalk proposal
  12:45pm-2:15pm
    Lunch with Grenwalk
```

Figure 1


```
Database - MAIN                        Global              Fri-Mar 15  10:31am
Alt + Views    Access    Functions    Help    Quit
 Date          Description                              Type        Category
→ Fri-Mar 15   Meeting with Kohl regarding new client   Schedule    None
               Complete Grenwalk proposal               Schedule    Grenwalk
               Lunch with Grenwalk                      Schedule    Grenwalk
  Tue-Mar 19   Rolland Grenwalk↓                        People      Grenwalk
               Monthly WP training                      Task        Grenwalk
  Thu-Mar 28   Install Q&A application                  Task        Chesin
  Thu-Apr 11   Mellon Financial Corporation↓            People      None
  None         Adam Chesin↓                             People      Chesin
               Gail Goldsmith                           People      None
└─ <PgDn> More ──── <PgUp> Previous ──────────────── <Esc> Exit ──┘
View:  Timeline
```
Figure 2

day, explicitly listing periods of open time.

Access lets you select specific types of information, such as data relating to a specific individual, priority, or key phrase.

Functions include actions that you can perform on the database, such as adding, modifying, or deleting information.

Setup is used to change the display and system characteristics, enter passwords, and modify the hot key used to call up Instant Recall.

Calendar allows you to change the date and select other days for review.

Instant Recall works with both a mouse and keyboard. If you are using the keyboard, you access the menu bar using ALT key combinations, such as Alt-V to pull down the Views menu. Once a menu is displayed, you can press the arrow keys to move from menu to menu, or ESC to return to the displayed function.

Fortunately, Instant Recall is designed so you can avoid clumsy key combinations to perform most functions. When you are not actually entering data, single keystrokes are used to initiate operations. For example, from the overview mode, you can quickly change to the people view by pressing P, or change databases by entering F for files. This process lets you perform many functions faster with the keyboard than with the mouse.

Suppose, for instance, that you are using WordPerfect and want to add an appointment to the schedule. You would press Shift-Ctrl to call up Instant Recall, then S A (for Schedule view, then Add) or A S (for Add Schedule). Either combination will immediately display the form for adding an appointment.

In most views, the keystroke functions are consistent. A always adds, M modifies, and D deletes. Use Z to zoom in, or display the details on a particular item, or J to jump to a different date.

R prints a report based on the current view. You can select to print details on a single entry, multiple entries, or all of the entries in the view. Output can be sent to either the printer or a disk file. Instant Recall prints all reports as ASCII text, there are no installable printer drivers to take advantage of special fonts or other printer features.

When data about a person is displayed, you can also press U to use the phone dialer, or L to print mailing labels from the address information.

The mailing label function lets you print labels, or send the address information to the clipboard or directly to the keyboard in the program running when you popped up Instant Recall. (You'll learn about the clipboard later.)

For example, suppose you use Instant Recall to maintain your mailing list but you want to use WordPerfect to generate a letter. Instead of typing the address in the letter, start WordPerfect, then pop up Instant Recall and display address information by pressing Shift-Ctrl P, for the people view. Figure 4 shows how the people view appears. Select the address you want then press L (for Labels), Enter, then F1, which selects keyboard destination. WordPerfect pops back to the screen and the selected address appears character by character just as if you were typing it yourself.

The Clipboard

Instant Recall's clipboard is designed to transfer text from one application to another, or to and from ASCII files on disk. While it is part of the program, it can be called up independently, without actually popping up the PIM itself. I often find myself using the clipboard almost as a separate application, during computing sessions when I do not use the rest of Install Recall at all. Once Instant Recall is loaded into memory, and from within it, another application, or at the DOS level, press Shift-Ctrl-C to pop-up the clipboard menu with five options.

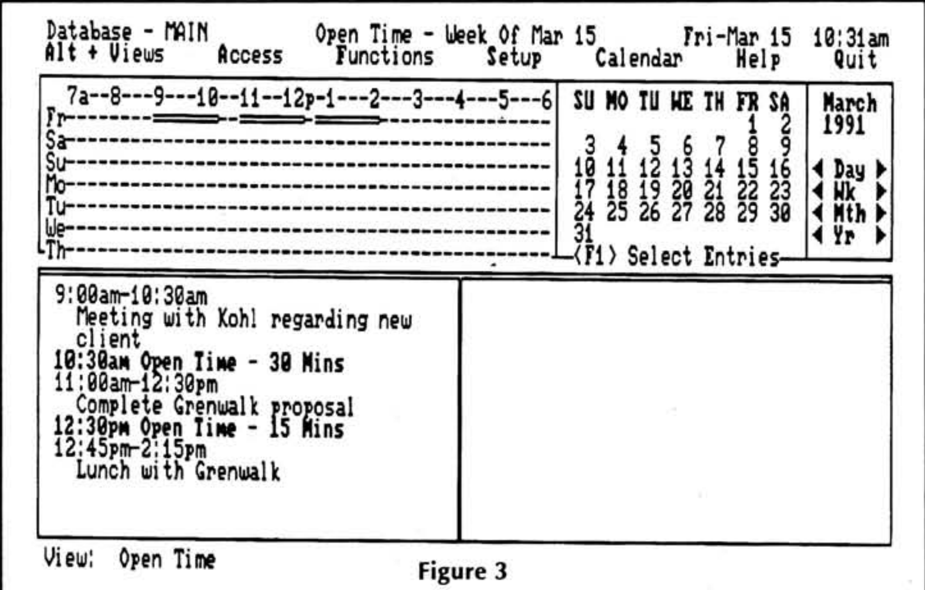
The clip from screen option allows you to copy displayed text to the clipboard's memory. You move the cursor to one end of the block you want to copy, press Alt-M to mark the start, move the cursor to the end of the block, and press Enter to save the text the clipboard. You can select either line or rectangle modes. You can only mark text if it appears on the screen, you cannot scroll the selection. If you want to transfer more text than appears, use the Report function.

Append to clipboard operates the same, except it adds the marked text to any existing contents.

Paste from clipboard inserts the contents of the clipboard at the location of the cursor.

File to clipboard inserts the contents of a disk file into the clipboard.

Write clipboard to file saves the clipboard to an ASCII text file.


```
Database - MAIN          Open Time - Week Of Mar 15        Fri-Mar 15  10:31am
Alt + Views     Access     Functions     Setup     Calendar     Help     Quit
  7a--8---9---10--11--12p-1---2---3---4---5---6  SU MO TU WE TH FR SA   March
Fr-------═════--═════-═════-------------------                   1  2   1991
Sa--------------------------------------------    3  4  5  6  7  8  9
Su--------------------------------------------   10 11 12 13 14 15 16  ◄ Day ►
Mo--------------------------------------------   17 18 19 20 21 22 23  ◄ Wk  ►
Tu--------------------------------------------   24 25 26 27 28 29 30  ◄ Mth ►
We--------------------------------------------   31                    ◄ Yr  ►
└Th-------------------------------------------└<F1> Select Entries─
 9:00am-10:30am
   Meeting with Kohl regarding new
   client
 10:30am Open Time - 30 Mins
 11:00am-12:30pm
   Complete Grenwalk proposal
 12:30pm Open Time - 15 Mins
 12:45pm-2:15pm
   Lunch with Grenwalk

View:  Open Time
```
Figure 3

Use the label function and the clipboard to transfer all of the addresses into WordPerfect for a mass mailing. In the people view, press L F2 F2 to transfer all of the addresses via the clipboard. Paste the clipboard into WordPerfect, then create and use a macro that adds the necessary merge codes.

Password Protection

When you install Instant Recall, the default database called Main is automatically created. You can store all of your information in Main, or you can create additional databases.If you store information in a file other then MAIN, you have to tell Instant Recall to use that database before you can access the data.

Databases are independent from each other, so you can password protect sensitive information in one file, while leaving another open for general access. Once you add a password to a file, via the Setup option, Instant Recall requests the password before a database can be accessed.

Unlike some database programs, which allow various levels of password protection, Instant Recall's locks the entire database. A program called DECODE is supplied that will reset, or remove, the password. You do not need to know the password to remove it with DECODE, it is mainly a safety feature in the event you forget it.

Anyone with access to the DECODE program can remove the password. But since DECODE is not copied to the working disk when you install Instant Recall, the offender would need an original distribution disk. This means that your password won't be of any use if someone has a strong desire to get at your data. They just have to purchase their own copy of Instant Recall and run the DECODE program on their own disk.

Still, the password scheme offers some security against the casual user who might have access to your system.

Categories

If you want the convenience of storing data about various clients in one file, you can create categories.

You create categories using the Setup function to establish groups to which entities are related. Instead of using a separate file for each client, for instance, you can create a category for each and maintain all of your information on one database.

When you insert a person, task, appointment, or other element in the database, you assign it to one of your categories. By selecting the category option in the access menu, you can then limit the items displayed in any view to those associated with a specific category.

Categories are useful even when you create a separate database for each client. Use categories to separate items into the different projects for the same client, or for tracking types of billing activity. You might have distinct categories for recording phone conversations, on-site visits, off-site visits, and other types of activities which you bill at different rates or reporting intervals.

Text Entry

The notes view provides a word processing-like function for text entries up to 65,000 characters per note. Notes can be connected to other database entries through links with the person the note relates to (in the people view) and the category to which it is assigned.

Like all Instant Recall elements, the note can also include an advance notice and alarm time entry. These are used to alert you that an activity or key time is approaching.

You can display notes using any of the methods on the access pull down menu, but the key phrase feature is particularly useful. Key phrase access lets you select notes, or any other element, by searching for a text string. The search string can even include the and, or, and not boolean operators.

You can enter text directly into the note view, or import it from a word processing file via the clipboard — just make sure the file is in ASCII format. To import a file, display the Note Add screen, then press Shift-Ctrl C to display the clipboard options. Select File to clipboard, and enter the file name that you want to copy into the clipboard. Then, press Shift-Ctrl-C again and select Paste from clipboard to insert the text into the note.

To transfer a note to a word processor, select the note in note view and use the Report function to write the note to a file or keyboard.

Tracking Time

Now that you've have an overview of Instant Recall, let's take a closer look at using it to keep track of your time for billing. We'll do this by looking at Instant Recall's quick timer view, which allows you to maintain information about time spent on an activity.

Rather than discuss quick timer function by function, let's see how it can be used.

Suppose you're working at your computer and a client calls on the phone for information. You would press Shift-Ctrl to pop up Instant Recall, then A Q, for Add Quick timer, to display the screen shown in Figure 5. Notice that the current date and time appear at the Start prompt.

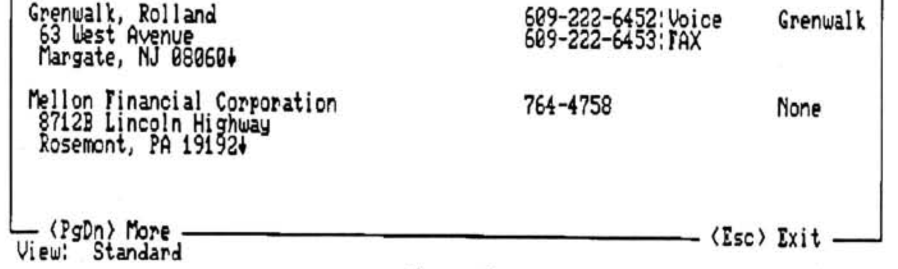
As you speak, you can enter a description or jot down notes about the client and the purpose of the call. When you hang up, press Ctrl-Enter to record the description and automatically insert the length of the call at the Elapsed Time prompt. With the call logged into the database, you can easily recall it a the end of the month to include the time in your invoice.

If you want, you can manually change the start date and time, and the elapsed time. This way you can record into the database periods of activity that you spent away from the computer.

The other prompts in the view provide for optional functions. For example, use alarm time to limit the time you spend on the current activity. After entering the description, insert the number of minutes or hours you want to spend on the activity at the alarm time prompt. Install Recall will beep and display a warning message when the time expires. I use this often when making long distance phone calls, and for limiting the amount of time I spend on non-productive calls. The beep sounding in the background provides a perfect excuse to get off of the phone.

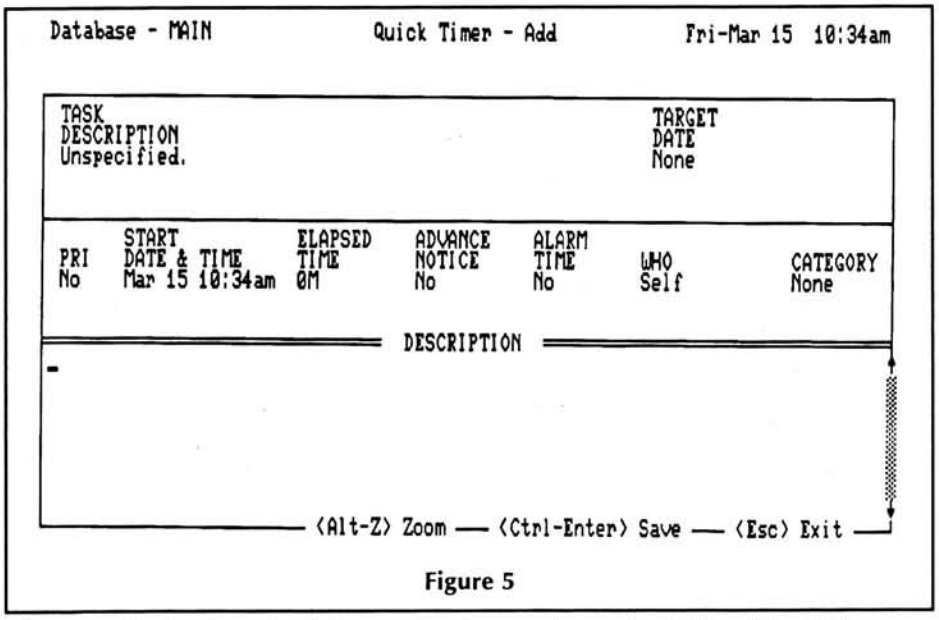
Pop Up Timer

Of course, if you are very busy, you might not have time to complete the quick



```
Database - MAIN                          People                Fri-Mar 15  10:34am
Alt + Views      Access      Functions      Help      Quit
 Name                                         Telephone              Category
→ Chesin, Adam                                55-1876:Voice          Chesin
   283 Lockhard Road                          555-1762:FAX
   Philadelphia, PA 19111↓

 Goldsmith, Gail                              None                   None
   3875 34th Street
   Ocean City, NJ 08176

 Grenwalk, Rolland                            609-222-6452:Voice     Grenwalk
   63 West Avenue                             609-222-6453:FAX
   Margate, NJ 08060↓

 Mellon Financial Corporation                 764-4758               None
   8712B Lincoln Highway
   Rosemont, PA 19192↓

 <PgDn> More                                                   <Esc> Exit
View:  Standard
```

Figure 4



```
Database - MAIN              Quick Timer - Add           Fri-Mar 15  10:34am

TASK                                                     TARGET
DESCRIPTION                                              DATE
Unspecified.                                             None

