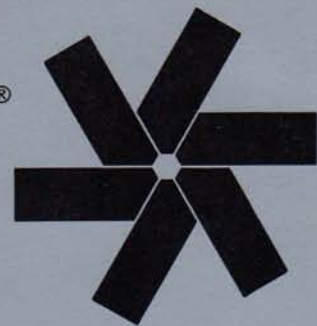
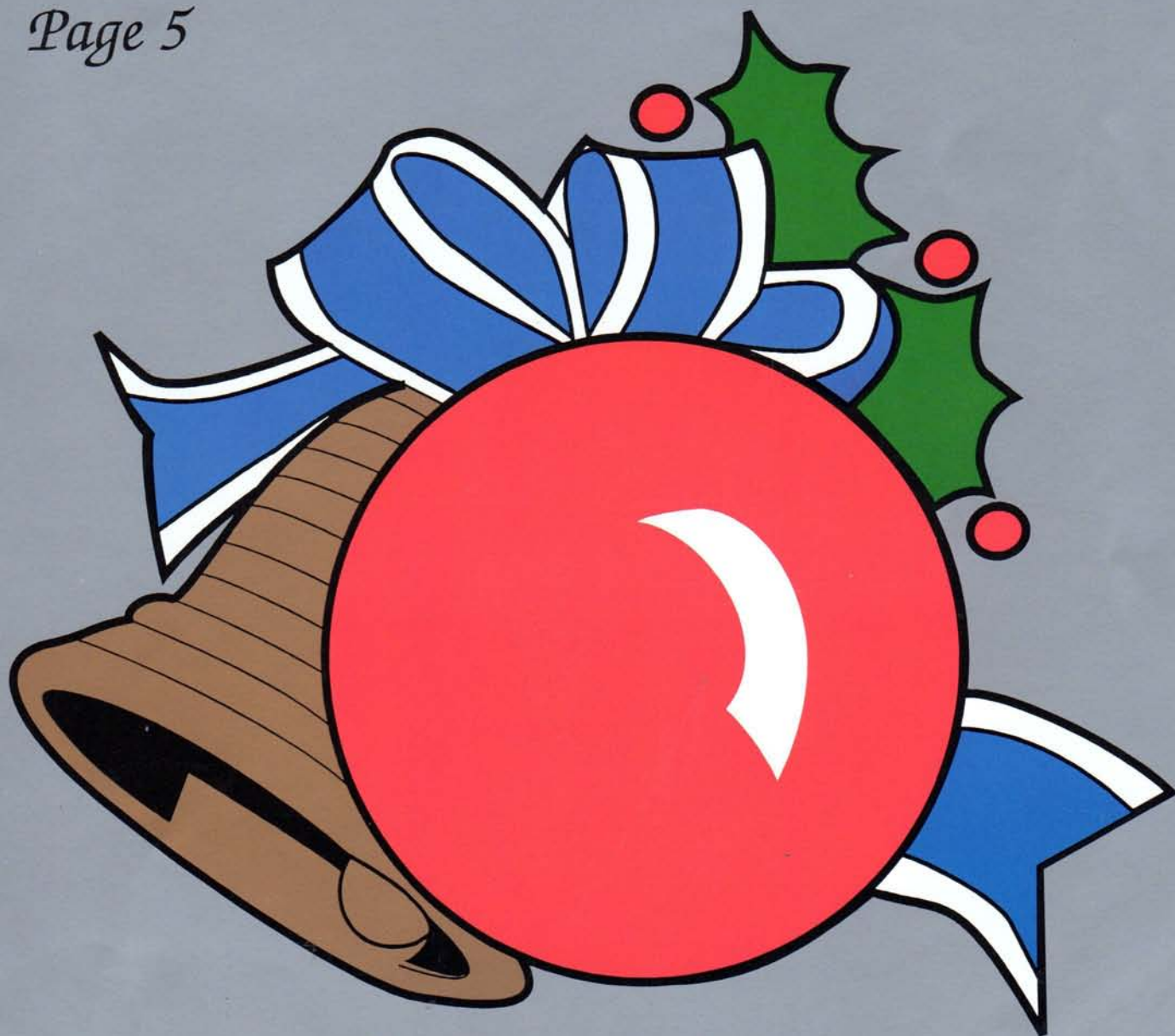


REMark[®] December 1991

The Official Zenith Data Systems Magazine



Install a High-Performance Z-649 Video Card
Page 5



Clearing Up the Memory Mystery
Page 23



Share the Knowledge!

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

Share the Knowledge!

REMark®

December 1991



The Official Zenith Data Systems Users Magazine

Install a High-Performance Z-649 Video Board into Your Computer

David R. Veit 5

Introduction to C++ Eleventh Installment

Lynwood H. Wilson 9

Getting the Most From Your Computer Part 7

John P. Lewis 13

I Can Do It! — Part 2

David Warnick 18

Binary Conversion by Division Why it Works

Gil Hoellerich 21

Clearing Up the Memory Mystery

Pat Swayne 23

Developing Applications Programs for Windows 3

Harold C. Ogg 26

Budget Desktop Publishing is Here!

Pat Swayne 31

LANtastic

Jeff Babcock 34

Son of Small Computer System Interface

Nathan Baker 37

WordStar — Past and Present

William Warren Pitts 44

Advertising

	Page No.
Domino Computers.....	30
FBE Research Co., Inc.	36
QuikData, Inc.	8
W/PUG.....	20
WS Electronics	17

Resources

Software Price List.....	2
Renewal Form	4
HEPCAT Patch	25

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

To Locate your Nearest:

Dealer 1-800-523-9393
Service Center 1-800-777-4630

	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.
Check ZUG Product List for availability of bound volumes of past issues. Requests for magazines mailed to foreign countries should specify mailing method and include appropriate, additional cost.

Send Payment to: Zenith Users' Group
P.O. Box 217
Benton Harbor, MI 49023-0217
(616) 982-3463

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

Articles submitted by users and published in REMark, which describe hardware modifications, are not supported by Zenith Data Systems Computer Centers.

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

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

Copyright (c) 1991, Zenith Users' Group

Software

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

Price List

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

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

Attention!!

Zenith Data Systems Owners

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

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

ORDERING INFORMATION

For VISA, MasterCard, and American Express phone orders, telephone the Zenith Users' Group directly at (616) 982-3463. Have the part number(s), description(s), and quantity ready for quick processing. By mail, send your order 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.

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). 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	<input type="checkbox"/> 19.95	<input type="checkbox"/> 22.95
Address: _____	U.S. First Class	<input type="checkbox"/> 32.95	<input type="checkbox"/> 37.95
City, State, Zip: _____	APO/FPO Surface Overseas	<input type="checkbox"/> 32.95	<input type="checkbox"/> 37.95
Daytime Phone #: () _____	Air Printed Overseas	<input type="checkbox"/> 52.95	<input type="checkbox"/> 57.95

Install a High-Performance Z-649 Video Board into Your Computer

David R. Veit
Design Engineer
Zenith Data Systems

With computer users demanding PC video solutions requiring greater resolution, color depth, and processing speed than conventional IBM VGA video, the industry has responded by offering several options.

These options include video boards offering "extended" VGA modes of operation, usually referred to as SuperVGA. There is also IBM's venerable 8514/A, plus specialized video boards based on Texas Instruments' 34010 and 34020 family of graphics processors. As of this writing, there are also rumblings that IBM will license another vendor to produce its newly released EXtended Graphics Array (XGA). But until that XGA becomes available for the standard ISA PC bus from IBM, an IBM licensee, or until other video board vendors have a chance to "clone" it for the PC bus, it won't be an attractive solution. Currently, XGA is only available for PS/2 machines using the MicroChannel Architecture.

In the past, video solutions had been pretty much dictated by what IBM had to offer: MDA, CGA, EGA, VGA. These solutions are all based on variations of the display controller theme where the primary CPU (your 80x86 processor) did all of the video processing with the help of a dedicated video controller. As video resolution and color depth have increased, there has become a processing bottleneck and speed has suffered greatly. As an example of the increased video processing requirements, standard VGA resolution of 640x480 and 16 colors has 300K pixels (picture elements) of information, requiring 150K bytes of memory to be processed for every screen update. Increasing to 1024x768 resolution and 256 colors means 768K pixels and 768K bytes of memory to be processed.

For example:

16 color pixels: defined by 4 bits → 2 pixels/byte of memory
256 color pixels: defined by 8 bits → 1 pixel/byte of memory

640 x 480 = 307,200 = 300K pixels for one display page

1024 x 768 = 786,432 = 768K pixels for one display page

Therefore, 640x480 at 16 colors requires 150K of video memory and 1024x768 at 256 colors requires 768K of video memory.

Note that this increase in resolution and color depth has 2-1/2 times as many pixels, and 5 times as much memory to be processed. That seemingly small increase in resolution and color depth requires a vast increase in video processing power! So, if you've ever run a graphics intensive package, such as AutoCAD or Microsoft Windows, now you know why it sometimes takes a long period of time for the computer to redraw the screen! This also explains that while a SuperVGA solution may offer excellent resolution and color depth, it is notoriously slow! SuperVGAs simply do not have the capability to process great numbers of pixels quickly. It is a limitation inherent to the controller-type architecture of the video system. On the other hand, 8514/A- and 34010/20-based boards use a different theme. They transfer the burden of video processing from the host CPU to an on-board processor (34010/20 boards) or "graphics engine" (8514/A). The result is usually a great improvement in speed performance.

Zenith Data Systems recently introduced their Z-649 video board to address high performance video solutions above and beyond standard VGA. It has a 60 MHz 34010 graphics processor and utilizes the Texas Instruments Graphics Architecture (TIGA) as a software interface. It comes equipped as a standard accessory in several of their high-end 80486-based machines, but it can be installed into virtually any standard ISA or EISA PC. The beauty of this board is that its greatest impact is seen when used in lower-end machines, especially '286 and '386SX machines. The performance enhancement as seen in those

machines is incredible. They also provide enhanced performance in high-speed '386DX and '486 machines, but these processors are so fast that the video processing bottleneck is not nearly so severe.

What is a Z-649?

ZDS' Z-649 is a 34010 graphics processor-based, plug-in card and comes with TIGA interface software. TIGA is the primary means by which the PC application software or user can access the Z-649 hardware and have the 34010 processor, rather than the host 80x86 processor perform various video functions. The Z-649 is capable of 640x480 or 1024x768 resolution, and can display up to 256 simultaneous colors. Since the 34010 is a programmable device, it requires program memory in addition to the memory required for storing video display information. The Z-649 comes equipped with 512K for 34010 program memory and 512K for video display memory. Each memory type can be individually upgraded to 1 MB.

ZDS includes with every Z-649 a Windows 3.0 driver. This Windows 3.0 driver was written to a Direct Graphics Interface Standard (DGIS). Also included is all software necessary to run any application that has a TIGA software driver. For example, many independent software vendors have written a generic TIGA driver for their software products. This allows the particular application to run on any video board set up to use the TIGA interface. A Windows 3.0 TIGA driver for the Z-649 is not yet available from ZDS, but may be available in the near future.

How is the Z-649 Used in a PC Environment?

One issue that needs to be clear is that the Z-649, like most high-performance video cards, is not a compatible video solution in a PC environment. What this really means is that the Z-649 cannot replace your existing IBM-compatible video adapter, it sim-

ply adds capability to the overall video solution in your machine. It works along with your existing video card and "enhances" video operation by providing non-standard video modes and an alternate path for graphics output for packages such as AutoCAD and Windows.

Actually installing a Z-649 into your system and setting it up for operation is a breeze. The documentation that comes with the Z-649 is very complete and is especially helpful to the novice computer user. One thing to keep in mind, and something that seems to cause confusion among those unfamiliar with PC video operations, is that you will have two sets of video modes: PC-compatible and non-compatible. Your PC-compatible video system (your MDA, HGC, CGA, EGA, or VGA) provides all of the standard, "compatible" video modes while your Z-649 is the source of the non-compatible modes. Depending on the amount of display memory on your Z-649, and whether a hi-resolution display is attached, the following video modes are offered by the Z-649:

Display Resolution	Colors	Pages
1024 x 768	256	1
1024 x 768	16	2
1024 x 768	16	1
1024 x 768	2	2
1024 x 768	2	1
640 x 480	256	2
640 x 480	256	1
640 x 480	16	2
640 x 480	16	1
640 x 480	2	2
640 x 480	2	1

Note that several variations of the familiar 640x480 VGA resolution are offered. All of these 640x480 modes run correctly on a standard VGA monochrome or color analog monitor. All 16 and 256-color modes will show as variant shades of gray on the monochrome monitor, and as true colors on the color monitor. A hi-resolution monitor is required to run all 1024x768 modes. This hi-resolution monitor must run correctly at a 47.70 KHz horizontal refresh rate (standard VGA is 31.5 KHz). Unlike IBM's 8514/A and XGA, all video modes are non-interlaced, so the resultant video images on the screen are rock steady and have no flicker.

One of the unique and very useful features of the Z-649 is its ability to accept digital video information from an external video source and "pass it through" to a video display connected to its video output connector. This was designed explicitly to accommodate the video pass-through feature seen on all VGA systems. A VGA "features" connector is the source of digital VGA video information. Instead of pro-

cessing this information with the VGA's analog output circuitry, the information is passed to the Z-649 and processed through the Z-649's analog output circuitry. The analog output circuitry of both systems are functionally equivalent. The primary reason for passing VGA video information to the Z-649 is that it allows the user to have two video subsystems and still get by with using only a single display monitor. In a single display system, the Z-649 and accompanying software is "smart" enough to pass the correct video source to the display.

A system with multiple display monitors is also possible. This feature is not unique to the Z-649. It is quite common for an MDA card to co-exist in a computer alongside a VGA card. An analog monitor would be connected to the VGA, and a Monochrome Display attached to the MDA. The VGA would provide all of the VGA video modes except mode 7, which is provided by the MDA. Certain software applications then take advantage of this fact and provide the user with information on both displays. Likewise, when a Z-649 is resident, the IBM-compatible video subsystem would provide the PC-compatible modes, and the Z-649 would provide its own non-compatible modes. TIGA application software, such as an AutoCAD driver, take advantage of this feature and can provide information to the user through both mediums: the PC-compatible subsystem (i.e., VGA) provides text information, such as command lines and user menus with VGA video modes, and the Z-649 TIGA subsystem provides hi-resolution graphical information using the TIGA video modes.

The Z-649 will reside with any valid combination of PC-compatible video adapters. It is not absolutely required that the Z-649 co-exist with a VGA; in fact, it can just as easily co-exist with only an MDA card. It will also co-exist with most any other type of video card, such as an 8514/A, if one so desires. The point to understand is that the Z-649 was specifically designed to interface to the host computer entirely through a block of 16 I/O addresses, and a single user-selectable hardware interrupt. The Z-649 will not respond to any PC memory accesses or non-maskable interrupts (NMI's). With so little to get into the way of other resources, there is very little chance of conflict with other video cards, network cards, or anything else that can be inserted into, or run on your computer.

What is TIGA?

When the PC video industry began looking beyond VGA towards higher resolution and color depth, most took the route of simply extending the operation of the VGA to include extended video modes. There are two major problems with this

approach. First, we know that higher resolution and more on-screen colors requires greater processing capability. Using the traditional CPU + controller method meant that performance took a plunge. Second, the various video manufacturers could not agree on how to implement these extended video modes. It is absolutely critical to have some sort of consensus or standard among vendors to do things a certain way. Without a standard to go by, computer monitor vendors and application software developers would be very reluctant to develop products for a haphazard collection of proprietary video systems. To be fair to SuperVGA vendors, they are attempting to standardize. The Video Electronics Standards Association (VESA), a consortium of video vendors, was formed to address the issues of standardization.

Knowing that the field was wide open for someone to take charge, Texas Instruments threw their hat into the ring of video vendors and announced intentions of standardizing a method of high-performance video beyond VGA. They brought with them a pair of high-speed graphics-oriented processors (TMS34010 and TMS34020), which are ideal for accepting some of the video processing burden from the main CPU, and the knowledge that if they could develop a generic software interface that didn't compromise video performance, was essentially independent of hardware and color/resolution, and was programmer friendly and user extensible, they might create a standard for implementing non-standard, extended video modes. They did create a software interface, known as Texas Instruments Graphics Architecture (TIGA), and it indeed has become an accepted standard within the industry, and is probably the only non-IBM solution that is widely accepted. Its implementation is similar in concept to IBM's Adapter [Software] Interface (AI) for the 8514/A. However, it is much more flexible in that it allows a user to add custom functions. IBM's Adapter Interface does not.

Understanding the concept of TIGA with its non-compatible video is sometimes confusing. Consider the scenario where you have a VGA installed into your system. You can think of all applications, including DOS, as running under standard VGA video modes. All information sent to your display is handled by the host CPU and resident VGA. But, when you want to run an application that requires higher resolution or more colors, you can use your Z-649. Your application software must now "communicate" with the Z-649, rather than the VGA. The problem is that you now need special software that will allow your application to "communicate" with the Z-649 hardware. Having your software application communicate with your VGA is a trivial task because all VGAs are identical. However, it is

entirely impractical to expect software developers to provide support directly within their application for the numerous types of proprietary video boards in use.

Instead, this is where the TIGA software interface standard comes into play. The application developer provides a low-level software "driver" which communicates with another low-level driver provided by the hardware manufacturer. In this case, ZDS has provided what is commonly called a "TIGA communications driver" or TIGA CD, and is written explicitly for the Z-649 hardware. The application driver, if written to conform to the TIGA software standard will now work on any piece of hardware that includes a TIGA communications driver. In this way, software developers only need to write a single software driver conforming to the TIGA interface standard, and hardware developers only need to write a single TIGA communications driver. This is in stark contrast to the previous burden of one party developing a host of drivers that link specific software applications to specific hardware. The industry supporting SuperVGA is like this. For example, SuperVGA hardware vendors write drivers to allow many popular software applications to run on their unique hardware, and software vendors write drivers allowing their applications to work on popular, but non-standard video hardware. It is a vicious circle that both software and hardware vendors would like to see eliminated.