     START                ELAPSED  ADVANCE  ALARM
PRI  DATE & TIME          TIME     NOTICE   TIME    WHO        CATEGORY
No   Mar 15 10:34am       0M       No       No      Self       None

                              DESCRIPTION
-

              <Alt-Z> Zoom — <Ctrl-Enter> Save — <Esc> Exit
```

Figure 5

when you find yourself quickly moving from one project to another. These times are perfect for Instant Recall's pop up timer function.

The popup timer is a special form of quick timer entry that you can initiate without popping up the full Instant Recall screen. When you start an activity, press Shift-Ctrl-A and enter a brief description of the activity, such as a phone call or drop-in visitor. When you press Enter, the pop-up timer begins recording the time spent on that activity. When you're done, press Shift-Ctrl S to stop the timer for that activity.

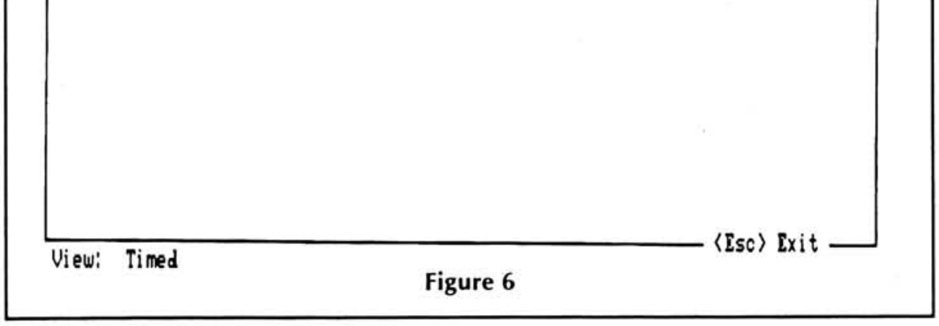
You can begin tracking one activity while one is being timed by starting the pop up timer again. Any timers running will be suspended. To enter details on the activity, switch back and forth between activities, or time more than one simultaneously, you have to pop-up Instant Recall and enter the quick timer view.

Figure 6 shows the quick timer view with a number of activities listed. The ones marked with an asterisk are suspended. The others are active, or currently being timed. To stop or start an activity's timer, select it on the screen and press S, then Enter. You can suspend only one activity from within an application using the pop-up timer. This is the Hot Entry marked with the triangle in the right margin of the quick timer view. To change the hot entry, select the activity on the view and press H then Enter.

When I'm ready to prepare an invoice, I use the Report function to save the quick timer entries to a disk file. This gives me an ASCII file with the description, start time, and elapsed time for each activity devoted to a client or project. The file becomes the detail lines of the invoice, and I add the address using the Label function.

As I mentioned previously, there are programs devoted to client billing. These provide greater flexibility and capability for preparing invoices and tracking time. Instant Recall does not include mathematical functions for computing the total of elapsed time or multiplying time by hourly rates. I usually use a separate pop-up calculator, like Sidekick, to calculate the totals after I've formatted the invoice. It would be nice, however, if Chronologic considered adding a feature for summing elapsed time by category for quick time entries.

The great advantage of using a PIM such as Instant Recall, is that it is not limited to a client billing view. You can instantly pop-up a global picture of activities, tasks, people, and notes, by topic, view, or date. You can see how the different types of information relate to each other, making it easier to manage, organize, and evaluate your interests. ✻



```
Database - MAIN                  Quick Timer            Fri-Mar 15  10:38am
Alt + Views     Access     Functions     Help     Quit

  Pri  Description                        Start            Elapsed  Category
→ No   Call from Chesin                   Mar 15  10:32am  6M       None     ◄
  No   Respond to Roland                  Mar 15  10:30am  *2M      None
  No   Call from Smithson                 Mar 15  9:20am   *17M     None

                                                          <Esc> Exit
View:  Timed
```

Figure 6

Continued from Page 5

board until you have placed it in its conductive bag. If you do not have a conductive bag, before you touch the board to another object, hold the board in one hand and touch the object with your other hand. This will equalize the electric potential between yourself, the board, and the object.

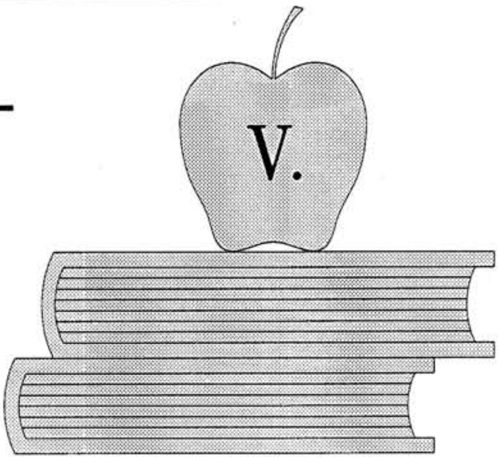
There is one more area which you should be aware of and use extreme caution. When removing the main board, take note of the lithium battery. A fire, explosion and severe burn hazard exists from the lithium battery on the main board. Do not heat the battery above 73 degrees Celcius (162 degrees Farenheit) or it may leak or explode. Do not short circuit, charge, over-discharge, disassemble, crush, penetrate or incinerate the battery.

Other than those two areas which require caution, disassembly is relatively easy. Once you have completed this part of the upgrade, you can now begin reassembly by simply replacing the boards with those in the upgrade kit. Don't forget the power supply, lock assembly and drives - they have sort of an important function!

A reliable source within Zenith Data Systems has informed me that they are working on a similar kit for government customers. It does not include a computer chassis. Watch for it in the Bargain Centre!

I do have one more tip: You will want to assure that your software installation is taking full advantage of the 386 computing power and the amount of memory you have installed. To accomplish this, refer to your software manual. ✻

Introduction to C++
Fifth Installment



Lynwood H. Wilson
2160 James Canyon
Boulder, CO 80302

Introduction

As a free lance I get to work on other people's code a lot. I don't hate it. I know people who do, who would rather be beat with a stick, but I don't mind. It's one of the best and most widely overlooked ways of learning. Who tries to write in any human language without first reading a good deal of other people's writing? And yet most programmers rarely read anyone else's code. A lot of them don't even like to read their own.

It's hard. Other people use different conventions, different formats and different choices of variable and function names. What's more, some other programmers aren't as good as we are and their work is hard to read because it doesn't make much sense. And no one puts in enough comments, or comments on the things we really need more information about. And in every one of those ways I was another person a year ago.

You learn a lot about what not to do from reading other people's code. And occasionally you find a gem, a perfect little bit of code that is so clean and neat that you learn something important about what to do.

In the past, I read code looking for such titbits. Perfect little functions such as you find in *The C Programming Language* by Kernighan and Ritchie. Lately I find I am looking more at the structure and organization. I think this is caused by having to actually do something with the code, as opposed to reading a program in a magazine. After all, the toughest test for the structure and organization of a program is for someone who doesn't know the program to try to fix a serious problem or make a major modification. The thing I find I most appreciate in this situation is not having to read every line of the code to figure out how the program works.

Structured programming is usually presented as a better way to write programs, a way to help the programmer deal with the complexity of the problem he is trying to solve. And down at the end it says "It makes the program easier to maintain too." I think this emphasis is backward. Perhaps because I have been doing maintenance lately I have been more aware of the problems.

Maintenance (including modification for new capabilities) takes about two thirds of the effort spent over the life cycle of a major program. The other one third covers specification, design, programming, module test, integration, integration test, documentation, and everything else. This may seem impossible to those of you who haven't been involved in a major project, but those who have agree that it is true.

The particular attribute of structured programming which seems to help the most in maintenance is the ability to read the program at a high level without reading all the code. Each function should be designed and written so that its operation and the flow of the related data are easy to follow and you don't have to read all the code in every function it calls to understand it. This allows you to work at the level of the problem you are trying to solve without having to understand all other levels, higher and lower. It is a way of managing complexity which allows you to focus on your work.

This is the core problem in computer programming, and it always has been ever since the guys used to walk up and down the passages inside the machines with shopping carts full of vacuum tubes (remember those big glass transistors?) trying to replace them as fast as they burned out. As soon as the hardware improved to the point that there was a fair chance of running a 10 line program to completion someone immediately wrote a 20 line program. The complexity of the software has been growing just slightly faster than the tools we use to make sense of it ever since.

The problem is the programmers. The machines have no difficulty with the most complex programs, as they only deal with one tiny machine language instruction at a time. The computer does not "know" anything about the structure or the complexity or even the size of a program it is running. It only knows about the particular instruction it is performing at the moment. Programmers, on the other hand, must be concerned with the whole program at once. And there is the problem. A large program is too complex to understand all at once.

From one perspective, C++ is C with extensions to better handle complexity. To help you to understand what you need to understand and to ignore what you can safely ignore. And from that same perspective, most of the great advances in programming have had that same goal.

More Functions

Most of the structure of the functions we covered last time would work as well under C as under C++. This lesson will cover some of the things C++ added to plain C functions. First a bit of program organization which is common to C and C++.

All programs must have a function called main. When the program is executed, it begins with the first line of the function called main. When the last line of main has been executed, the program ends. Other than those two rules, all functions are equal and any function may call any other.

A function may even call itself. This is called recursion, and I will give an example or two later.

Default Parameters

In C++ you may have a function which will automatically supply a default value for a parameter if the function is called with no value for that parameter. See Figure 1 for a simple example.

This is one of many useful functions for moving the cursor around the screen. Friendly user interfaces require that you do better than just printing lines of text to the bottom of the screen and letting them scroll off the top. This is of course just a

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

void cur_right(int distance = 1);


main()
{
   cout << "Move cursor 5 spaces right.";
   cur_right(5);
   getch();
   cout << "\nMove cursor 1 space right.";
   cur_right();
   getch();
}


void cur_right(int distance)
{
   gotoxy(wherex() + distance, wherey());
}
```

Figure 1

start.

Several of the library functions used here are new. The functions wherex() and wherey() return integers representing the x and y coordinates of the cursor. Gotoxy(x, y) moves the cursor to the coordinates represented by the arguments x and y. These are library functions from the Turbo C++ library.

There is a long standing element of confusion about how locations on the screen are described. Sometimes the upper left corner position is 1,1 and sometimes it is 0,0. In the past programmers have usually thought of it as 0,0 (0 is after all the first number) and users have thought of it (when they thought of such things at all) as 1,1. Borland has broken with this convention, and called the upper left corner 1,1. No problem, as long as you know.

When this program is executed, the main function is called first and it prints the first line of text and calls cur_right() with the argument 5 which moves the cursor 5 spaces to the right and returns. Main then calls the function getch() which waits for a key stroke. When you hit any key, getch() returns with the ASCII value of the key (which is not used) and the second line of text is sent to the screen and cur_right() is called with no argument. Cur_right therefore gets 1, the default argument, and moves the cursor one space to the right. Again the program calls getch(), waits for a keystroke, and then ends.

Note that the default argument is supplied when the function is declared (prototyped) at the head of the program rather than when the function is defined. This means that you can supply default arguments for functions you do not have the source code for, such as library functions.

This ability should be used with some discretion, to avoid excessive confusion to someone who is used to using the library function in its normal way.

In the case of a function which takes several arguments you may supply defaults for more than one of them, but the default arguments must go at the end of the argument list, after any arguments which do not have defaults.

```
some_func(int a, int b = 1, int c = 2);          // OK
some_func(int a = 1, int b, int c = 2);          // WRONG
```

This makes sense if you consider that the compiler must be able to tell which parameters are missing in order to assign default values to them. If you call this function with 2 arguments, the compiler assumes that c is missing and supplies the default value 2. If you call it with 1 parameter, the compiler assumes that b and c are missing and supplies the defaults. If you call it with no arguments, the compiler decides you made a mistake and supplies an error message. (Because there is no default value for the parameter b.) There is no way that you can call the function with value for argument b and none for a or c.

The ability to supply default arguments for a function is particularly useful when you have decided to modify a function to increase its capability. You can add one or more new arguments to the function call and with the use of defaults you do not have to go back and modify every existing call to that function in your code. You can use defaults to ensure that the function works just as before if called just as before.

Many programmers would not write a function like cur_right() with only a single line of code for its body, but would just put the single line of code in their program wherever they needed it. I think that in many such cases a function with a well chosen name can offer better readability.

I am not, however, convinced that cur_right() offers a clearer picture of what it does than cur_right(1), but it makes a good demonstration of default arguments.

Inline Functions

In a case where you would like the readability of a function call without the overhead, C programmers have traditionally used a preprocessor macro. However, macros lack data type checking and can be the cause of some elusive bugs. C++ offers the solution in the form of inline functions.

The inline function code is substituted directly wherever the function is called, rather than being stored somewhere and jumped to like a normal function. This means that the overhead of passing control to a function and back again after it ends is saved.

The definition of an inline function is the same as its declaration. (Actually, the terminology gets a little shaky here.) At the head of a file where you would expect prototypes (declarations) of functions defined elsewhere you may also have definitions of any inline functions. These functions do not have any other existence, no definition elsewhere, so their bodies must be here, unlike other functions.

Since all the code of the function occurs here, we call it the definition of the function rather than the declaration. An inline function does not have anything which would reasonably be called a declaration, or a prototype.

See Figure 2 for our example function, in inline form.

The availability of inline functions should overcome any reluctance to write short functions, and for this and other reasons we will see more short functions in C++ than in most other languages.

Inline functions must be defined in all diskfiles which use them, and so if you plan to use one over several files it makes sense to put it in a header file which is #included in each of the files which uses the function.

Overloaded Functions

When two or more functions have the same name but take different argument lists, they are called overloaded functions. We have seen a similar situation in the arithmetic operators. The multiplication operator, for example, does a slightly different job depending on the types of the data it is operating on. Floating point multiplication is different in the details of the operation from integer multiplication, but conceptually they are doing the same job.

The argument list can vary in number or type of arguments.

Figure 3 shows a simple example of overloaded functions.

When main() calls oprint() with an integer argument the first oprint() which was defined with an integer argument is called. When main() calls oprint with a character argument, the second oprint(), defined with a character argument is called.

Note that the two functions each have their own declarations at the beginning of the program, and their own separate definitions. They are two separate functions which just happen to have the same name.

In earlier versions of C++ (earlier than

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

inline void cur_right(int dist = 1)
      {gotoxy(wherex() + dist, wherey());}


main()
{
   cout << "Move cursor 5 spaces right.";
   cur_right(5);
   getch();

   cout << "\nMove cursor 1 space right.";
   cur_right();
   getch();
}
```

Figure 2

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

void oprint(int x);
void oprint(char ch);


main()
{
   oprint('Q');
   oprint(79);
}


void oprint(int x)
{
   cout << "\nThe int is " << x << '.';
}


void oprint(char ch)
{
   cout << "\nThe char is " << ch << '.';
}
```

Figure 3

2.0) the keyword "overload" was used to tell the compiler what was going on. It is no longer necessary.