How Well Does the Z-649 Perform?

Trying to come up with meaningful performance criteria for a Z-649 card and comparing resultant benchmarks with a VGA system is a very tricky proposition. First, just what exactly constitutes a meaningful test? Some of the tests that are popular with graphics-oriented video cards are ones that rate performance of line and curve drawing functions, bit-block transfers, windowing operations, and graphical text display. Since Microsoft Windows 3.0 is becoming so popular, I decided to run PC Magazine's Windows Benchmark program as a benchmark to evaluate overall performance of the Z-649. Identical tests were run on two separate machine configurations:

1. 12 MHz 80286 computer
80287 numeric coprocessor installed
Z-649 TIGA and Z-559 VGA video installed
2. 25 MHz 80486 EISA computer
(numeric coprocessor is integral to 80486)
Z-649 TIGA and Z-559 VGA video installed

The Z-559 video card is a high-performance, 16-bit VGA card with 256K bytes of video memory installed, upgradeable to 512K, and is based on the Headland Tech-

nology (Video 7) HT-208 VGA chip.

ZDS ships Z-649 cards as standard in certain models of their '486 EISA computer. I ran the Windows Benchmark test in a 25 MHz '486 machine with results shown in Figure 1. Note that in standard 640x480 16-color resolution, the Z-649 ran about 30% faster than VGA. In 640x480 extended color mode, the Z-649 was considerably faster, over 3 times as fast as the VGA. Just for comparisons sake, I included results for some of the hi-resolution 1024x768 modes. Note that the Z-649 was able to process considerably more video information in benchmark times very comparably to 640x480 resolution VGA.

color mode, the Z-649 outperformed VGA by 2-to-1, while in 640x480 high color mode, the Z-649 had a similar 3-to-1 performance increase. Running 1024x768 modes on the Z-649 were also much faster than 640x480 modes on the VGA.

These benchmarks should give users an idea of what kind of performance gains they can expect from the Z-649 card. It is meant to be run with a high-resolution monitor, such as ZDS' ZCM-1650 monitor, but if you only have a standard VGA monitor limiting you to the 640x480 modes, the performance increase still may make the Z-649 an excellent investment. For example, the '286/12 with Z-649 runs 640x480 16-

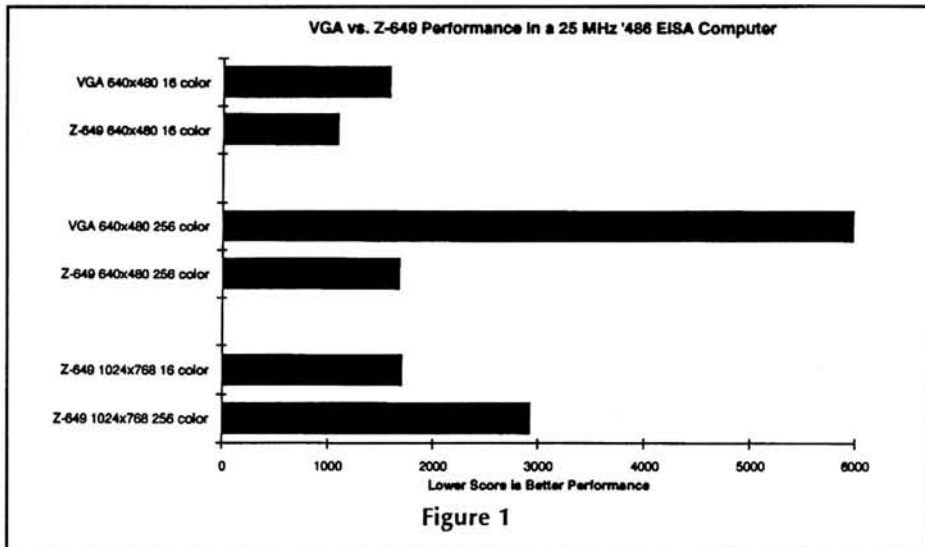


Figure 1

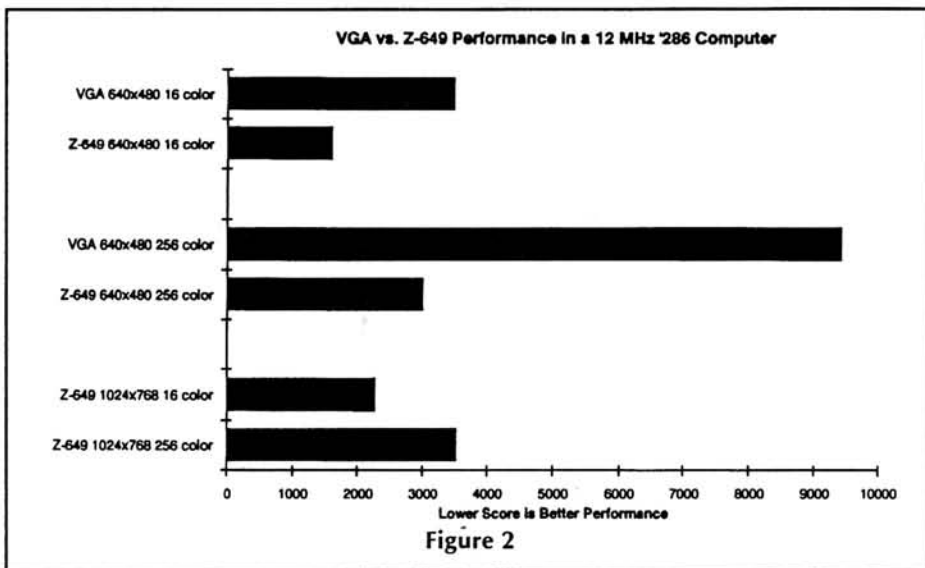


Figure 2

Since a '486/25 machine with VGA is a very fast machine, the video processing bottleneck is not nearly so severe as that for a much slower machine. Figure 2 shows results for the exact same Windows tests as above, except they were run on a 12 MHz '286 machine. As expected, the performance gains by using the Z-649 over a VGA were more dramatic. In 640x480 16-

color Windows almost exactly as fast as the '486/25 EISA with VGA! To make things better, the '286/12 with Z-649 runs 640x480 extended color Windows almost twice as fast as the '486 VGA machine! If anything, the Z-649 certainly is a bargain if you simply need to speed up your graphics intensive applications! ✨

QUIKDATA - 16 YEARS OF H/Z SUPPORT!

YOUR H/Z ENHANCEMENT EXPERTS!

ACCELERATE YOUR PC/XT/AT!

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

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

EXP-12 - \$249

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

For your H/Z241, 248 and 386/16 we have the new direct replacement **WESTERN IMAGING Z-33 ZUNI MOTHERBOARD** that's just loaded with features. 33Mhz 80386 main board with ten expansion slots and built in are two serial ports, one parallel port, floppy disk controller and an IDE disk controller! That gives you 10 slots. Add video (and your old MFM controller if you desire) and you still have plenty of expansion left! Support to 32 meg on-board RAM using SIMMs, and 32 additional megs with memory expansion.

WIZ-33 - \$849 with 0K RAM

80387-33 - \$195 Coprocessor

SIM1X9-7 - \$59 1 meg X 1 70ns SIMM DRAM

SIM256-8 - \$13.95 256K X 1 80ns SIMM DRAM

MEMORY UPGRADES

Note: All memory upgrades come without memory chips. 150ns 256K DRAMs are \$1.59 as of this printing.

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

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

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

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

3MB RAM BOARD for Z241/248 computers is an excellent memory card. Will backfill your 512 to 640K, and provide both extended and expanded RAM; all can coexist. Uses 100ns M256-10 RAM chips, 36 per megabyte desired. Minimum of 18 DRAM chips required (\$1.79 each).

EVA TRD - \$119

Z248/12, Z286LP RAM UPGRADE Z605-1 consists of 2MB SIMM 80ns RAM kits to upgrade your H/Z systems.
Z605-1 - \$125

Z386/20, Z386/25, Z386/33, Z386 EISA 2MB SIMM 80ns UPGRADE to add increments of 2MB to these systems. Two required.
ZA3600ME - \$89

ZA3800MK - \$275 4 megabyte SIMM upgrade for Z386/20, /25, /33, /33E. Must have 4-1 meg SIMMs installed first.

We also have memory upgrades for just about every H/Z desktop and laptop unit except the 171, 181, 183, and 184 series. Tell us what you need and we can quote you a price.

WINCHESTER UPGRADE KITS

PCW40 - \$319 Complete MFM winchester setup for a H/Z150, 148, 158, 159, 160, PC etc. Includes **42 meg** formatted half-height Segate ST-251 28ms drive, controller, cable set, doc.

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

PCW80 - \$569 80 meg with Segate ST-4096 full size drive.

ST-4096 - \$495 Bare drive only

DTC ON - \$59 PC/XT hard drive controller board

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

IDECON - \$95 AT bus IDE/floppy controller for placing an IDE hard drive in any AT compatible.

FLOPPY DRIVE SAMPLE

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

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

MF353 - \$75 720K 3.5" drive in 5" frame

MF355 - \$79 1.4 meg 3.5" AT drive in 5" frame

TM100-2R - \$65 40tk DS refurb (H8/89/Z100 PC type)

Also other drives and full line for older systems

ANY DRIVE IN YOUR PC/XT/AT

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

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

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

BPACK2.8 - \$279

BPACK1.4 - \$229

BPACK1.2 - \$229

BPACK360 - \$229

ADD AN EXTERNAL 80MB TAPE BACKUP DRIVE to any laptop or PC with the Microsolutions Backpack tape backup system. Plugs into a parallel port (do not lose the port) to give you an affordable way to easily backup your hard drive and/or transfer data. Uses DC2000 tapes. Fast! **BPACKT8 - \$445**

8-BIT/Z100

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

OTHER STUFF

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

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

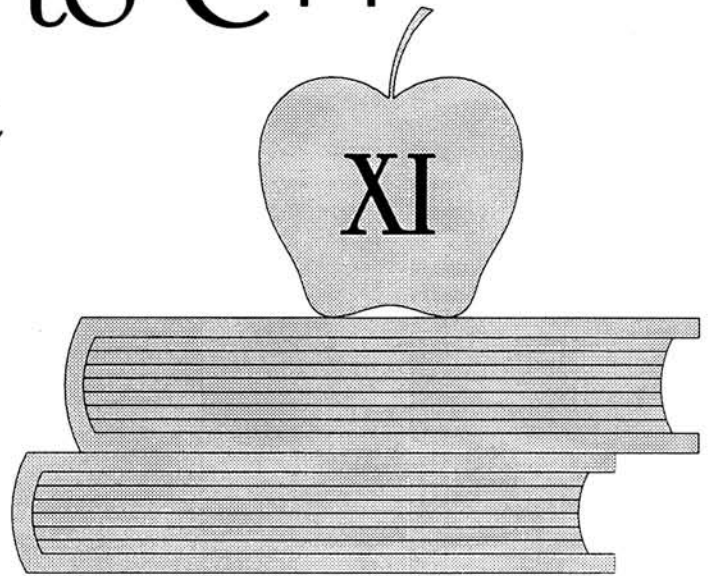
QUIKDATA, INC.

2618 PENN CIRCLE
SHEBOYGAN, WI 53081-4250
(414) 452-4172

Introduction to C++

Eleventh Installment

Lynwood H. Wilson
2160 James Canyon
Boulder, CO 80302



This last month or so I've been writing an assembler. It's an interesting project, one of the primal problems in computer science. As I recall the history, about three days after the first computer was running someone said "It sure takes a long time to program this sucker, seeing as how we have to do it in machine code and all." So an assembler was certainly one of the very first system programs. The problem has been solved many times in many ways, and the classical solutions are well known. However, my client wanted it done in C++ and an object oriented solution is not so well known.

I am going to devote the next installment or two to the design and implementation of the assembler. I think the problems and solutions of a real project are just as useful as tutorial material on the details of the language and a lot harder to find in articles and textbooks. And even harder to find is information on the process of specification, design, and implementation of a program the way it really happens. In addition, the assembler uses techniques which are useful in many other programs.

The Beginning

My client had designed a microprocessor chip with an instruction set optimized to control a complex machine and needed an assembler for it. The assembler would be a simple one, lacking macro capability and a lot of the other features we see in most modern assemblers. The programs written for it would be fairly short, and such features were unnecessary. At first the client insisted that the assembler be written in ANSI standard C. I argued for C++, but only briefly. If you want their money, you have to give them what they

want.

When I started on the project the specification consisted of a preliminary description of the instruction set of the new chip and a page of overview which said, in essence, "Do it like any other assembler." My contract was to design and write the assembler, but my first task was to work with the client's people to help develop a proper specification. This is not an unusual situation in my experience. Fortunately, the people I was working with understood the need for a solid specification and did not begrudge the time and money to write it. So we went to work on the specification.

Specification

I wrote up a spec based on the information I had and the client's people corrected and added to it. We went through several such cycles and finally produced a document we were all satisfied with. It was not a complete specification, such as recommended by the texts on structured programming, but we had reached the point of diminishing returns.

There are several reasons for beginning design before having a complete and perfect specification. One of the best is that a complete and perfect specification is impossible outside of the classroom. Since you will have to stop short of the ideal, better to admit it and make a reasoned decision rather than to pretend.

Another reason is that some kinds of decisions are better deferred. Some decisions which are difficult to make at the time of writing the spec are easy later after part of the design is done and things have begun to settle into place. This is an area where OOP is of great benefit. Because of the strong divisions between program ele-

ments in an object based program, the interactions between these elements are weaker and more clearly defined. This allows the effects of one area of the program on another to be understood better. It also allows the internal workings of an area of the program to be left undefined longer once you have defined its outward links.

A working prototype is the best kind of a specification. No amount of ink on paper can convey the kind of information about the finished program that a prototype can. The best way to find the remaining flaws in a specification is to show the client a prototype based on the spec. The very best way to help someone figure out what he really wants is to give him what he asks for. Thus the division between specification and design (and, for that matter, implementation) grows a bit vague.

While we worked together on the specification, I began to research assembler design.

Classical Design

I found several books on compiler design, but none on assemblers. Some sources seem to consider the assembler as a simple case of a compiler. This may be true, but it's of little help unless you want to learn how to write compilers first. Finally I turned up an old book, "Assemblers, Compilers, and Program Translation" by Peter Calingaert. It was published by Computer Science Press in 1979 and is a bit dated and long out of print but it is still the best source I found on classical assembler design.

For those of you who have no experience with assembly language, I recommend that you look into it. Even if you do not write assembly language as a regular thing, a passing acquaintance will make

you a better programmer. In the meantime, here is a very short description.

An assembler is a program which takes in as data an ASCII file of another program written in assembly language and translates it into machine language ready for execution. In assembly language, each line of code can represent at most one machine instruction. Some of the lines represent directives to the assembler or address labels and produce no machine instructions at all. This makes the assembler seem simple, merely read a line of the source code and write the equivalent machine instruction to the object (machine language) file. There is a bit more to it than that, however.

Even a simple assembler has to handle symbols and symbolic addresses. Here is a sample of assembly code.

```
ibm    equ 1      ;this is a comment
dec    equ 0

begin  move a, 7  ;move 7 to the a
        register
      move b, 5  ;and 5 to the b reg.
      jump label1 ;jump to the
                address held by
                label1

      move b, 2
      add  a, b

label1 sub a, ibm
      jumpn begin ;if the a reg is 0,
                jump to begin
```

The words in the second field are either directives for the assembler or opcodes to be translated into machine instructions. The word `equ` is a directive to the assembler to set the value of the symbol in the first field to whatever is in the third field. All the other words in the second field, `move`, `jump`, `add`, etc. are opcodes.

The words in the first field on the left side are all symbols. The ones followed by `equ` are set to the value which follows the `equ`. The symbol "ibm" for example is given the value 1, the symbol "dec" gets the value 0. They are like constants and hold the values from then on.

The other symbols, `begin` and `label1`, are address labels and are given the address of the line they appear on. `begin`, for example, is given the address 1, since it is on the same line as the first machine instruction. The two lines with `equ` in the second position are directives to the compiler and do not produce machine instructions. The blank line is ignored, and also does not produce a machine instruction. `label1` gets the value 6, since it follows the fifth machine instruction. (This is simpler than most assemblers, since the instructions contain whatever data they need and thus all instruction steps are the same length.)

You can see here one of the complications of the problem. In order to be able to write the machine instruction for the line "jump label1" we need to know the value of `label1` but it is not yet defined. The traditional solution is for the assembler to

make two passes through the source code. In the first pass the assembler will figure out the values of all of the symbols and save them in the symbol table. Then in the second pass the machine instructions are produced and written to the object file.

And, of course, whenever the assembler finds an error, it should write an error message to the error file. This message should include the line which contains the error.

At this point, the design for the assembler could be written in pseudocode like this:

First Pass

```
open the source file
open the error file

while there is another line in the source file
  read in a line
  parse the line into symbols, instructions and arguments
  if cannot parse the line
    write a message to the error file
    write the current line to the error file
  if there is a symbol definition
    if there is an error in symbol definition
      write a message to the error file
      write the current line to the error file
    put the symbol and its value in the symbol table
```

Second Pass

```
go back to the beginning of the source file
open the object file for the machine code

while there is another line in the source file
  read in a line
  if there is an opcode
    if an argument is a symbol
      look up its value in the symbol table
    if there is an error in the opcode or its arguments
      write a message to the error file
      write the current line to the error file
    translate opcode and its arguments to machine language
    write the machine code to the object file

close the source, object, and error files
```

There are a lot of details which are not addressed in this design but it describes in broad outline how a classical assembler works. Note that like most pre-OOP designs this one concentrates on the operations to be performed and pays only passing attention to the data.