If there is not a match between the argument list being passed by the calling code and one of the versions of the function, the compiler will try the standard type conversions which we discussed along with mathematical operators. This can turn into a pretty complex business. At this level of experience you should ensure that there is a function to match your argument list and avoid the problem.

This idea seems simpler in my mind if I think of them as different functions with different names. Think of them as having the argument list added to their names, so that the first one would be named oprint(int) and the second is named oprint(char). The compiler actually does something very similar to this when it compiles the code, so it can keep them straight. If it works for the compiler it will work for us.

Global Variables

It is possible to get data in and out of functions without explicitly passing it. A variable declared outside of the functions of a program is accessible to all functions in the program. Figure 4 is an example.

This simplifies the program by eliminating the need to pass the data around, but it is not as good a deal as it might seem. For one thing if the variable ch turns up with bad data in it you have no idea where to look for the culprit since every line of code in the program has access to ch. More importantly, you cannot follow the flow of the data through the program without reading every line of code (and having an exceptionally good memory).

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

void print();

char ch;      // declaration of global variable


main()
{
cin >> ch;
print();
}


void print()
{
cout << "\nThe char is " << ch << '.';
}
```

Figure 4

A lot of the ideas and concepts of good programming style in general and object oriented programming in particular seem unnecessary, even counter-productive, when the program in question is an example 25 lines long. Large programs, 10,000 lines or so, are different from small programs and have different problems that are not apparent until you have struggled with them.

Global variables can be useful but I try to avoid them as much as possible. There are cases where there is just no other reasonable way to solve a problem, but not very many.

Preprocessor Directives

I have mentioned preprocessor macros and #define without properly describing them, so here they are.

Before the compiler gets your code it is run through another program called the preprocessor. In many systems it isn't really a separate program but just a part of the compiler. In any case, we will think of it as separate since the job it does is independent of the compiler. In most cases where it is part of the compiler, you can set a compiler flag so that you can save the output from the precompiler before it is compiled, just to see what it has done.

We have been using the preprocessor in all our programs to copy header files into our code with the #include command. I discussed this briefly in the first installment.

The preprocessor responds to any line which begins with #. These statements are called sometimes compiler directives, but I will continue to call them preprocessor directives since I think that helps to make it clear that the operations they call for occur before the compilation of the program begins.

The #define Directive

The #define preprocessor directive causes the preprocessor to perform a simple text substitution. The first group of characters (up to the first space) after the #define is replaced throughout the file with the rest of the characters on the line. Figure 5 is an example.

```
#include <iostream.h>


#define PI 3.14


main()
{
   float radius;


   cout << "Enter the radius: ";
   cin >> radius;
   cout << "\nThe area is " << radius * radius * PI << '.';
}
```

Figure 5

The preprocessor searches the file for the word PI and every time it finds an instance of it the rest of the line is substituted, in this case 3.14. The result of this substitution is that the calculation in the above program is changed from radius * radius * PI to radius * radius * 3.14.

This is a very powerful and useful tool. Suppose you wanted to change the value of pi to 3.141592653 in a search for greater precision. You need only change it in one place, rather than hunting through the whole program.

Another use for the #define is found in the endless fight against magic numbers, numbers which do some magic thing that is not apparent when reading the program. For example, if you wished to determine whether a particular character was a carriage return you might write a line of code like this:

```
if(ch = 13)
```

And if you wanted to be particularly kind to anyone else who had to read it you might include a comment such as:

```
// 13 is RETURN
```

but it would be a lot clearer if you wrote:

```
if(ch = RETURN)
```

which you could do if only you wrote this line at the head of the program:

```
#define RETURN 13
```

The main reason for this is readability, but it is also useful if you should ever port the program to another environment in which the carriage return is not 13. You have only one change to make.

Note that by convention and old habit, words to be the subject of a #define are usually written in all capital letters. This is not required, but helps tell them from variables.

Constants

There is another way to achieve the same purpose in C++ and that is the constant. Here is the same program using a constant in place of the #define.

```
#include <iostream.h>

const float PI = 3.14;


main()
{
   float radius;

   cout << "Enter the radius: ";
   cin >> radius;
   cout << "\nThe area is " << radius * radius * PI << '.';
}
```

Figure 6

```
#else, used after #ifdef just as else is used
after if.

#elif: used like else if.

#ifndef: means "if not defined", works like
#ifdef.

#undef: which undefines the word following it.
```

Figure 7

Constants are not always written in upper case like defines are, but I think it is a good practice for those of us who are used to the idea from C. I think it makes things a bit clearer.

The word const can be used before the data type in any declaration. Also, a const has scope just like any other
data object, but this is done a bit differently. A const declared outside all functions, such as the one above, has file scope, that is its scope is the disk file within which it is declared. If you want to reference this const in another file you must define it like this:

`extern const float PI = 3.14;`

and in the other files that use it, declare it like this:

`extern const PI;`

This use of the constant differs from that of C. This is one of the areas in which C++ is not a proper superset of C. These changes are intended to encourage the use of constants instead of defines.

The const MUST be given a value when it is declared since it cannot be given a value anywhere else. To attempt to change the value of a const is a compile time error, as is an attempt to declare one without initializing it.

Note that the disadvantages of global variables do not pertain to global constants. There is no problem with the value being changed by a function which shouldn't and they do not create confusion about the flow of data between functions since they don't change. They represent data which is intended to be available everywhere, but whose flow you are not interested in following.

This technique has the advantage that the compiler can do the usual data type checking on a constant, but not on a #define. Also a symbolic debugger will know about a constant but will not know about a #define, since the text substitution has occurred before the symbol table is constructed.

#define Macros

The preprocessor #define can also be used to write macros, which are like small functions without the function call overhead. If that reminds you of something, you have been paying attention. Actually, C++ inline functions were intended to save you from the problems of macros. As a result macros are seldom used in C++ except by old C programmers who haven't completed the transition. We will have a quick look at one, in case you work with old C programmers.

`#define mult(x, y)  x * y`

In this case x and y are arguments just like the arguments of a function. If the macro is used in a program like this:

`z = mult(7, 4);`

it will be expanded by the preprocessor to this:

`z = 7 * 4;`

and the value 28 will be assigned to z. The problems occur when the call is more complex. For example, what about this?

`z = mult(7 + 2, 3);`

You have every right to expect z to get 27, but the macro will be expanded to:

`z = 7 + 2 * 3;`

which assigns 28 to z. This problem can easily be solved by putting parenthesis around the arguments in the #define like this:

`#define mult(x, y)  (x) * (y)`

which expands to

`z = (7 + 2) * (3);`

but what if you write a macro like this one:

`#define sq(x) (x) * (x)`

and call it like this:

```
y = 3;
z = sq(y++);
```

It will be expanded to

`z = (y++) * (y++)`

which will assign 12 to z and leave y with a value of 5, not at all what you would expect. So although the macro capability is carried over into C++ for compatibility with old C, you should use inline functions instead because of unexpected problems like these.

`#ifdef`

The last preprocessor commands we will deal with are a set of commands for controlling which portions of the code are to be compiled.

These are commonly used with test code, portions of the program which are put in to help in test and debugging and are not intended to be part of the final product. It is done like this:

Part of the program......

```
#ifdef TEST

       test
       code

#endif
```

the rest of the program......

If TEST is defined when the code is compiled, then the test code is compiled too. If TEST is not defined, the code between the #ifdef TEST and the #endif is ignored.

You can cause TEST to be defined by adding this statement to the head of the program:

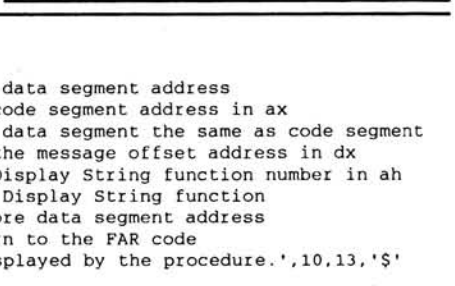
`#define TEST`

which defines TEST as NULL since you didn't say what you wanted it defined as. That activates the #ifdef, since it doesn't care what TEST is defined as, only whether it is defined at all.

There is a compiler flag in Turbo C++ to define a symbol. If you place -DTEST in the line which invokes the compiler, TEST will be defined. This allows you to control the conditional compilation without actually changing your code.

There are other directives in this family, and they are shown in Figure 7.

Check your manuals for examples of these less common commands, and don't forget to write a few programs between now and next month. ✻



Continued from Page 19

```
        over3   PROC      FAR
        push    ds                    ;Save data segment address
        mov     as, cs                ;Put code segment address in ax
        mov     ds, ax                ;Make data segment the same as code segment
        lea     dx, messa             ;Put the message offset address in dx
        mov     ah, 09h               ;Put Display String function number in ah
        int     21h                   ;Call Display String function
        pop     ds                    ;Restore data segment address
        ret                           ;Return to the FAR code
messa   DB      '***This message is displayed by the procedure.',10,13,'$'
over3   ENDP
code    ENDS
        END
```

✻

dBASE III
Part 12

D. R. Cool
7421 Troy Manor Road
Huber Heights, OH 45424

The LOAD and CALL Commands

The LOAD and CALL commands are virtually ignored in most texts on dBASE III. Although the dBASE III manual does give an example of a program that changes the cursor, it gives absolutely no information on how to manipulate data that has been passed to the program as arguments using the CALL command.

One question may come to mind - if you can run any external program using the RUN command, why would you need LOAD and CALL? One answer is speed. The RUN command requires loading a program from disk each time it is invoked. If you wanted to paint screens, for example, direct to video memory using assembly language, using the RUN command would cancel out any speed advantage. Using the LOAD command, a program is loaded into memory once, usually at the beginning of a program. It can then be executed instantly at any time using the CALL command.

When I first attempted to use these commands, I almost gave them up as a lost cause. I would LOAD my program then try to execute it with the CALL command only to have the system respond with "Program not loaded". Yet the CURSOR program given as an example in the dBASE III manual worked fine. Finally, I discovered the problem. It seems that dBASE III has a bug that prevents it from loading any program with eight characters in the basic file name. To illustrate, take any file and copy it to TESTPROG.BIN. Now, enter dBASE and try to load this file with "LOAD TESTPROG". Now try to run the program with "CALL TESTPROG". dBASE should respond with "File was not LOADed." Now Press F6 and continue to the end where it says "Modules loaded:" You should see "TESTPROG" followed by two

Listing 1

```
; DRAWSCR.ASM
; Program to draw main menu
; To be used with LOAD and CALL commands of dBASE III
; Written by:  D.COOL
; Assemble using A86.  Rename resultant .OBJ file to DRAWSCR.BIN.


        org 0000h                  ;zero offset (required by dBASE III)
 jmp    start                      ;jump over data

; data area
scrn_color  db ?                   ;store attribute here
screen1     db 80 dup(32)          ;start of menu text
db '                           +-----------+                                  '
db '                           | MPCAG SYSTEM THREE |                                   '
db '                           +-----------+                                  '
db '   +--------------------------------------+   '
db '   |__+---------+_____________________+---------+________________|   '
db '   |_+|     SEARCH      |--------+___+|   ASSISTANCE    |-------+_|   '
db '   |_|+---------+             |___|+---------+             |_|   '
db '   |_|      1. By vendor PN            |___|   A. The Assistant           |_|   '
db '   |_|      2. By SMD PN               |___|   D. DOS Commands            |_|   '
db '   |_|      3. By M38510 PN            |___|   H. Help                    |_|   '
db '   |_|      4. Project info            |___|   L. Lock Computer           |_|   '
db '   |_|      5. Reg. Users              |___|                              |_|   '
db '   |_|      6. Special                 |___|                              |_|   '
db '   |_+----------------+___+----------------+_|   '
db '   |__+---------+_____________________+---------+________________|   '
db '   |_+|   MAINTENANCE   |--------+___+|  INFORMATION    |-------+_|   '
db '   |_|+---------+             |___|+---------+             |_|   '
db '   |_|      B. Backup                  |___|        DATE          TIME    |_|   '
db '   |_|      T. File transfer           |___|                              |_|   '
db '   |_|      U. Data Base Updates       |___|                              |_|   '
db '   |_|                                 |___|   LAST UPDATE:               |_|   '
db '   |_+----------------+___+----------------+_|   '
db '   |____________________ Enter selection or X to quit: ____________________|   '
db '   +--------------------------------------+   '

;code:
start:  mov al,[bx]                ;get screen attribute from command line
        push ax                    ;save temporarily on stack
        mov ax,cs                  ;set DS to CS
        mov ds,ax
        pop ax                     ;retrieve attribute and store
        mov [scrn_color],al
        mov ax,0b800h              ;set destination to screen
        mov es,ax

; Write attribute first:
        mov di,0001h               ;set destination pointer to address of
                                   ;first attribute
        mov cx,2000                ;set counter to 2000 bytes
        mov al,[scrn_color]        ;get attribute
wrt_attr:   mov es:byte ptr di,al
                                   ;segment override required since ES
                                   ;is the segment wanted
        inc di                     ;skip character
        inc di
        loop wrt_attr              ;do until counter = 0
```

Listing 1 (Cont'd.)