Object-Oriented Design

I won't even mention writing the code, since it was in standard C and this is an article about C++. I will only mention that I had the first pass running when the client decided that it should be in C++ after all. Fortunately, my mind has been pretty well corrupted by the object paradigm and I had written it a lot like C++ anyway. In a future article I will take up the topic of writing object oriented programs in standard C, for now I will only say that the conversion was not as difficult as you might expect. But I did have to re-think the design a bit. Back to the literature.

The literature on object oriented design is still pretty sparse. I've read several of the books available, but none that I would

recommend very strongly. I think that there are three different factors here. One is that the paradigm is new, or at least newly popular. Another is that OOP seems to require less attention to design. And the third is that no one seems to know how to do it. Or at least, no one (that I've read) seems to know how to describe it formally in a way that I can apply in practice. There are certainly people designing successful OO programs, I've done it myself. But I too have trouble describing what I do.

I do not want to minimize the importance of having a solid and comprehensive

design before you begin writing the code for any sort of a program, but it seems that in C++ the design does not need to be quite as complete.

As always, or at least since 1980 when the first edition came out, I turned to my favorite design book. The Practical Guide to Structured Systems Design by Meilar Page-Jones is the most readable and useful book on program design I have ever found. I expect I've read it a dozen times, through two editions. I recommend it most highly, even though it does not even mention objects or classes. It turns out that a lot of the principles carry over from structured design into the object world. OOP does not in any sense replace or invalidate structured programming, it only offers a new way to define structure.

A large part of the book is devoted to the design of functions, the operations to be encapsulated in a function, the linkages between functions, data passed in and out, and so on. If you substitute classes for functions, a great deal of it applies pretty well to OOD. The central questions are

similar. What will the classes be, what data and operations will each contain, what hierarchy of inheritance will they fit into, what will be the other links between them.

Selection of Classes

The words are important here. We are going to select the classes of objects for this program from among the data items we see in the problem. We will be guided by the problem. In a sense, we are selecting from things which are all ready in existence, rather than creating something new. (The data items we will use are of our creation, certainly, but somehow it seems that they are more an intrinsic part of the problem than the code, that it is somehow easier to see the data we will need to write the program than it is to see the operations to be performed on it.) Of course the program is a new thing we will create, but the basic materials we use are drawn from the problem. This seems to me to be one of the most important differences between OO design and its predecessors.

How to start. If we look at the assembler as a black box, its input is a source file and its outputs are an object file and an error file. (The machine language file has always been called the object file, but it has nothing to do with the other objects we are interested in.) Therefore, it might make sense to have a File class with objects derived from it to handle the disk files. (It is my convention to capitalize the first letter of class and structure names.)

What data would a File class hold? A pointer to a FILE structure, certainly. We can safely leave the rest of the data for later.

Now we have the data component for the first of our classes. But what operations should be encapsulated with it? This is a question of lesser immediacy. Given a good selection of classes, operations may be added later as the design progresses and the need for them surfaces. We can see the need for some of the operations right now. Certainly we will need to open and close files. If we are to make two passes through the source, we will need to return to the beginning of a file. If we are to process the source a line at a time, we will need to be able to read the next line from a file. And we will need to write a line to a file.

There are many different ways to represent a class during the design process. I use the actual class declaration. It contains all the necessary information about the class and when the design is complete, you just move it to your code.

In the early stages of the design we can be informal. We have not defined the word line, below, but we all know what we mean.

```
class File {
    FILE *fptr;
public:
    void open();
    void close();
    void rewind();
    line read_line();
};
```

```
void write_line(line);
};
```

Thinking about how the File class is to be used, it seems that it will open the associated disk file when the object is created and close the disk file when the object is destroyed. If so, the open function should be in the creator function and the close function should be in the destructor. In that case, the name of the file to open should be passed into the constructor.

```
class File {
    FILE *fptr;
public:
    File(char *fname) {open(fname);}
    ~File() {close();}
    void rewind();
    line read_line();
    void write_line(line);
};
```

Now that we have the File class, what next? We chose the File class by looking at the assembler as a black box, and the only data types we could see were files. In order to find more classes, we must go inside the assembler. When I did so, the first things I saw were the first pass and the second pass. (I was still thinking in terms of operations rather than data, as you can see.) When I looked closer I realized that the first pass existed to build the symbol table and the second pass got some of its data from the symbol table. Therefore the next class I created was the symbol table.

First we must decide what a symbol is. For the purposes of this job, a symbol is an eight character (or fewer) name with an associated integer value. Sounds like a structure to me, so I made it one.

```
struct Symbol {
    char itself[9];
    int value;
};
```

The data portion of the symbol table is an array of those things. Having that, it is time to turn to the code necessary to deal with them.

First, it seems reasonable that the constructor for the symbol table should know how to fill the table from the source file. The source file object should therefore be passed into the symbol table constructor. Thus what we originally thought of as the first pass of the assembler will now be part of the constructor for the symbol table.

We will also want to be able to look up a symbol and get its value since that is what the symbol table is for. Here is my first version of the Symbol_Table class.

```
class Symbol_Table {
    Symbol s_Table[1000];
public:
    Symbol_Table(File src_file);
    int get_val(char *symbol_name);
};
```

The pseudo code for the constructor, Symbol_Table(File src_file), will look a lot like the original design for the first pass. See Figure 1.

Again, this is a procedural description. If we force ourselves to look at it in terms of the data, we find several potential classes. How about a Line class which knows what

is in its fields and how to parse itself into field 1, field 2, arguments and comments? And after that, fields that know whether they are symbols, directives, instructions, or arguments. Here is a first design for the Line class.

```
class Line {
    char line[81];
public:
    Line(char *src_line);
    field1[9];
    field2[9];
    arg1[9];
    arg2[9];
    arg3[9];
};
```

Note that the fields and arguments are public. Generally the data in an object is private so that only the functions which are part of the object can get at it. In this case it seems more efficient to make them public. We can reconsider this decision later.

Here is the pseudocode for the constructor, Line(char *src_line). The pseudocode gets to looking a lot like C++ sometimes.

```
if there is a first field
    fill field1
else
    field1 = null
if there is a second field
    fill field2
else
    field2 = null
if there is a first argument
    fill arg1
else
    arg1 = null
if there is a second argument
    fill arg2
else
    arg2 = null
if there is a third argument
    fill arg3
else
    arg3 = null
If cannot parse the line
```

Note that I am ignoring the details such as how to tell whether there is a first field and how to know when a field token ends. These details have nothing to do with the design at this level of abstraction, and thus are better left out so as not to take our attention from the level we are working on.

Note also that in spite of what I just said our design is complete at the level we are working on. In other words, the details I am leaving for later are at a lower level of abstraction and we can hide them under broad statements like "if there is a third argument" or "fill field1". At this time there is no need to know how to fill field1, but there is a need to know when and from what data to fill it. Now that we have the Line class, we can go back and use it in the constructor for the Symbol_Table. I am also using the File class here.

```
File src_file("src.asm")
```

This line creates a File object called src_file based on a disk file whose name is src.asm. Note that as the pseudocode is expanded it looks more and more like C++.

while there is another line in the source file

This is the next line to deal with, and it brings up the question of how to know

```

while there is another line in the source file
  read in a line from the source file
  parse the line into fields
  if cannot parse the line
    write a message to the error file
    write the current line to the error file
  if there is a symbol definition
    if there is an error in symbol definition
      write a message to the error file
      write the current line to the error file
    put the symbol and its value in the symbol table

```

Figure 1

when we get to the end of the file. It seems reasonable that the File object would be the one to know when the end of file is reached, so let's add that capability to it. Here is the new File class. It has a flag to indicate when the EOF has been reached. The flag is set initially to 0 by the constructor, and is set to 1 by read_line() when it reaches the End of File.

The first line creates a File object with the filename src.asm. This opens the disk file of that name. The next line calls read_line(), a member of the Line object. This function returns a line from the source file which is used to create a Line object which is called curent_line. The line from the source file is passed to the creator for the Line object.

```

class File {
  FILE *fptr;
  int end;
public:
  File(char *fname) {open(fname); end = 0;}
  ~File() {close();}
  void rewind();
  line read_line() {if(EOF) end = 1;}
  void write_line(line);
  int at_end() {return(end);}
};

```

Now we can finish writing the constructor for the Symbol_Table;

```

File src_file("src.asm")
Line current_line(src_file.read_line())
if not src_file.at_end()
  if there is a symbol definition
    if there is an error in symbol definition
      write a message to the error file
      write the current line to the error file
    else
      put the symbol and its value in the symbol table

```

Notice that the constructor is shorter and some of the straight procedural code has been replaced by objects and their member functions.

The creators of the various classes contain the code which gets the data used in its class and does whatever processing it requires. The creator for the Line class gets the line from the disk file and parses it into its fields and arguments. The creator for the Symbol_Table gets the source code from the disk file (using Line objects) and from the data creates an array of symbols and their values.