```
; Write text:
        lea si,screen1          ;set source pointer to screen text
        lea di,0000h            ;set dest pointer to start of screen
        cld
        mov cx,2000             ;set counter to 2000 bytes
wrt_txt: movsb                  ;write ASCII code to video memory
        inc di                  ;skip attribute
        jnz wrt_txt             ;do until counter = 0
        retf                    ;exit to dBASE
```

additional characters. Use a file name seven characters or less and no problem. (This bug was corrected in dBASE III Plus.)

There is one thing in particular you should pay attention to if you use the LOAD and CALL commands in any of your programs. If you have data contained within the program, you must set the Data Segment (DS) register to the data segment (or code segment if data and code are in the same segment) before attempting to manipulate the data. This is because dBASE III initially sets the DS register to point to the segment containing any memory variables passed with the CALL command.

Two other points: The program must start at zero offset instead of the usual 0100H, so start the program with the statement "ORG 0000H". Second, the program must end with a FAR return instead of the usual DOS exit.

Listing 1 is the assembly listing for a routine which paints a main menu by writing ASCII code directly to video memory. Note that the screen text is contained within the program rather that writing it from a file, which would defeat the purpose of the program - namely, a very fast screen display. Note the first statement

OK. give me the story one more time, you're reading a borrowed REMark?

"org 0000h" as required by dBASE III. The next statement - "jmp start" - jumps over the data area to the code.

The data area contains a series of data statements starting with the label "screen1". The first statement is simply a series of 80 spaces, since the first screen row of this menu is blank. The next lines contain the actual menu characters. To construct your own screen, simply type characters exactly the way you want your screen to appear including any extended graphics characters. (Generation of graphics characters was explained in my last article.) When you have constructed your screen display, put your word processor or editor into INSERT mode and insert "db" plus a space plus a single quote (') before each screen line. At the end of each screen line insert another single quote. Make certain each line contains exactly 80 characters or spaces between the single quotes. Also, include exactly 25 data lines even it contains nothing but spaces.

Next, we come to the code. To be flexible, since this program is used on both monochrome and color monitors, I pass the color attribute as a memory variable with the CALL command. Consequently, the first thing this program does is to get the attribute value and save it in the AL register. As stated in the dBASE III manual, the BX register points to the first memory variable on the command line. The next command - push ax - saves the color attribute on the stack. (The program can't store it within the code segment yet because the DS register is not set to the proper value.) The next three instructions wet the DS register to the same value as the Code Segment. Next the program retrieves the attribute and stores it for later use.

The next two instructions set the ES register equal to the video memory segment at B800H. This address is where the 4000-byte video memory buffer that represents the first page of display memory starts. This memory is arranged in groups of 2-byte addresses. The even addresses contain the bytes that the hardware interprets and displays as characters. The odd addresses contain attribute bytes which determine the foreground and background colors of each character.

The next instruction sets the destination pointer (DI register) to the address of the first attribute: B800:0001. Next, the counter is set to 2000 bytes (80 columns by 25 rows) and the AL register is loaded with the attribute value. The next four instructions form a loop which writes the attribute value to 2000 locations in video memory. The final series of commands write the menu text to video memory using the MOVSB command. The program ends with the required FAR return.

This program was compiled using A86, a compiler that can be downloaded from the HUG bulletin board. If you use this compiler, take the resultant .OBJ file and rename it using the extension ".BIN". To use the program, you must first load it:

```
LOAD DRAWSCR
```

You can then call the program:

```
CALL DRAWSCR WITH CHR(attribute)
```

where attribute is any number from 0 to 255. If you have a color monitor, attribute is decoded as follows:

bits 0-3:

foreground color:

black	0	0	0	0
blue	0	0	0	1
green	0	0	1	0
cyan	0	0	1	1
red	0	1	0	0
magenta	0	1	0	1
brown	0	1	1	0
white	0	1	1	1
gray	1	0	0	0
lt blue	1	0	0	1
lt green	1	0	1	0
lt cyan	1	0	1	1
lt red	1	1	0	0
lt magenta	1	1	0	1
yellow	1	1	1	0
bright white	1	1	1	1

bits 4-6:

background color:

black	0	0	0
blue	0	0	1
green	0	1	0
cyan	0	1	1
red	1	0	0
magenta	1	0	1
brown	1	1	0
white	1	1	1

bit 7:

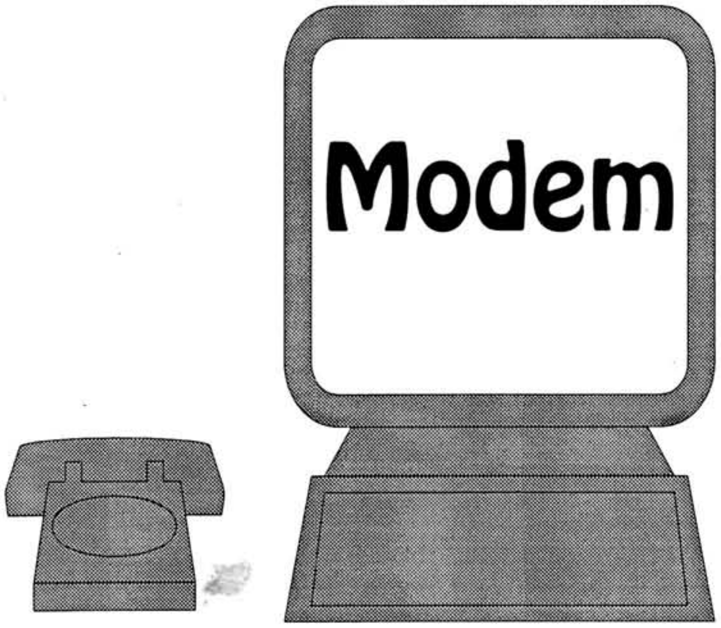
0 = normal

1 = blinking

Thus, an attribute value of 23 (17H) would produce white letters on a blue background.

Using the LOAD command, you can lôad up to five files up to 32,000 bytes each. To un-LOAD a file, use the RELEASE command followed by the file name. The extension is always assumed to be ".BIN".

This completes my series on dBASE III. Although I have not discussed every command, it was my intent to focus primarily on programming. I will be glad to answer any questions on this series or on dBASE III or dBASE III Plus (except for networking). You can write to me at the above address or put a message on the HUG BBS.



A Data Transfer Solution

Part 3
Data Compression

Robert C. Brenner
9282 Samantha Court
San Diego, CA 92129

To accurately transfer data at speeds above 2400 bps, the information must be modified to remove redundancy. Data compression is the electronic equivalent of removing the white space on a sheet of text. The bare (compressed) text is then transmitted and the white space is re-inserted (decompressed) at the receiving end. This allows more data capacity per unit of physical storage and lowers data transmission costs because more information is shared in a standard unit of time.

Today, text, spreadsheets, forms, graphics, still video, motion video, and voice are usually compressed before transmission. Different methods are used to cram as much data as possible into a finite range of time or frequency. Text data is squeezed using Run Length, Huffman, Shannon-Fano, Lempel-Ziv (LZ1, LZ2), and Lempel-Ziv-Welch (LZW) compression coding. For graphics and line art we use Huffman, Quantization, Text Compression, and Discrete Cosine Transform (DCT) coding. For facsimile we'll use DCT, Modified Huffman, and Relative Element Address Designate (READ) coding. For full motion video we'll use DCT, Color Cell Compression, or Digital Video Interactive. And voice incorporates a myriad of proprietary methods, most based on Adaptive Differential Pulse Code Modulation. Each of these compression algorithms is optimized for the type of information to be stored or transfered.

Data compression can be accomplished in software or hardware. Users of CP/M run SQ.COM to compress ASCII files by about 40 percent and USQ.COM to decompress the files back to their original. Unix users are familiar with utilities such as COMPRESS and COMPACT. In our MS-DOS world we have programs such as ARC from System Enhancement Associates, the freeware program LHARC from Haruyasu Yoshizaki, and Phillip Katz's shareware program PKZIP from PKWARE, Inc. These programs incorporate one or more compression algorithms. For example, SQ.COM uses Huffman coding. ARC, PKZIP and COMPRESS all use the LZW compression algorithm, and one stage of PKZIP uses a close relative of Huffman coding called Shannon-Fano coding.

Commercial programs have grown so large that some are now compressed and packaged with a decompression program included. CorelDRAW, for example, comes compressed from 14-disks to seven. It includes LHARC to expand the CorelDRAW program files during installation.

Files that have been compressed must be decompressed before use. Many bulletin board files that you download are compressed before transmission. At your end, you run a decompression program to restore the file to its expanded size before you can run the software.

Some files can compress much better than others. ASCII text, listings, spread sheets and forms with lots of repeated characters, character strings, or words can compress best. Binary files and .EXE files are usually less compressible.

The amount of compression that an algorithm achieves defines its compression ratio. It compares the uncompressed to the compressed file length. Compressions from 2:1 for data and 50:1 for graphics are common.

Compression that compacts data 2:1, but allows the exact original information to be recovered, is called lossless. Complex compression algorithms are used to transfer huge video image files at high compression ratios (50:1), but some loss in pixel information occurs. This lossy compression technique works fine for video because a few incorrect pixels have little effect on image quality (particularly if the image is full motion video). This article focuses on lossless compression because this is common in modem data transfers.

Huffman Coding

In the early 50s David Huffman designed a statistical data compression technique that replaces symbols such as 8-bit ASCII characters with codes of varying lengths. The file is compressed into fewer bits and these codes are then sent. At the other end a decompression algorithm based on the same technique replaces the code with the original data.

Frequently used symbols such as e, t, and a are assigned shorter codes (1- to 4-bits wide). Less frequent symbols such as j, q, and z can be represented as 8- to 16-bit codes. Three bits can represent the first four most common characters, eight bits can be used to represent the 5th through the 19th most common symbols and 16 bits can be used to represent infrequent symbols.

Several forms of Huffman coding exist. Static Huffman coding uses a conversion table of symbol occurrence probabilities compiled from statistical observations or generated by prescanning input data. As the data stream is read, an encoding tree is constructed starting with the least probable character combination (symbol). Each symbol represents a leaf on the tree. High probability symbols are located close to the root and assigned shorter codes than low priority symbols. Combinations of symbols (leaves) are combined as nodes and assigned a probability that represents the products of the probabilities of the leaves. Nodes are then combined to produce other nodes representing further combinations of symbol leaves. The code lengths are variable. Once the look-up table tree is constructed, compression can begin.

Static Huffman coding requires a table of probabilities for each type of data. When the symbol types aren't known, a Dynamic Huffman Coding technique is used to construct the tree on the fly and compress the data as it is being read. The Huffman tree changes dynamically with the changing symbols and symbol combinations so the occurrence probabilities vary constantly. The coded output is sent to the receiving modem where a Huffman decompression algorithm uses the same compression tree to convert the code and dynamically update the tree.

Huffman coding increases system overhead because the frequency distribution of symbols in the source data must first be determined to build the probability table. The symbol combinations (nodes) cannot change during transmission. Therefore, two passes are used at each end of the transmission link. The sending system examines the data (Pass 1) to build or update the table; then another pass is made to create the compressed output file. The receiving system reads the symbol substitution table sent with the compressed file to define the decoding rules and then expands the data as it comes in.

Compressed files are at risk because the non-linear code length used in Huffman coding implies that corruption of a single data bit in the compressed data stream can desynchronize the entire file producing gibberish during decode. The requirement for an end of compressed data file signal makes the data stream a few bytes longer. However, even with these limitations, Huffman coding has been a long time favorite. It is used as the last stage in the Joint Photographics Experts Group (JPEG) color image and Microcom Networking Protocol (MNP) Level 5 data compression protocols.

LEMPEL-ZIV

Another important lossless compression technique was described in the IEEE Transactions on Information Theory in May 1977 by A. Lempel and J. Ziv. Like Huffman, their Lempel-Ziv (LZ) compression algorithm looks for repeated characters or strings and substitutes a shorter code for the repeated sequence. The compression software scans text, graphics, or binary files and performs statistical analysis to create a code substitution table. The difference between LZ and Huffman is that LZ replaces frequently appearing symbol strings with fixed-length codes. Over 60 percent of the words in a text file can be replaced with these short codes. In Lempel-Ziv, a fixed-length 11-bit code is used to represent up to 32 compressed characters. Where Huffman is suitable for transfers under 9600 bps, Lempel-Ziv works well above 9,600 bps. At 9,600 bps, its a toss-up. Telebit's Trailblazer uses Lempel-Ziv to operate at an effective 19.2 kbps on dial-up lines normally limited to 12 kbps.

There are two primary implementations of Lempel-Ziv, LZ1 and LZ2. LZ1 compression is accomplished using a memory-resident 2K sliding dictionary window in which strings of data are coded and consecutively stored. Repeated strings are replaced by codes and kept in the look-ahead window. During decompression, the uncompressed strings and codes are read and the sliding window is searched for code matches. A match causes the code to be replaced by its appropriate string of data. The sliding window of LZ1 adapts quickly to a changing data stream. With no dictionary to manage, this technique imposes little processor overhead.

LZ1 has been implemented in firmware to perform group decompression in some hard disk drives and is used in quarter-inch-cartridge (QIC) tape and digital audio tape (DAT) drives. It's also implemented in software utilities such as SY-TOS Plus, PC Tools Deluxe, and Norton Backup.

A second Lempel-Ziv technique, LZ2, converts a unique string of data into a 4096-entry dictionary with a numeric value assigned to each string. Again, once the dictionary is constructed, repeated dictionary entries are transmitted as compressed data code words. LZ2 synchronizes both the compression and decompression functions during data transfers. The dictionary at the receiving end is filled by the sending modem and then decompression begins.

There are several derivatives of Lempel Ziv. The British derivative, British Telecom Lempel Ziv (BTLZ), uses a slightly different library for storing representations of compressed characters. The Hayes Computer Products derivative, Hayes British Telcom Lempel Ziv (HBTLZ), prevents random binary data from being inadvertently expanded by a compression algorithm.

In 1984, Terry Welch refined the Lempel-Ziv algorithm to replace strings of characters with single codes. This LZW method does no analysis of the incoming text. It simply adds every new string of characters to a table. Compression occurs when a single numerical code replaces a string. Although difficult to predict, the optimal dictionary matrix size has been established at 4096 codes. This means that both sending and receiving systems must include RAM for storing the LZW dictionary. In a 4096-entry dictionary, the first 256 codes usually represent ASCII characters and the remaining 3840 codes represent strings of characters.

Each codeword is between nine and 12 bits long. Of the 4096 logical entries in the dictionary matrix, the first 512 are nine bits wide; the second 512 are 10 bits wide, the next 1024 are 11 bits wide, and the final 2048 logical entries are 12 bits wide. This makes the actual 4096 entry dictionary 45,568 bits wide (slightly under 6K) minimum. To avoid hashing collisions that could slow the compression rate, the dictionary RAM is typically 8K to 16K.

As data is passed through the LZW algorithm, it is checked for strings. If a string is already in its table, it sends the code substitute. If not, it adds the string to the dictionary, assigns a code, and then sends it. The dictionary table builds rapidly, and compression starts once approximately 100 codes have been assigned.

At the receiving end, a decompression algorithm decodes the string and code components to produce an identical substitution table. The incoming data doesn't include a string translation table because the code itself contains the information necessary to reconstruct the dictionary. A single exception case involving repeated characters in the same word is automatically handled by the LZW algorithm. As described in an October 1989 Dr. Dobb's Journal article, compressing the string "/WED/WE/WEE/WEB/WET" causes the sequence "/ W E D 256 E 260 261 257 B 260 T" to be transmitted. The decompression algorithm decodes this back to the original data.

Re-creating the translation table, positions, and data sequence elements from the compressed information is one of LZW's strongest features. This makes data transfer time inherently shorter. To reduce string search and comparison time, strings are stored as code/character combinations. A hashing routine applies an XOR function to group strings of data by content and to store them in sequential dictionary locations. For example, all the strings starting with "6" are stored serially. Hash coding cuts the search (hence the translating) time. Hashing is not easy to implement, but it makes string matches possible in a single search. Extracting the data at the other end is typically fast and accurate.

LZW compression works well on English text (2:1 common) and fairly well on saved screen displays, but is less capable with data files. Expanding from 12 to 14- or 15-bit codes achieves better compression ratios on large files, but degrades performance on small files. This is why the data compression program ARC was written for variable length codes. ARC may use 9-bit code for one string and 10- or 12-bit code for another.

As long files are read, the LZW compression ratio tends to degrade, because once the finite size table is full, new strings are not encoded, but get sent without compression. Some strings at the beginning of a file may be seldom used again but retain a code substitute in the dictionary table. To counter this, some LZW applications monitor the compression ratio and periodically flush the table of strings that aren't frequently used. Dynamically rebuilding the dictionary greatly enhances the compression ratio, but it requires more executable program (hence operating RAM) space.

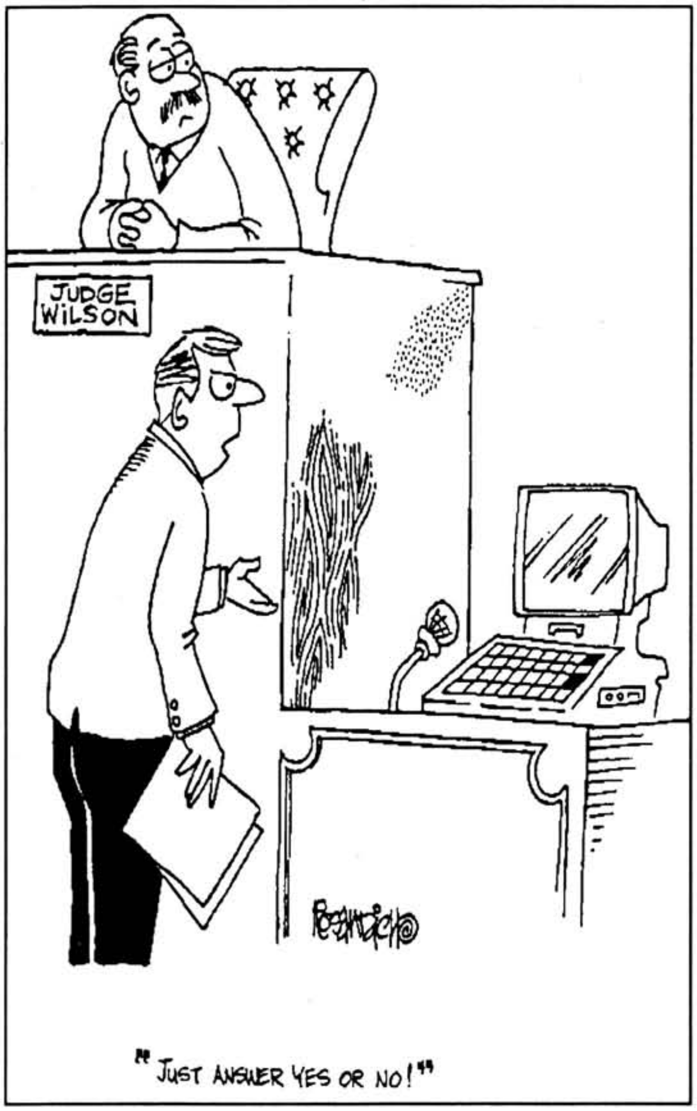
Some applications run LZW codes through adaptive Huffman code filters to gain a few more points to the compression ratio. Although efficient, this increases the complexity of the code and adds to the compress-transfer-decompress time.

LZW is the basis for the program PK-ZIP and Unix's COMPRESS utility. It's also used in CompuServe's GIF file format, ARC, and Stuffit. The LZW algorithm was patented by Sperry (now UniSys).

Data compression in software is inexpensive. If data files are compressed before modem action, high performance transfers can occur, but both ends of the transmission line must use the same compression/decompression algorithm. Then as long as the sending and receiving electronics can buffer and convert the data fast enough, system performance is optimized. Because high speed modems push the limits of current technology, maximum performance is achieved by compressing and decompressing data "on the fly." As transfer speeds increase, software cannot compress/decompress fast enough. This prompted companies like Rockwell to design integrated circuits (ICs) that can perform data compression and decompression in hardware. Some companies offer adapter boards that implement LZ2 data compression before modem transmission.

Derivations of LZ2 are appearing. Hewlett-Packard and Advanced Hardware Architectures, Inc. jointly proposed a variation of LZ2 called data compression Lempel-Ziv (DCLZ) as a standard for quarter-inch QIC tape drives. DCLZ uses hash-coding for dictionary lookup string matching. As we'll also see in the next article, standards and protocols such as Microcom Systems MNP5 use dynamic Huffman compression, MNP7 combines Huffman coding with a predictive algorithm to shorten the Huffman codes, and the international V.42bis standard incorporates all three forms of the Lempel-Ziv algorithm LZ1, LZ2 and LZW, to achieve a theoretical maximum transfer rate of 38,400 bps. Stay connected.

This article has been adopted from Modems Made Easy. The author's book will be published next month. To order, send $19.95 plus applicable sales tax to Brenner Information Group, 13223 Black Mountain Rd. 1-430, San Diego, California 92129, call (619) 538-0093, or check with your local bookstore.
Modems Made Easy (ISBN 0-929535-09-X), $19.95.



EGAD Screen Print
Now Available for Z-100 -or- PC

Graphics & Text Screen Print for VGA, EGA, CGA displays (PC version), or Z-100 series.

* Print any part of the screen - crop box pops up when Shift-PrtSc (PC) or Shift-F12 (Z-100) pressed; use arrow keys to select region.
* Enlarge graphics 1-4 times
* Prints in color (color printers) or black & grays
* SET program selects printer colors, other options.

Supports most popular dot matrix, laser, and ink jet printers (including Epson, NEC-8023, MPI, Okidata, etc.) EGAD for PC's, **Order # 270**; EGAD for Z-100, **Order # 271. Either is $35.00** postpaid.

Z-Screen Library for 'C'

Screen & Keyboard I/O library for Microsoft 'C'. Two sets of source code included (one makes programs which will run on PC; the other, for Z-100). Both libraries have same interface to your code. No royalties for use. Call or write for complete description. Z-Screen, #246, **$20.00**

Lindley Systems—4257 Berwick Place, Woodbridge, VA 22192-5119 (703) 590-8890
Call or Write for Free Catalog. **Reader Service #136**

Reader Service #136

MENU.BAT

Steven W. Vagts
2409 Riddick Road
Elizabeth City, NC 27909

Tired of listening to the complaints of the kids and spouse that your computer just isn't user friendly? Tired of the familiar 'A>'? How about trying a friendly batch file to get you off the hook!

I'm sure some of you cringe at the term, batch, because it is synonymous with the term, programming. But, fear not. We're going to take this in small steps, but before we start, let me explain something else.

Those of you who read my article, "Dress Up the CP/M `A>' Prompt With a Picture!" may have been disappointed that MS-DOS doesn't have a similar capability. Ahhh . . . but it does - at least on Z-100 machines (I still have to work out the graphics problem on the PC-compatibles). Use my Z-100 version of PAINT, as described in the December 1989 and February 1990 issues of "REMark" to make the desired graphics screen. Then, using the MS-DOS command RDCPM copy the file to your MS-DOS working disk. I use the file name HELLO.SCN. Insert the lines, TYPE HELLO.SCN and PAUSE, in the batch file AUTOEXEC.BAT and the screen will be displayed in the same great (color) form.

Confused? Well, I think it will become a little clearer than mud as we explore this other dress up method, so bear with me.

MS-DOS looks for a batch file, AUTOEXEC.BAT, while it goes through its BOOT functions. If it finds one, it will run it before ending with the familiar 'A>' prompt. The key is to insert a small routine (program) under this file name. This will involve dusting off the MS-DOS manual . . . Oh, I can hear the groans already.

There are two ways to make an AUTOEXEC.BAT file. One uses the COPY command — a simple process but errors require starting all over again — or using a line editor or simple word processor to create the file and modify it later, if necessary.

Let's briefly touch on the COPY method first. Reviewing the section in the manual entitled, *How to Create Batch Files*, we start at the 'A>' and type the following:

```
COPY CON AUTOEXEC.BAT
```

Press RETURN. This command tells MS-DOS to copy the information from the console (keyboard) to the file AUTOEXEC.BAT.

Before proceeding further, what do we want the routine to do?

Many of us have a real-time clock installed, and MS-DOS has a command file, RTCLOCK, that reads and displays the date and time on the screen. If you are tired of the usual DATE? and TIME? questions during the BOOT process, we can eliminate those very neatly at the same time. Type:

```
RTCLOCK
```

Press CRTL-Z (type the letter Z, while holding down the CRTL key), and then press RETURN.

MS-DOS now saves the batch file and displays the message "1 File(s) copied" to show it created the file.

To execute the file, simply type AUTOEXEC at the 'A>' prompt. You will find the date and time displayed on the screen and a new 'A>' prompt displayed beneath.

Something I didn't discover for several years — if you leave out the CTRL-Z, you may end up with two 'A>' prompts, one under the other! Don't forget to end your batch file with a CTRL-Z. Unless your word processor permits you to insert a CTRL-Z at the end of your file, this will be a problem.

That was simple. However, we need a more complex routine than this and, as I mentioned before, this method is intolerant of errors. We need a line editor that can modify lines. Enter EDLIN to the rescue.

Review the chapter entitled, The Line Editor (Edlin), in the manual. Though not the easiest Line Editor to use, it isn't all that bad, either.

Before we leap into our new file, entitled MENU.BAT, it would be nice, though not required, to place two other files on our system disk or boot directory. As you play with EDLIN, you will find that you can't insert blank lines on the screen when it is run - at least I can't. This is a ridiculous limitation that I can't believe exists. Maybe one of you has a solution.

In any case, someone else must have also had a problem, because on a HUG disk, P/N 885-8046, entitled MS-DOS ASM UTILITIES, is a neat little program, CRLF.COM, from Mr. John Stetson. Its sole purpose in life is to insert blank lines on the screen! The source file is also there to mess with, if desired.

Another file on the same disk provides the date and time like RTCLOCK, but also includes the day of the week. Also from Mr. Stetson, the file, DATETIME.COM, is only 1K as opposed to the file RTCLOCK.COM which is 3K! Its source code is also included.

As I said, these files are optional. Without CRLF.COM you simply can't skip lines, and you already have RTCLOCK.COM. Now, let's get on with MENU.BAT.

At the 'A>' prompt, type EDLIN MENU.BAT and press RETURN. We're only going to use a few of EDLIN's more simple commands.

Following a brief opening line, you end up with a command asterisk, *, in the left margin. Type I and RETURN. Each new line will start with a line number and colon followed by an asterisk, *. Type the following lines after the asterisk. Check each line carefully before pressing RETURN, then the next line number will appear. Don't worry about a mistake at this point.

Following the line, there may be a comment in brackets, {}. Don't type these. Also, for space considerations, a number in square brackets, [], indicates the number of spaces to type between the preceding character and the next one. Don't type the brackets or the number. Here's my routine:

```
1:*ECHO OFF
```
{Commands won't be displayed on the screen as they are executed}
```
2:*CLS
```
{An internal MS-DOS command that clears the screen}
```
3:*DATETIME
```
{Displays the day of the week, date, and time. If the file, DATETIME, is not available, then type RTCLOCK instead}
```
4:*CRLF
```
{As discussed above, this skips a line. If not available, don't type anything - just press RETURN}
```
5:*CRLF
```
{I wanted to skip two lines}
```
6:*ECHO[10]* * * * * *[10]GAME MENU[10]*
* * * * *
```
{Or type whatever title you wish}
```
7:*CRLF
8:*ECHO[10]ADVENT[9]BABY[11]BOMBER[9]CELTIC
```
{While your file names may be different, use these for now}

press RETURN. Let's change line 8. Following the *, type 8 and press RETURN. You'll see an indented line, providing the entire old line, followed on the next line with 8:*. On some terminals (my Z-100 won't for some reason, but my Z-181 does), if you use the right arrow key to move the cursor across the line, you'll find the entire line print as you move to the right end. Backspace, or use the left arrow, to the point you want to change, type the new name or names and press RETURN at the end. The new line is now saved and you are returned to the command asterisk.

Press the letter L and RETURN and the entire routine is displayed with the new line. To delete a line, at the command asterisk type the line number followed by the letter D and press RETURN. If you list the routine again, the line is gone. To insert lines, at the command asterisk type the line number at which you wish to start, followed by the letter I and press RETURN. The first line number and colon is displayed followed by an asterisk as it was when we first entered the lines. Following the last of your new lines, type CTRL-C on the next blank line to exit the insert mode. If anything else were typed on that line before typing the CTRL-C, IT WOULD BE LOST! Always type the CTRL-C on the NEXT BLANK LINE.