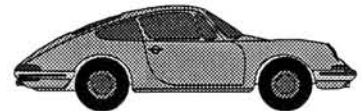
The classes also contain functions used from outside the class to access and process the data, and to determine or change the state of the class. The function at_end() in the File class is used to find out whether the end of the file has been reached. The rewind() function is used to return to the beginning of the file.

In the next installment I will continue this design exercise.

Resources

Assembly Language Primer for the IBM PC, Robert Lafore. New American Library, 1984.

The Practical Guide to Structured Systems Design, Meilar Page-Jones. Yourdon Press, 1988. ✱



MOVING?
Don't miss a single issue of *REM*!

Please let us know 3-4 weeks before you move. Call (616) 982-3463 or write:

Zenith Data Systems
P.O. Box 217
Benton Harbor, MI 48023-0217

All Checks must be made out to
Zenith Data Systems

January 1, 1991

Zenith Data Systems 00.00

Zero dollars and zero/100

S A M P L E

Getting the Most From Your Computer

Part 7

John Lewis
6 Sexton Cove Road
Key Largo, FL 33037



The subject of data base programs is the topic for this month's discussion, but before proceeding I would like to reiterate one point. The purpose of this series of articles is not to present a complete tutorial on programming in Pascal. There are a number of good books available that provide excellent coverage on that facet of learning. My intention is to get readers interested in learning to write their own programs, focusing on Turbo Pascal as the tool of choice.

Instead of examining the minute details of individual code fragments, we will discuss the logic used to construct the most interesting and/or complex modules of our program, leaving the mundane for the individual to investigate. In short, we're going to look at the "big picture", concentrating on the "highlights".

If, in so doing, I can provide some motivation for learning then I will have succeeded in reaching my goal. As incentive, the source code for some rather interesting and useful software will be presented with each installment, while providing an object which can be dissected for study.

The source code included in this article was written for, and tested on, an IBM PC compatible (Zenith 386 SX) and will compile under Turbo Pascal Version 4.0 or later.

If you have any questions pertaining to programming in Turbo Pascal or any of the articles which comprise part of this series, feel free to write me at the address shown above and include a S.A.S.E. if you wish a

reply.

There are a multitude of data base programs available for the PC. Most of them are quite sophisticated (and expensive). If you need a data base program that can be configured for a variety of tasks (from a magazine topic index to a mailing list) and don't wish to apply for a second mortgage, then read on.

This month's listing makes no pretense at being a full blown relational data base, rather a "flat file" program. What it does though, is deliver the goods in a hurry. My own needs are rather modest so I don't have a huge data base to search but I can tell you that our program will search through over a hundred records before you can get your finger away from the "Enter" key. The search "key" used may be ambiguous if desired, if so the program will display each "match" in turn. Another nice feature is that the user can input any part of a key and still enjoy a successful search.

My first attempt at creating a data base program using Turbo Pascal presented one rather large drawback. Actually, the program succeeded quite well in its basic goals but when I wanted to use the data base records produced by my latest creation with another program (written in "C"), one rather large problem presented itself. The records were incompatible. When the program written in "C" tried to display "Pascal" records on the screen, the display went berserk.

The cause for this anomaly was really not too elusive; however, the solution was another matter. When a string is processed

under "C" protocol, a zero (null character) is used to delimit the character array (this is called an "ascii" string, the (z)ero occupying a position following the ascii characters to be printed). If no zero is found, characters will continue to be output. If the screen is the target device, things get very messy awfully fast. Needless to say, a rather undesirable situation. When a string is constructed under the auspices of a Turbo Pascal environment, the first character in the string (subscript zero) is not a character at all, but a length byte. For instance the string: "This is a test" would have, in byte zero, not the ascii character "T", but binary 14 (this incidentally, is the reason that a string in Pascal cannot exceed 255 characters in length). Since the length of a Pascal string can be determined before attempting to print it, no delimiting zero is needed, hence the incompatibility of records.

What to do? The easy solution to the above problem would be to just avoid a situation where compatibility was an issue. Certainly not an impossible task but not very "elegant" either (programmers are always looking for elegant solutions).

Fortunately, if the situation is reversed, the Pascal environment has no trouble in digesting a zero placed judiciously within a string. Thus we can insert a delimiting zero into a Pascal string with no adverse effects. Please remember that this "zero" is not an ascii character (decimal values between 32 - 127) but is a binary zero (null character). An ASCII zero "0" has a decimal value of 48 (30 hexadecimal) and may be printed. A binary zero cannot be printed to the screen

(without conversion to ascii), at least not in the form we are used to. Actually, there is a corresponding character in the extended IBM character set but that is another story.

The next hurdle is almost too easy. When writing records to disk, the Pascal environment omits the length byte (only on files of type "text"), leaving us with a disk record that is compatible with records created under the "C" protocol.

Before we leave the subject of compatible records, let me define the crucial parameters (as they apply to our data base program). Also, let me add that the following information applies when configuring a program to fit an existing data base, regardless of the language used to create the parent program.

First, the record length in file "A" must be the same as the record length in file "B". Each field must match its counterpart, both in length and type (if a field is of type integer, it will be incompatible with a field of type string). Needless to say, the number of fields within the records being compared must also be the same. The easiest, and probably the best way to meet these criteria is to make each "record field" of type "string", converting the strings to integers (or other data type) within the parent program where needed. Given the above and making provision for "asciiz" strings alluded to previously, we can create a program that is truly versatile.

Since we intend to make all our programs as "user friendly" as possible, there is another facet of Turbo Pascal that could use a little help. As we add sophistication to our programs we will be including features which involve the function and cursor control keys. There is no provision made, using the library functions and procedures, to directly decode these keys. We can, of course, use a "kludge" (such as calling readkey twice) to accomplish this for us but that is neither "elegant" or fast. What we need is a function which can perform this job for us. Eureka! Included within "Dbasetoo" is just such a routine (see function "Get_Ch(var s : char):Char" within Listing1).

Although quite short, this function incorporates a number of new facets to programming in Pascal that we have not encountered before. It uses a new data type (registers), calls a "Bios" interrupt (16 in Hex), and uses a "shr" instruction. Since this bit of code involves a fairly high level of complexity and some new terminology, I'm going to break one of my own rules and explore the routine in some detail (don't yawn and turn to the next article, this little rascal is a VERY handy routine).

Let's look at the "Registers" data type first. Turbo Pascal uses a "variant" record type to emulate the CPU registers. By declaring a variable of type "Registers", as in: "Regs : Registers" "Regs.AX" will ad-

dress the cpu register: "AX". Thus we can load the cpu with the statement: "Regs.AX:=0;". Since the 16 bit "AX" register comprises two eight bit registers "AH" and "AL", the above statement loads both "AH" and "AL" with zero.

What is the object of all this cryptic code? Are we going to be communicating with "ET"? or possibly with a "UFO"? Not really.

Using one of the computer's interrupts, whether it be "Bios" or "Dos", involves some rather low level programming and Turbo Pascal is up to the job by providing us with these tools.

The code fragment: "Intr(\$16, Regs);" will pass the value previously loaded into the "AX" register and "Call" interrupt number 16 (Hex, 22 decimal). This interrupt is a keyboard request (returns the next key entered on the keyboard). The value returned is in the "AX" register so all we have to do is examine the contents and act accordingly. If we look at the lower part of the "AX" register (AL), and it contains a zero "If Regs.AL = 0" then the key pressed was a function key. If the register contained a non zero value, it was an ascii character. Now all we have to do is declare that: "FuncKey = True" (assuming a zero in Regs.AL) and then shift the value in "AH" down into "AL" (Regs.AX:=(Regs.AX shr 8)). This statement shifts the value in the "AX" register 8 bits to the right (right into the "AL" register). Now we can exit the "If" statement and declare "S:=Char(Regs.AL);". Quick and neat, might I add "elegant" to that summation? Take another close look at the listing where function "Get_Ch(var s : char):char" is shown. I hope you agree that this will be a valuable addition to your library of Turbo Pascal source code.

I'm sure you have noticed that the enclosed program is not a complete data base. It is actually just the first installment on what will become a fairly sophisticated piece of software. The complete source code, in its present configuration, is nearly 16 Kilobytes. A bit too large to include in one issue of this magazine, and much too large to explain at one sitting.

I have broken the program up into Turbo Pascal units, making my task of presentation easier and your job of testing and debugging less arduous.

Now let's look at the function "Read_Str(var str : String; Max : Integer):String;". This routine accepts a character at a time, constructing an ordinary string which is then converted into an "asciiz" string by substituting a zero (null) character for the carriage return (13). Placing the null character (0) just behind the last character input. By monitoring the length of the string, it also ensures that it will become no longer than the length defined in the constant declarations. Through the use of a while loop in conjunction with a

"case" statement, it provides an editing capability as well. If the user inputs a backspace, the cursor obligingly backs up and prints a space over the last character. Then, using the Turbo library routine "copy", it deletes the previous character in memory.