```
 9:*ECHO[10]FIRE[11]GERM[11]HILIFE[9]KILLER
10:*ECHO[10]MAZE[11]SNOOP[10]TRIAC[10]WORD
11:*CRLF
12:*ECHO[10]* * * * * * * * * * * * * * * * * * * * * * * * * * * *
13:*CRLF
14:*ECHO[10]Type one of these names or MENU to return to this menu:
```

{At the end of line 14, type CTRL-Z, as discussed above, and press RETURN}

At line 15:* type CTRL-C. This returns us to the command line, an asterisk at the left margin. Type the letter E and RETURN. This ends EDLIN, saves the file MENU.BAT, and returns us to the MS-DOS 'A>' prompt.

Type MENU and press RETURN. The screen should show the information at the start of this article. Note any changes you wish to make, such a removing or adding spaces or asterisks. Note the file names you wish to use. As above, I don't bother with file extensions if the files run independently. If you're using ZBASIC files, however, I still don't use file extensions, but line 14 might read, "Type ZBASIC and the name of a file above. Typing MENU will give this menu."

Got all your changes? OK, let's press on. Type EDLIN MENU.BAT and press RETURN again. Changes are easy. At the command asterisk, type L and RETURN. The screen displays the list of all lines in our routine. You'll note, however, that only the first line has an asterisk after the line number. This is EDLIN's line pointer, showing the current line you're on. It's of no concern for our minor editing problem.

This time, at the command asterisk, type the line number you wish to change and

Make all your other changes and type E at the command asterisk. When you exit this time, your original MENU.BAT file now becomes MENU.BAK and MENU.BAT is replaced with the new file — as with many word processors. After ensuring MENU.BAT does what you want, you can delete this .BAK file.

Now, let's fix the AUTOEXEC.BAT file. You can use the COPY command as we did above, starting over, or we can use EDLIN again. If you are just running a simple floppy disk system use COPY:
A>COPY CON AUTOEXEC.BAT and press RETURN.

Type MENU and CTRL-Z and press RETURN. All done. Try rebooting and MS-DOS should run AUTOEXEC.BAT, that runs MENU, and you get your MENU displayed on screen. You're on your way.

For those running a hard disk, type EDLIN AUTOEXEC.BAT. At the command asterisk, type L and press RETURN. If your file was made as described above, you'll find:
```
1:*RTCLOCK
```
Using the EDLIN commands as described above, delete this line and try this file:
```
1:*ECHO OFF
2:*PROMPT $p$g
```
{PROMPT lets us change the default MS-DOS prompt. This will display the working directory with the greater than sign, >}
```
3:*PATH = E:\DOS
```
{Causes MS-DOS to check the \DOS directory on the E: drive or partition for command files. This is where I keep CRLF.COM and DATETIME.COM, for example}
```
4:*CLS
5:*MENU
```
At the end of line 5, don't forget the CTRL-Z. At line 6 type CTRL-C to exit to the command asterisk.

Finally, if you want to display a PAINT file as discussed when we started, then when any key is pressed, display a menu file, insert (*5I) the following lines to your AUTOEXEC.BAT file:
```
5:*TYPE \DOS\HELLO.SCN
6:*PAUSE
```
Type CTRL-C at line 7. Typing L at the command asterisk shows your line 5 to now be:
```
7:*MENU
```
Still with me? Good! For hard disk users, let me go one step further. Those of you with an H/Z-100 will have hard disks with several partitions - probably several for MS-DOS applications, another one or two for CP/M, and probably a few more for ZPC and PC-DOS files - up to sixteen partitions! Confusing? My wife and kids would be in a panic trying to manipulate around all these, if they bothered trying at all!

Try this on for size. My AUTOEXEC.BAT file for MS-DOS is as listed above. The MENU.BAT lists the names of the other partitions. (See Figure 1.)

Then, each of the above has their own batch file, most of which must assign another partition (the Z-100 is limited to 4 active partitions at a time, labeled E: thru H:). MENU.BAT gives the above menu selection.

As an example, GAME-Z.BAT is:
```
1:*ECHO OFF
2:*ASGNPART 0:GAMES;Z-DOS G:
```
{ASGNPART is another file in my \DOS directory. This command assigns a logical drive name (from the range of E-H) to a physical hard disk partition. 0 is the hard drive unit (0-3) that I want to use. GAMES is the partition name. Z-DOS is the name of the operating system used & stored on the specified partition. G: is the logical drive name (E-H) that we want to use}
```
3:*G:
```
{Makes G: the default drive}
```
4:*MENU
```
{Each of the partitions has its own menu directory similar to that used above for each floppy disk}

The batch file, GAME-PC.BAT, is the most complicated because we must also run ZPC before our games:
```
1:*ECHO OFF
2:*ASGNPART 0:PC-DOS F:
3:*PATH = E:\DOS;F:\ZPC
4:*F:
5:*IF EXIST \ZPC\ZPC.COM GOTO X
6:*ECHO ZPC.COM file not found!
7:*GOTO Y
8:*:X
```

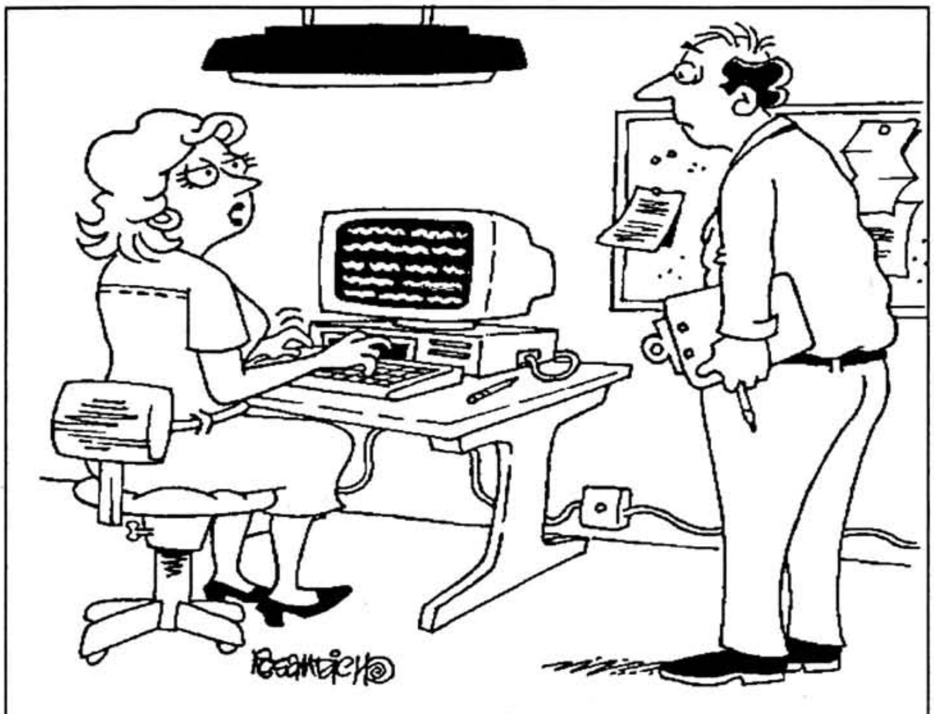
```
1:*ECHO OFF
2:*CLS
3:*DATETIME
4:*CRLF
5:*CRLF
6:*ECHO Current directory is Z-DOS.
```
(This is not actually Z-DOS, but is labeled to differentiate from IBM MS-DOS, which is ZPC)
```
 7:*CRLF
 8:*ECHO * * * * * *            DIRECTORY MENU            * * * * * *
 9:*CRLF
10:*ECHO BASIC       DBASE         GAME-BAS     MENU          Z-ART
11:*ECHO CAD                       GAME-PC      PT            ZPC
12:*ECHO                           GAME-Z       WP
13:*CRLF
14:*ECHO Type one of these commands or ZDIR for full directory:
```
Figure 1

```
 9:*ZPC
10:*ANSISYS
11:*CRLF
12:*CRLF
13:*CD F:\GAMES
14:*MENU
15:*:Y
```
This gives an example of conditional (IF-THEN) branching to X or Y, loading ANSISYS for IBM PC escape sequences, and changing directories (CD). All this complexity takes a lot of time to sort out and set up, however, it sure takes the pain out of moving around the various partitions.

The individual partitions also include a ZDOS.BAT batch file to get you back to the original MS-DOS directory from which you started.

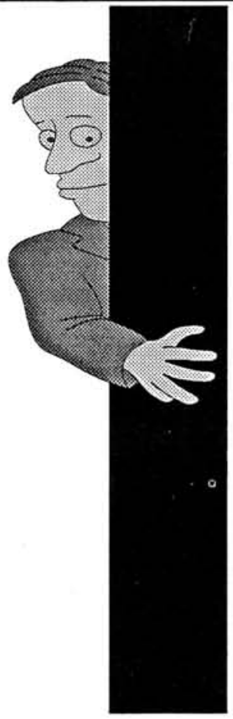
One last batch file for those using WordPerfect 5.0 under ZPC. Entitled WP.BAT, it consists of:
```
1:*ECHO OFF
2:*ASGNPART 0:PC-DOS F:
3:*PATH = E:\DOS;F:\ZPC
4:*F:
5:*IF EXIST \ZPC\ZPC.COM GOTO X
6:*ECHO ZPC.COM file not found!
7:*GOTO Y
8:*:X
9:*ZPC
10:*ANSISYS
11:*CD \WP
12:*PC
13:*WP
14:*CD \
15:*MENU
16:*CRLF
17:*ECHO  Enter  PC  to  emulate  IBM-PC
computer.
18:*ECHO Enter Z100 to return to normal
mode.
19:*:Y
```
I hope you have found this dissertation both useful and entertaining. If it has helped in some small way to straighten out the complexities of batch file use, it would have been worth the effort.

Until the next time I get the urge to write and say 'Hi', I wish you all the best in your programming efforts. "REMark" — keep up the good work. I enjoy every issue.


"I'LL GET TO THE JOHNSON SURVEY REPORT AS SOON AS I FINISH MY SON'S HOMEWORK."

ACCOUNTING & TAX

Not sure if you need the expensive 'Chinese Flower 1-2-3', or 'Spanish Numeral Four' spreadsheet programs? Then find out for only $20! "CheapCalc" will do double precision addition, subtraction, multiplication, division, power, SUM, and roots (using fractional powers). CheapCalc has many other functions too numerous to mention (just like the expensive spreads)! CheapCalc is available for all Heath/Zenith computers and operating systems. For more information, check out page 58 of the Software Catalog Update #1, or call HUG and order your copy today.



We are still trying to get caught up on our issues.

We would like you to know how much we appreciate your patience. Thank You!

A 286LP 8 to 12 Megahertz Upgrade Made Easy

George C. Ludden
2102 Planters Row Drive
Midlothian, VA 23113

Background

In June, 1990, Greg Braithwaite described his development of a method for upgrading 8MHz 286LP computers to 12MHz. The upgrade consists of soldering new IC's and other components to the main board. I thought long and hard about whether I was confident enough to tackle this task. The downside risk was that I could end up with an expensive boat anchor if I damaged the board. Eventually, I decided to go ahead. If Greg could figure out how to do all this, the least I could do was give it a try to see what happens.

Greg's procedure can be found as 286LP12.ZIP in the ZDS file section of the COM1 bulletin board.

Planning and Parts

Greg's instructions included a list of the parts I would need to install. I had the board part number 85-3384-01 described in the instructions. I began my search for the parts by calling the Heath parts department. The folks there were very helpful. They had all the parts I needed except one (which I will mumble about later) and the total cost was only $25.63. They accepted my VISA number over the phone (little did they know) and the parts arrived in just a few days.

Since I was going to be working at the board level, I decided to upgrade my monitor ROM (444-643-1) to the current version (444-643-10). That cost an extra $12.60, but probably will prove useful in the future.

Part number 969-1158, the 24MHz crystal, turned out to be a "Zenith only" part. Now to find a dealer/repair shop that might have one. Hah! It just happened that most of the dealers are Heath dealers. None in this area had the part. I had trips to San Antonio and Phoenix planned, so I tried the various dealers in those cities, figuring I could pick one up while I was there. Nope - none there either. I ended up having to order one through a Zenith Data Systems dealer. It cost $23.50, almost as much as all the parts from Heath together.

The instructions indicated that seven new IC chip sockets were needed. I got those and some solder (desoldering) braid at a local electronics store.

My soldering gun is old and made for the days of welding leads to buss bars in the chassis. Anybody remember those days? A lot of heat was needed, which today could fry the fragile foils on a circuit board. I decided to upgrade my tools appropriately for delicate work. A new lower temperature tip for my soldering gun and a small soldering iron with a built-in desoldering bulb both proved to be very helpful during the project.

As Greg suggested in his instructions, I ran the PCTools system information test, which showed the computer operated at 455% of the speed of the original PC. This was useful for comparing the old and new speeds. I then did a printscreen of the monitor ROM setup so that I wouldn't have to guess what the setup was later.

I decided to exchange the monitor ROM before doing the speed upgrade to make sure it worked. I would hate to have nothing work later and wonder whether it was the new ROM or the speed upgrade that had a problem. I used a small screwdriver to pry out the old ROM and made sure the pins on the new ROM were straight before inserting into the socket. A brief test of the computer showed everything was working properly. The only difference I could tell was the option to put a security password on the system.

Getting Started

Finally, a day came when I could spread out all the parts and the computer in the kitchen and work all day without being disturbed. Another one of my fantasies; with three boys and a dog rampaging through the house, distractions were common. "No, Jeff, you can't take that neat looking IC to school for show and tell; the computer won't run very well without it. And don't try to see if the dog thinks it tastes good!"

HINT: Use a camera with the computer cover removed or draw pictures to show arrangements, where various wires, cables sockets go, etc. This will help when reassembling the computer later.

I used a large antistatic plastic bag (the type used for shipping circuit boards, etc.) to cover the workplace to prevent accidental static discharges. Always touch the plastic first to minimize the chance of zapping an IC chip.

First came the cover. Pull the cabinet forward carefully. The edges are sharp and could cut the floppy/hard drive cables that rest on top of the drives. Examine these cables closely. If you see any sign of abrasion or cutting of the insulation, cover it with electrical tape. If it looks like any wires are cut, or the drives don't work properly later, there's a good chance that the cable needs to be replaced. After removing the cover, I removed the Z-549 video card and put it in a safe place.

HINT: When you take out the various screws and other small pieces, make a sketch showing where the screws came from. Then store the screws, etc. in an envelope or Ziplok bag until you need them again. This will help you from getting to the end of the job and saying, "Where do all these extra parts go? Ah, heck, I guess we don't need 'em anyway." It also helps to make sure which screws go in which holes, since there are some unused holes, and you won't have to try to figure out which ones to use.

I cleaned off the years of accumulated dust and dirt from the inside of the cabinet with a vacuum cleaner. This can be a chancy operation since a small static discharge could ruin one or more circuits. Don't try this unless you are confident that

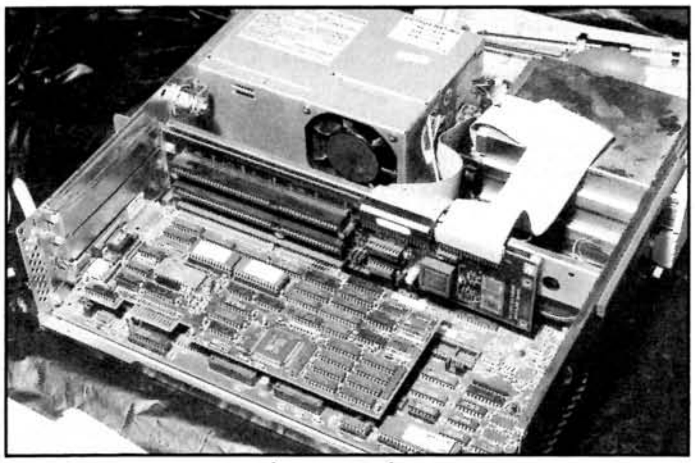


Photo 1. Before.

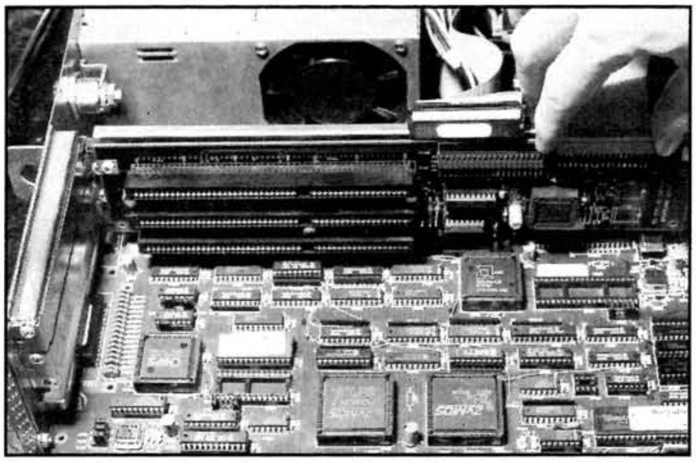


Photo 2. Drive Cables.

there is no chance for static.

Next, remove the floppy/hard drive cables from the backplane board by pulling up gently but firmly. There is no need to remove the cables from the drives or to remove the drives themselves.

There are three connectors on the mother board that have to be removed. They are located between the backplane connector and the drives. Unplug the speaker lead, the battery lead, and the power supply from the motherboard and push the leads aside.

Use a good pair of pliers or hex wrench to unscrew the six screws holding the parallel and serial ports to the chassis. Don't forget to put these in a safe place for reassembly.

Remove the two screws attaching the backplane board to the chassis. One is at the top rear and the other is somewhat hidden between the top two card connectors. (See Photo 3.) Squeeze the plastic clip connector at the top front of the board with a pair of pliers and pull the board slightly. The board then will pull straight up from its connection to the motherboard. Put the board and screws in a safe place.

Remove the five screws holding the mother board to the chassis. Gently pull the keyboard connector mini-board up from the motherboard to disconnect it and store it safely.

Gently work the motherboard out of the chassis. There may be a little interference near the parallel and serial connectors. It may help to move the speaker up out of the way slightly and lift the front edge of the motherboard. Whatever you do, don't force anything. If (or when) you get frustrated and want to loosen it with a two pound hammer (you know that will work), quit! Take a break. Yell at the kids. Kick the dog. Break a window. But don't, whatever you do, don't try to force anything. Patience will eventually work, and force will probably break something permanently. Put aside the chassis until it is time to reassemble the computer.

Put the motherboard on your (preferably anti-static) work surface and examine it closely. Compare it to the list of components to be replaced and find the location of each on the board. Always touch a ground surface (such as the antistatic surface or a parallel or serial port) first to dissipate any built up static.

Everything to this point has been reversible. From here on you will be making electrical changes to the mother board. If you're squeamish, think twice about going any further. Once you get started, you can't back out. At this point, Quincy (a second generation mutt) decided that she wanted to fix the board her way. She stuck her nose right in there and threatened to bury the board next to the dog house. That was the last she saw of the kitchen until the job was finished.

Solder Removal

There are two methods to remove solder from circuit boards. The desoldering iron with suction bulb works well where holes in the board have been filled in; the wicking action of solder braid is best for removing solder where components are soldered in place. I used both of these methods for this project.

HINT: When using a desoldering iron, make sure you never, never, never squeeze the bulb when the iron is near the board. You may squeeze molten solder onto the board and create shorts and a multitude of other problems. Always squeeze the bulb over a moist paper towel, hold the bulb while you melt the solder joint, and release the bulb to suck up the solder. Then squeeze the bulb over the paper towel to expel the solder.

Old Parts Removal

The first parts to be removed were IC chips at U518 and U520. Located near the keyboard connector, I used desoldering braid to remove as much solder as possible. (Photo 6 shows the working area.)

HINT: Double and triple check component numbers to make sure you are work-

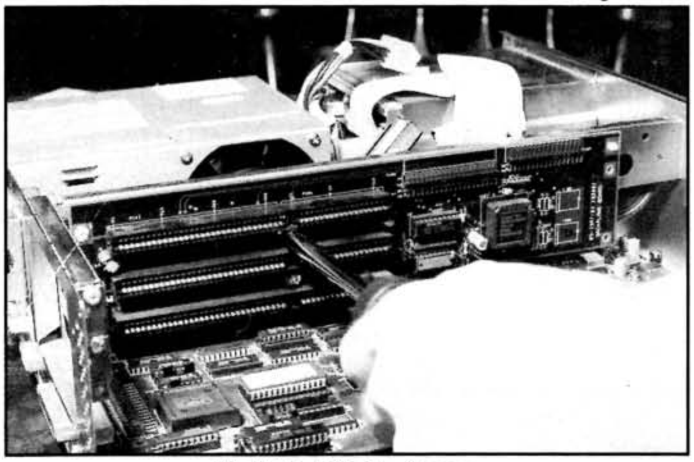


Photo 3. Hidden Screw.

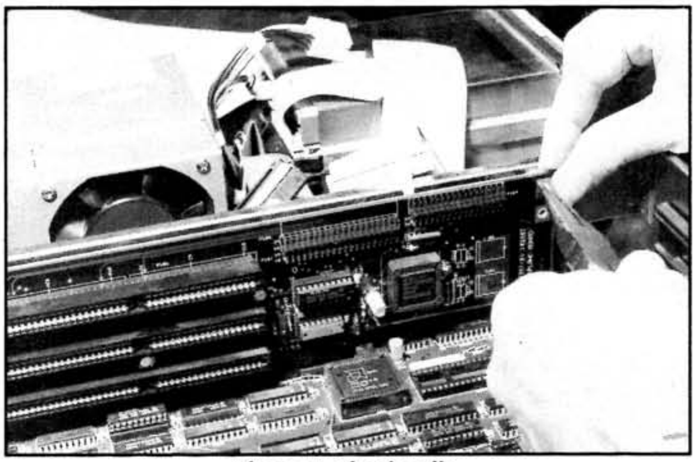


Photo 4. Plastic Clip.

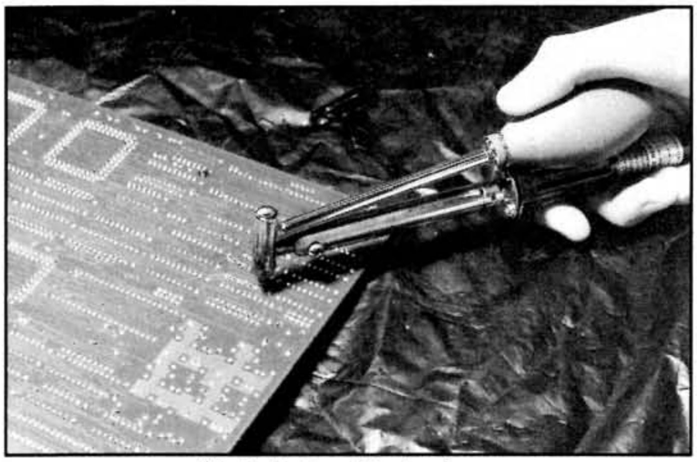


Photo 5. Desoldering Iron.

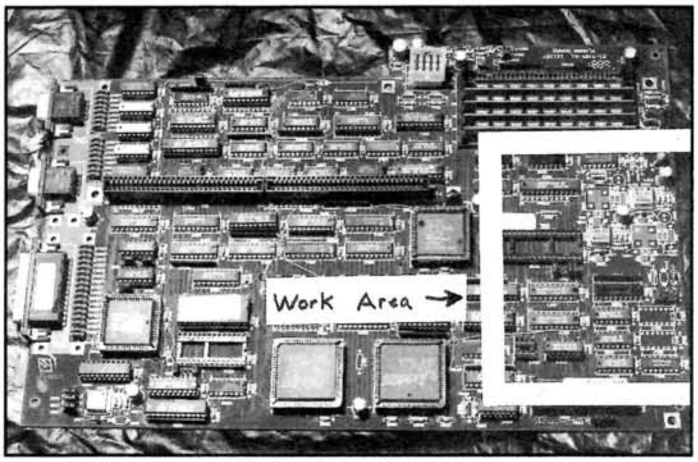


Photo 6. Work Area.

ing on the correct one. You sure don't want to take out the wrong one, although my son Mark thought it was such fun, he thought we should take them all out. Mark, go play with Quincy.

Removing existing IC's may be the toughest part of the whole project. If a component has only two or three soldered pins, it's easy to remove it by alternately working on each pin. But when removing a multi-pin component, it's virtually impossible to remove it intact, even by using the desoldering braid and desoldering iron. Enough solder is retained to hold the component firmly to the board. In these situations, I cut each pin on the component side of the board, remove the component, and then desolder each pin separately. Make sure that you get all the solder and pins out of the hole, but be careful not to damage the foil.

HINT: Take your time. This can be very frustrating, and frequent breaks will help. Hurrying and trying to force the component won't help, and may lead to damaging the board.

Clean the solder from U517, U519, C518 and C523. Since there are no existing components at these locations, the desoldering iron is probably the easiest to use.

HINT: After you have finished these steps, hold the board up to the light to make sure all the holes are clear and ready for the next step.

New Parts Installation

Now solder in four 14-pin IC sockets at U517, U518, U519, and U520. Note that the end of the socket with the notch should line up with the marked end on the board. Solder two opposite corner pins on each socket and recheck the sockets to make sure all pins go through the board and all sockets lay flat against the board. Then solder the remaining pins. Check all solder joints to make sure that they are good joints, with no solder bridges or extra pieces of solder. Now solder in capacitors at C518 and C523, cut the excess leads, and check the joints.

HINT: Use a magnifying glass to check for good solder joints, loose solder, etc. It's easier than burning your eyeballs out trying to see those tiny little buggers.

There! Wasn't that easy? If you made it this far, you've got it made. Your blood pressure should be coming down by now.

Now it's time to insert the IC's in the four sockets you installed. Make sure you touch the grounding material before you remove and handle each IC (to prevent static). Check the pins on each IC to make sure they are straight; bend them as necessary to make them straight. Note that the end with the notch should be aligned with the notch on the socket. Since these four IC's are almost identical to each other, double (or triple) verify the part numbers as you install them.

Remove the jumpers at U525. You may find it easy to cut each wire and then desolder the holes to remove them. Also desolder the holes at RP-503. Install the 20-pin socket at U525 and the resistor pack at RP-503, similar to the 14-pin sockets above. Plug in the IC at U525. Be careful about the orientation of the resistor pack and the IC when installing.

HINT: Not all of the holes may have jumpers installed; note which holes have jumpers in case you need to reinstall them at some future time.

There is a step to add a capacitor at C529, but my mother board already had a capacitor there so I skipped this step. If your board doesn't have the capacitor, desolder the holes and install a capacitor.

The next series of steps requires desoldering a number of holes for crystals and capacitors. Several of these would not open either with desoldering braid or the desoldering iron. I had one of my sons hold the board on edge and applied heat from the foil side. Being very careful not to damage the foil, I used a very sharp, pointed set of tweezers to twist and drill through the hole

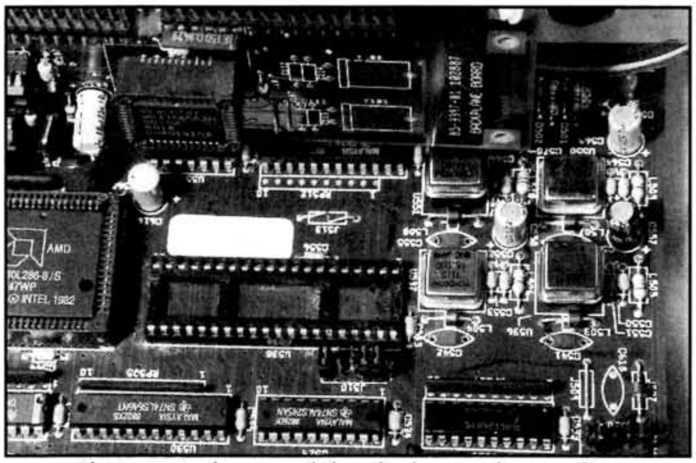


Photo 7. A closeup of the clock crystal area after modification and reinstallation.

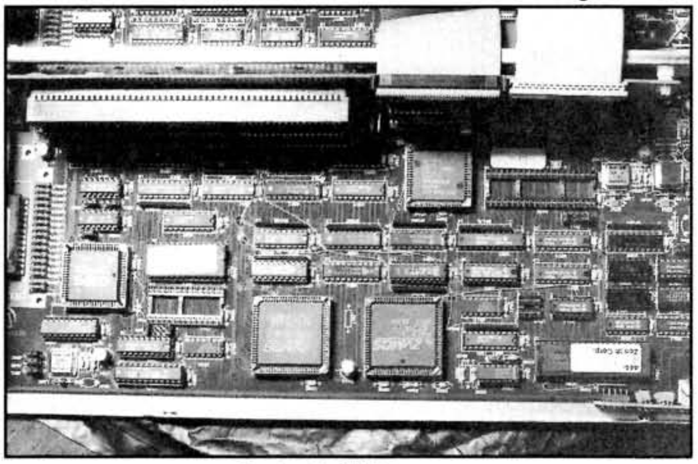


Photo 8. All Done.

from the component side.

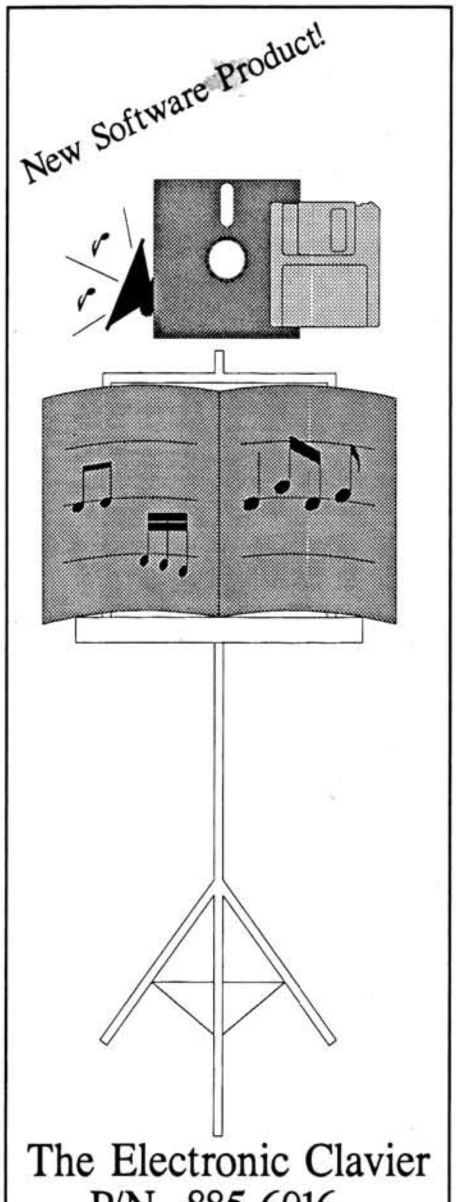
The new crystals could be soldered into place but it is easier to solder a socket and then just press the crystal into the socket. It also makes it possible to remove the crystal later if needed. An 8-pin socket is used. Since the crystal has only four pins, the middle two pins on each side are removed by pulling out with needle nose pliers. The sockets are then installed at U536 and U551. Solder capacitors into C550, C551, C566, and C567.

Solder the electrolytic capacitors into C557 and C570. Since electrolytic capacitors are polarized, be careful to match the "+" or "-" sign on one side of the capacitor to the appropriate hole on the board.

Solder the striped coils into L505 and L510 and the ferrite coils into L503 and L508.

Insert the 24MHz crystal into the socket at U551 and the 32MHz crystal at U536. In lieu of other instructions, I oriented the crystals similar to U537 and U550 already present. Photo 7 shows this area.

There is a single jumper at J513. Cut this jumper in the middle of the wire and separate the cut ends slightly. Don't remove the wire, since it may be needed if you ever want to slow the machine back to 8MHz.



The Electronic Clavier
P/N 885-6016

Final Checks and Reassembly

Once again, use a magnifying glass to check all solder joints and look for loose pieces of solder and solder bridges. I used a "microduster" blower to make sure any tiny pieces of solder, wire, or IC chip leads were removed from the board before reassembling.

Carefully slide the motherboard back into the cabinet. Move the speaker up to make room to lift the front side of the board. This will help get the serial ports into position.

HINT: Make sure there are no loose parts or wires under the board as you slide it into place. It is easy to let one of the wires next to the disk drives wander under the board where it can get stuck.

Don't forget to include the small keyboard connector with the front screw. Plug it into its connector and then insert the screw.

HINT: Gently remove the plastic card holder on the front face of chassis to make it easier to install the keyboard connector. After the mother board and keyboard connector are installed, reinstall the card holder.

Insert and gently tighten the five mother board screws. Don't overtighten as there is a possibility the board could crack. Install and tighten the hex screws for the parallel and serial ports.

When inserting the other boards, I gently cleaned the edge connectors with a pencil eraser to remove any oxidation which may have built up over the years. This will help ensure good contact in the future.

Insert the backplane board into the mother board. Push the hole in the upper front corner onto the plastic clip and insert and gently tighten the two screws. Connect the power supply, speaker wires, battery and the two disk drive cables securely.

Install the video card and any other cards and replace the cabinet, being careful not to cut the disk drive cables. Now it is time to hook up the monitor and check out your work.

Power Up

The bets were running heavy in our neighborhood by now. 5 to 1, the speed increase wouldn't work; 3 to 1, nothing would ever work again; even odds, the monitor would smoke, too, when I turned it on.

I couldn't look. But I didn't smell any smoke. Finally I peeked. There was a message on the monitor. My system configuration was missing. A push of the <ESC> key took me to the monitor ROM. All configuration settings had been lost when the battery was disconnected. I pulled out the printscreen sheet and reset the time, date, RAM, and drive information.

This time the system boot was successful. The machine went through all the gyrations it has to in my config.sys and autoexec.bat files. I couldn't believe it! Tentatively, I started up various application programs. They all worked! Fantastic!

I hot keyed into PCTools and ran the system information check. It now shows a speed of 690% of original PC speed. That correlates exactly with the clock speed change from 8Mhz to 12Mhz. Since that day, I haven't found any bugs in the system.

Caveats, Always Caveats

My upgrade was successful. But was I just lucky? Needless to say, the modifications are not supported by Zenith Data Systems. There are a lot of potential pits to fall in when attempting this type of change. The following are some of the most obvious.

1. The CPU was supplied to run at 8MHz. It may not be able to handle the higher speed. If so, it can be replaced with a 12MHz 80286 chip.
2. The RAM memory for the 8MHz units is 100 ns, which may not be fast enough to keep up with the higher speed CPU. It can be replaced by 80 ns memory. That would be a good excuse to buy 1 meg SIMM memory and get full benefit from the additional memory.
3. There may be other speed sensitive chips on the board. I'm not a computer designer, so I just crossed my fingers and followed the directions. Everything on my board seems to be compatible with the upgrade.
4. Zenith Data Systems, as with most manufacturers, makes running changes on their products. Most of these won't be documented and don't affect normal operation, but if your board has a change that affects these modifications, it's possible the upgrade won't work. So be it. At least you can revert to the original 8MHz operation by removing the crystals, IC chips, and reconnecting the jumper. I verified that this works.

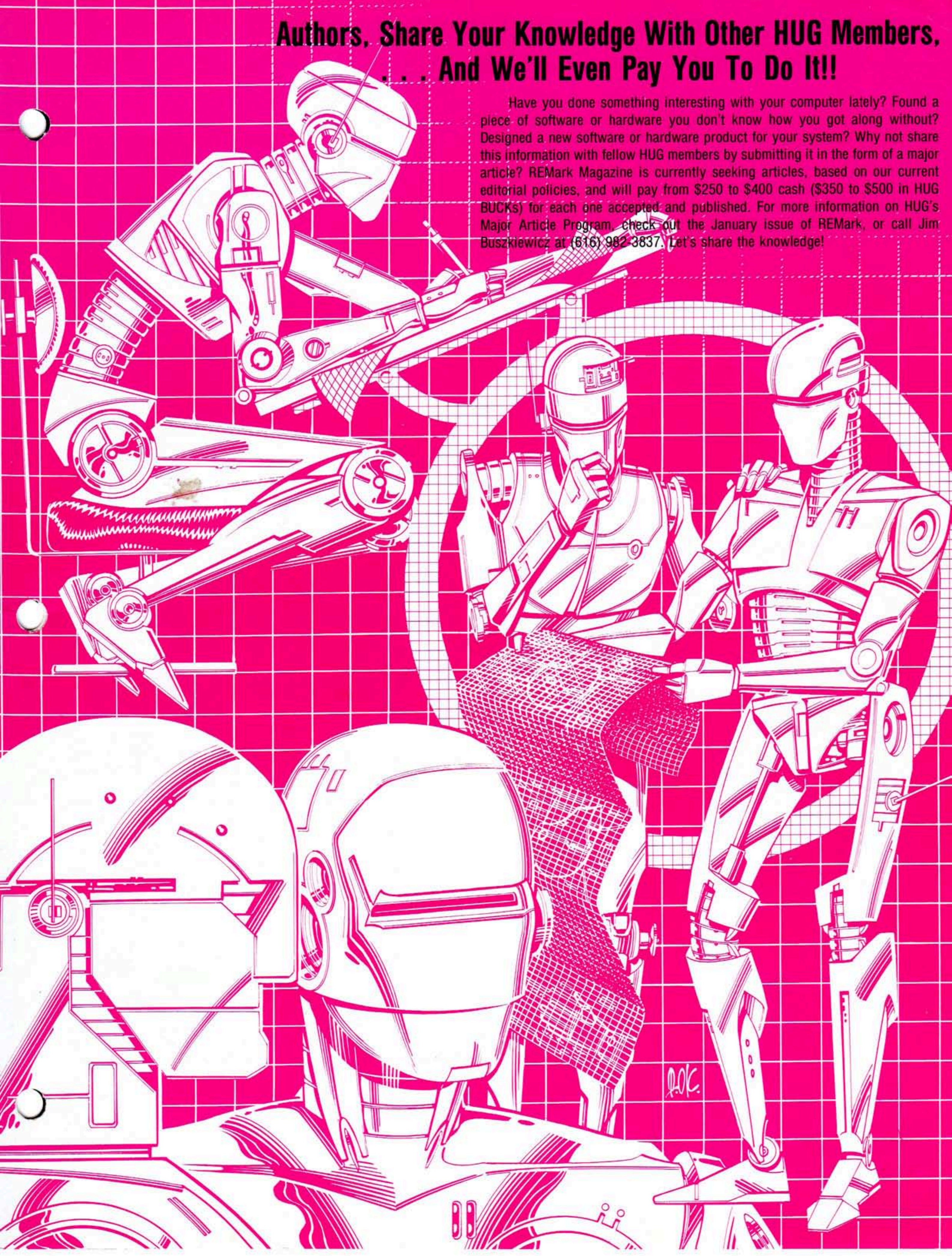
Finis

The 8MHz 286LP is a great machine, perfect for home use or any application where bunches of expansion slots aren't needed. At 12MHz, it's a screamer. Ready to give it a try? Chances are that your upgrade will work perfectly. It's certainly worth the try.

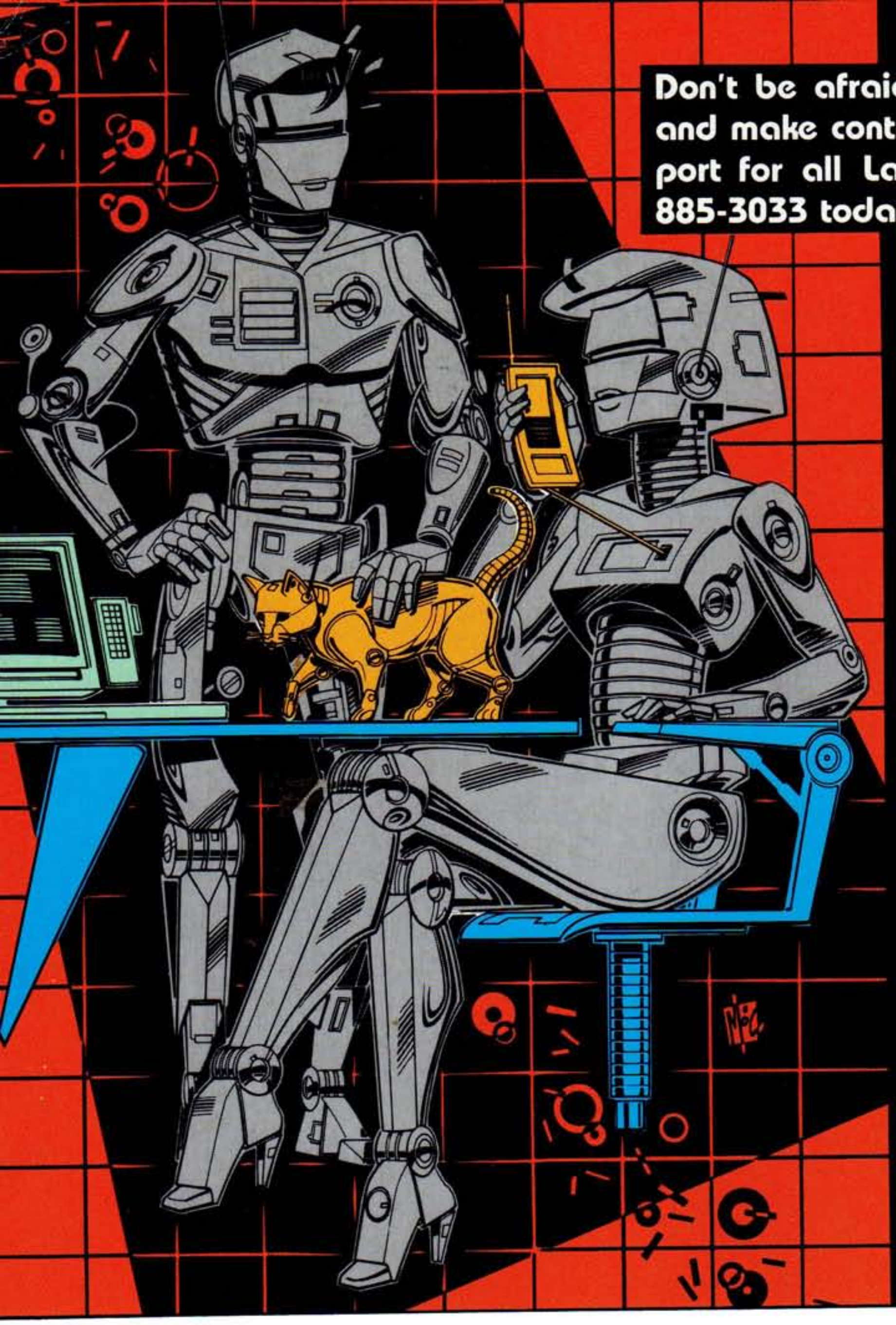
OK, let the kids and dog back in the house now. I'm finally finished. It's time to get back to the serious stuff, like Indiana Jones and the Last Crusade.

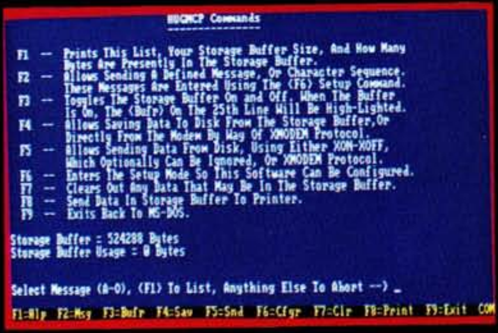
Authors, Share Your Knowledge With Other HUG Members, . . . And We'll Even Pay You To Do It!!

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 software or hardware product for your system? Why not share this information with fellow HUG members by submitting it in the form of a major article? REMark Magazine is currently seeking articles, based on our current editorial policies, and will pay from $250 to $400 cash ($350 to $500 in HUG BUCKs) for each one accepted and published. For more information on HUG's Major Article Program, check out the January issue of REMark, or call Jim Buszkiewicz at (616) 982-3837. Let's share the knowledge!

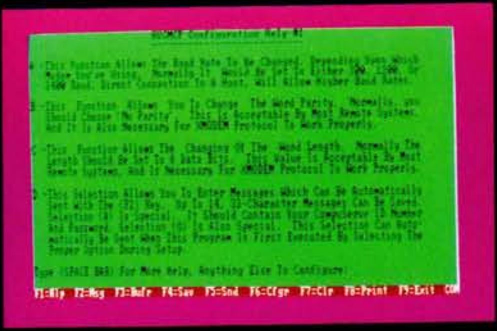


Don't be afraid to communicate! Get HUGMCP and make contact the easy way. Now with support for all Laptops, order HUG Part number 885-3033 today.

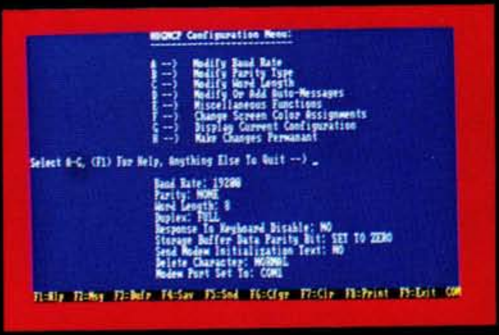