Procedure Write_File utilizes a familiar routine that we first saw in our directory sorting program (part five of this series) and then again in part 6 when doubly linked lists were employed to provide a means of perusing through files. Now we are using "P:=P^.Next" to advance through the data base stored in memory.

Each record in the file is treated as a line and an entire record is written at one time, employing "Write"'s ability of accepting multiple arguments (P^.Field1, P^.Field2 etc.). Here we had to be a little bit devious. The writeln(argument list) command inserts a carriage return after every line, but "write" does not. Since each record MUST contain a precise number of characters (by definition), we can't have any extraneous characters cluttering up our space.

If our only concern was adding new records to an existing list, we could merely append the records being added to our data base file. However, since we must have the ability to edit records created previously, we need the capability of altering fields in the middle of our file. That feature is not allowed on files of type text. Did we back ourselves up into a corner? Almost, but there is a way out. We'll make our changes in memory and then write the entire (altered) memory image back to disk, changes included. Due to the fact that we must be able to access the data file up to the moment we save the new image; we can't open the existing file in a mode which will allow us to write to it. Hence, we write the new data to a temporary file, erase the old file, and then rename the temporary file to the old file name. Sounds a bit convoluted but it works. Look closely at the bottom of Write_File and compare the statements with the logic just described. There should be a marked similarity.

Nearly all of the criteria have been met towards creating a compatible data file. One rather critical parameter remains. We have not made provision for record fields that may fall short of being the size dictated by the record definition. In other words, if a user inputs 15 characters into a field that is defined as 25 characters, then something must be done to "make it fit". Remember the compatibility criteria which were described earlier. Actually, all we have to do is add some "fill" to the end of the field in question.

Function "Padrec" to the rescue. Procedure "Create_Record" passes an integer to "PadRec" that is a measure of the number of characters that need to be tacked on to the field being processed to bring it up to "spec". PadRec then assembles a string

composed of spaces which "Create_Record" dutifully appends to the current field.

We have just crafted a text file which is completely compatible with files created under "C" protocol. This same file will respond to the "type" command under Dos by displaying itself in a manner commensurate with, what else, text files.

Unfortunately, there is not a whole lot you can do with this program as it stands, but it is only the beginning of what should be a very welcome addition to your software inventory. You can test this module by using option four from the menu (the only option implemented so far) by creating some records and then using the "type" command to display them. I would recommend using MS-DOS "Debug" to take a closer look at your creation. Paying particular attention to the imbedded zeros (null characters). After all, we expended a great deal of effort in placing them there!

Please notice that this month's listing is actually the source code for a Pascal program (fragment) as well as a Turbo Pascal unit. The module containing the program file will be modified in the next article and another unit will be added. The unit: "unitinpt" will not need any changes. My objective is to present the code in three installments. The next article will complete the basic data base program and the third installment (relative to data bases) will add some dressing.

Remember the windows I mentioned some months ago? Not MS-DOS Windows 3.0 but your own "pop-up" windows that will appear or disappear at the touch of a key. We'll add a touch of class and instructions for using the data base record editor at the same time.

As a point of interest to those of you wishing to acquire the code to some rather sophisticated software; the editor I'm using to write this article was written in Turbo Pascal and incorporates the above described data base program with the ability to "cut and paste" records into the text being created. In other words I can add the company name and address to a letter I'm working on without leaving the editor to look it up. Also on tap at the touch of a (Function) key is the calculator featured in part two of this series. Both of these utilities appear in a non-destructive window that obligingly disappears after use.

Yes, I do intend to present the above in future installments, but I'm getting ahead of myself. The quickest way to acquire the knowledge needed to generate new code is to understand completely the logic used in the programs thus far covered, and the best way to accomplish that is to dissect and analyze each routine that is not clear.

Now that you have something to keep you busy, I'll leave you 'till the next installment.

Listing 1

```

Program Dbasetoo;
uses crt, unitinpt;

begin
OpenFile;
if Newfile <> True then ReadFile;Find:=2;Modified:=False;
While Option <> 6 do begin
clrscr;esthetics;
gotoxy(5,2);write('Please enter...');
gotoxy(20,4);write('1. Re-define Search field, default = ',Field2);
gotoxy(20,6);write('2. Search for/display/edit record. ');
gotoxy(20,8);write('3. View window into database. ');
gotoxy(20,10);write('4. Create new record. ');
gotoxy(20,12);write('5. Change sort field, default = ',Field2);
gotoxy(20,14);write('6. Exit to operating system');
gotoxy(17,16);write('the number of your choice. ');
Readln(option);
case option of
1..3:begin NotImplemented;end;
4:begin create_record;end;
5:begin NotImplemented;end;
end; { end case option of }
end; { end of while }
Option:=0;If modified = True then Write_File;
Textcolor(LightGray);TextBackGround(Black);
window(1,1,80,25);clrscr;
end.

{ ***** End of listing for Dbasetoo.pas ***** }
{ ***** Listing below is for "Unitinpt.Pas", Each listing ***** }
{ ***** Must be compiled separately ***** }

unit unitinpt;
interface
Procedure Create_Record;
Procedure OpenFile;
Procedure ReadFile;
Procedure Esthetics;
Procedure NotImplemented;
Procedure Write_File;
Function Get_Ch(var s : Char):Char;
Procedure Init;
Procedure Rest;
Function PadRec( s : String;n : Integer):String;

Const
Len1 = 25;
Len2 = 25;
Len3 = 35;
Len4 = 32;
Len5 = 11;
Len6 = 20;      { unused in this implementation }
Len7 = 20;      { ditto }
Len8 = 20;      { ditto }

Field1 = 'Company'      { Definition of individual record }
Field2 = 'Name'; { fields, makes changes very easy }
Field3 = 'Street';      { to implement }
Field4 = 'City, State, Zip';
Field5 = 'Phone';
Filename : String[15] = 'DataBase.dat';
BakFile : String[15] = 'Tempfile.dat';
Found : Boolean = False;
All : Boolean = False;
NewFile : Boolean = False;
Mark : Integer = 20;    { Defines beginning of screen field }
Row1 : Integer = 2;    { Defines row within window }
Row2 : Integer = 4;
Row3 : Integer = 6;
Row4 : Integer = 8;
Row5 : Integer = 10;

Arrow : String[3] = chr(175)+chr(45)+chr(62); { assemble an arrow }
Eras : String[3] = ' ';

Up_Limit : Integer = 2;
Bot_Limit : Integer = 10; { Must be changed if No. of lines is changed }

{$DEFINE SHORT}      { Example only }

Type
String79 = String[79];
String20 = String[20];

```

```

ptr[max]:=chr(0); { make sure last character is }
c:= #0; { null if string becomes max size }
end;
end; { end while }
Read_str:=str;
end;

Procedure OpenFile;
Begin
assign(ObjectFile,Filename);{$I-}
reset(ObjectFile);{$I+};if IOresult <> 0
then Begin Rewrite(ObjectFile);NewFile:=True;End;
End;

Procedure Write_File;
begin
Records:=1;P:=start; { Write records to text file }
Assign(TempFile,BakFile); { stepping through linked list }
{$I-} Rewrite(TempFile); {$I+}; { using pointers }
Reset(ObjectFile);
repeat
write(TempFile,P'.Field1,P'.Field2,P'.Field3,P'.Field4,P'.Field5);
P:=P'.Next;Records:=Records+1;
Until (P'.Prev'.Next = Nil) or (Records >= Limit);
close(TempFile);Close(ObjectFile);Erase(ObjectFile);
Rename(TempFile,Filename);
end;

Procedure Init;
Begin
new(p);Start:=P;P'.Prev:=Nil; { define beginning of list }
P'.Next:=Nil;RunP:=P;Limit:=1;
end;

Procedure Rest;
Begin
New(p);P'.Prev:=RunP; { link each succeeding record }
RunP'.Next:=P;P'.Next:=Nil;RunP:=P { with previous and next }
Limit:=Limit+1;
end;

Procedure ReadFile;
Begin { Read file and link }
Init;
While Not Eof(ObjectFile) do
Begin
Read(ObjectFile,P'.Field1,P'.Field2,P'.Field3,P'.Field4,P'.Field5);
Rest;
End;
end;

Function PadRec(s : string;n : Integer):String;
Var I : Integer;
Begin { Pad each field of record }
s:='';for I:=1 to n do { to size dictated by definition }
s:=s+' ';
PadRec:=s;
End;

Procedure Print_UnderLn(n : Integer);
Var I : Integer;
Begin { Used to enhance screen display }