```
                         HUGMCP Commands
                         ---------------

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 (F6) Setup Command.
F3  --  Toggles The Storage Buffer On and Off. When The Buffer
        Is On, The (Bufr) 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  --  Allows Sending Data From Disk, Using Either XON-XOFF,
        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-O), (F1) To List, Anything Else To Abort --> _
F1:Hlp  F2:Msg  F3:Bufr  F4:Sav  F5:Snd  F6:Cfgr  F7:Clr  F8:Print  F9:Exit  COM
```



```
                 HUGMCP Configuration Help #1

A -This Function Allows The Baud Rate To Be Changed. Depending Upon Which
   Modem You're Using, Normally It Would Be Set To Either 300, 1200, Or
   2400 Baud. Direct Connection To A Host, Will Allow Higher Baud Rates.

B -This Function Allows You To Change The Word Parity. Normally, you
   Should Choose 'No Parity'. This Is Acceptable By Most Remote Systems
   And It Is Also Necessary For XMODEM Protocol To Work Properly.

C -This Function Allows 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.

D -This Selection Allows You To Enter Messages Which Can Be Automatically
   Sent With The (F2) Key. Up To 14, 32-Character Messages Can Be Saved.
   Selection (A) Is Special. It Should Contain Your CompuServe ID Number
   And Password. Selection (B) Is Also Special. This Selection Can Auto-
   matically Be Sent When This Program Is First Executed By Selecting The
   Proper Option During Setup.

Type (SPACE BAR) For More Help, Anything Else To Configure:
F1:Hlp  F2:Msg  F3:Bufr  F4:Sav  F5:Snd  F6:Cfgr  F7:Clr  F8:Print  F9:Exit  COM
```



```
                 HUGMCP Configuration Menu:
                 --------------------------

                 A --)  Modify Baud Rate
                 B --)  Modify Parity Type
                 C --)  Modify Word Length
                 D --)  Modify Or Add Auto-Messages
                 E --)  Miscellaneous Functions
                 F --)  Change Screen Color Assignments
                 G --)  Display Current Configuration
                 H --)  Make Changes Permanant

Select A-G, (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: NORMAL
                 Modem Port Set To: COM1
F1:Hlp  F2:Msg  F3:Bufr  F4:Sav  F5:Snd  F6:Cfgr  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 885-2135