```

```

Ptr = 'Datafile;
Datafile = Record { Turbo Pascal record defined }
Next : Ptr; { bears strong resemblance to }
Prev : Ptr; { structure in "C" }
RecNum : Integer;
Field1 : String[Len1]; { parameters established through }
Field2 : String[Len2]; { correlation to constants listed }
Field3 : String[Len3]; { above, makes a new configuration }
Field4 : String[Len4]; { very easy }
Field5 : String[Len5];
{$IFDEF LONG} { Ifdef statements can be used to }
Field6 : String[Len6]; { help with defining a new shell }
Field7 : String[Len7];
Field8 : String[Len8];
{$ENDIF}
end;

Var
Tempfile, Objectfile : Text; { File to be of type text }
E_nd, P, Start, Last, Pre, RunP, Nxt : Ptr; { see article text }
Find, Option, Limit, Records : Integer;
Modified, Menu, Funckey : Boolean;
Key : String20;
Ch : Char;
String : String;

implementation
uses crt, dos, printer, boxunit;
{ ***** }
Function get_ch(var s : Char):Char;
Var
Regs : Registers;
begin Funckey:=False { reset value }
Regs.AX:=0; { Bios call }
Intr($16, Regs);
If Regs.AL = 0 Then { Function key }
Begin
Funckey:=True;
Regs.AX:=(Regs.AX shr 8); { shift scan key value to AL }
end;
S:= Char(Regs.AL);
get_ch:=S;
end;

Function Read_Str(var str : String;Max : integer):String;
Var
c : char;
Len : Integer;
begin
str:=''; { This function creates an asciiz string }
while( c <> #0) do { utilizing Get_ch(c) for keyboard input }
begin { also provides for editing string input }
c:=get_ch(c); { using backspace key }
case c of
#32..#126: begin write(c);str:=str+c;end;
#13 : begin c:=#0;str:=str+c;end;
#8 : begin Len:=Length(str);
If Len > 0 then begin write(c);write(#32);write(c);{print bkspace, print space}
str:=copy(str,1,Len-1);end;end; { and then back up again }
end; { end case c of }
if (Length(str) >= max) then
begin

```

W S Electronics

(513) 376-4348

**** Since 1975 ****

(513) 427-0287

1106 State Route 380, Xenia, Ohio 45385

W S ELECTRONICS INTRODUCES KRIS TECHNOLOGIES

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

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

\$3550.00

and

\$2899.00

System 38M
80386-33MHz CPU
64K cache

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

\$1999.00

System 38
80386-35MHz CPU

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

\$1499.00

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

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

\$1225.00

and

\$1025.00

System 28
80286 60-bit CPU

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

\$750.00

ALL COMPUTERS HAVE A 2 YEAR PARTS, 18 MONTH LABOR WARRANTY

*Monitors, Video Cards, Harddrives are optional.



```

For I:=1 to n do { in input module I
write(' ');
write(' ');
end;

Procedure Create_Record;
Var Len : Integer; { data entry module }
Spaces : String99; { uses Read_Str & PadRec }
Const Col : Integer = 8; { "fills out" each field }
Col2 : Integer = 15; { to size defined in record }
begin
clrscr;gotoxy(5,2);write('Data record entry module...');
if Newfile = True then Init;
gotoxy(Col,4);write('Enter ',Field1,' ');Print_UnderLn(LEN1);
gotoxy(Col,6);write('Enter ',Field2,' ');Print_UnderLn(LEN2);
gotoxy(Col,8);write('Enter ',Field3,' ');Print_UnderLn(LEN3);
gotoxy(Col,10);write('Enter ',Field4,' ');Print_UnderLn(LEN4);
gotoxy(Col,12);write('Enter ',Field5,' ');Print_UnderLn(LEN5);

Len:=Length(Field1);gotoxy(col2+Len,4);
String:=Read_Str(String, LEN1);P.Field1:=String;
Len:=Length(P.Field1);Len:=LEN1-Len;Spaces:=PadRec(Spaces,Len);
P.Field1:=P.Field1+Spaces;

Len:=Length(Field2);gotoxy(col2+Len,6);
String:=Read_Str(String,LEN2);P.Field2:=String;
Len:=Length(P.Field2);Len:=LEN2-Len;Spaces:=PadRec(Spaces,Len);
P.Field2:=P.Field2+Spaces;

Len:=Length(Field3);gotoxy(col2+Len,8);String:=Read_Str(String,LEN3);
P.Field3:=String;
Len:=Length(P.Field3);Len:=LEN3-Len;Spaces:=PadRec(Spaces,Len);
P.Field3:=P.Field3+Spaces;

Len:=Length(Field4);gotoxy(Col2+Len,10);String:=Read_Str(String,LEN4);
P.Field4:=String;
Len:=Length(P.Field4);Len:=LEN4-Len;Spaces:=PadRec(Spaces,Len);
P.Field4:=P.Field4+Spaces;

Len:=Length(Field5);gotoxy(Col2+Len,12);String:=Read_Str(String,LEN5);
P.Field5:=String;
Len:=Length(P.Field5);Len:=LEN5-Len;Spaces:=PadRec(Spaces,Len);
P.Field5:=P.Field5+Spaces;

Rest;
Modified:=True;
end;

Procedure Esthetics;
Begin { NOTE!: The clrscr command has been }
TextBackground(lightgray);TextColor(Black); { removed from Boxunit }
clrscr;drawbox(5,3,75,24);window(6,4,74,23); { clrscr added here for }
textbackground(blue);textcolor(yellow);clrscr; { modified boxunit }
end;

Procedure NotImplemented;
begin
clrscr;gotoxy(5,8);
write('This option not implemented yet, press enter to continue ');
readln;
end;
end.

```

I Can Do It

Do Math Output Data Loop Structured Programming

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

We're back, and will get into full swing with some interesting work this month. In my last article, I pointed out that a lot of instructors give beginning programming students the job of printing a series of numbers, followed by those numbers squared, doubled, or something like that. This exercise, while it may seem trivial at first, has a lot of value in teaching programming and programming options. At the top of this article, I listed three (3) functions which your computer can perform. You should remember them as the last thing we discussed last month. Then I sneaked in a scary term which many programmers (unfortunately, this includes some very accomplished code writers) either have not heard of, were scared away from, or chose not to use.

Structured programming is not difficult. It is not some special program or set of instructions. Rather, it is a mind set, or a way of writing your program to make it easier to work on at a later date if you should choose to make modifications to it. But enough of that for now. It will all happen so automatically for you that I'll have to point out that we've done it. Many of us do so now, but just don't call our method 'structured' as that's supposed to be scary or difficult for some unknown reason.

For starters, we know that to print the numbers one (1) to ten (10), and to print their squares behind them requires that we:

Output Data — Do the printing

Do Math — Calculate the squares

Loop — Do the above two things for each number

Rather than get ahead of ourselves, and rather than try to learn three things at one time, let's do one of them. So that we can see the results of our work, let's output data. We can work in the BASIC language

for this, and we can do the simplest of things first. Let's print the numbers one (1) to ten (10) on the screen of our computers.

We'll do our work in the least efficient way possible at first. This is not to say that we don't all know a better way to do it. However, it is important to remember the simplest, most stripped down, version of performing a task as this is often the fastest and easiest way to write and to execute a program to see the results of something we're trying to do. So, now we'll write a separate program line to print each number, 1 to 10, on our computers' screen. The first step is to get our BASIC program manual. You know the one. It probably came with your computer. If you bought BASIC as a separate software package, it came with that. Yes, that's the one. It says GWBASIC or BASICA or just plain BASIC on it.

You've just been introduced to one of my hangups. It's probably the biggest of them all. **USE YOUR MANUAL!!!** It's like gold in the bank. Now turn to the page which explains the PRINT statement. Read that part of the manual. It will tell you that the PRINT statement will output data to the screen, and should show you several examples of doing that. So, you can see that we could write our program like this:

```
10 REM PRINT THE NUMBERS 1 TO 10
20 PRINT 1
30 PRINT 2
40 PRINT 3
50 PRINT 4
60 PRINT 5
70 PRINT 6
80 PRINT 7
90 PRINT 8
100 PRINT 9
110 PRINT 10
```

Now turn on your computer and enter the short program above. When you've got it all typed in, save it as any file name you like. I used the name NUM01.BAS. Having done this, run the program. You'll see that

the numbers 1 through 10 appear on the screen. So big deal, you say. What have we done? Well, we used an output-type function of the computer. In fact, we used it ten times. And each time we used it was just as easy as any other time. We wrote and executed a program which would at first appear to be ten times as long, and thus, ten times as complex as a program which only printed the number 1. But we know better because we wrote it. I'll keep pointing this out to you as we go along. Even though this was an extremely simple example, the same principal applies to large programs like word processors. They're all just a lot of small easy steps. The trick is to coordinate each of those steps so that they are performed in the correct order. That's what you'll learn very soon, so don't be dismayed at the simplicity of the examples. We don't want complex algorithms to get into the way of learning to program well.

Now that we've performed an output function, let's enhance it by adding the squares of each number to that output. Reading from the PRINT statement instructions in our BASIC manual, we see that if we separate two items by a comma (,) they will be placed in separate columns when printed on the screen.

In order to calculate the square of a number, we must now add some math to our program. The index of your BASIC manual should list 'arithmetic', 'arithmetic operators', or 'operators'. If you look those items up, you will see the math functions which your version of BASIC supports. There are two operators which can do the job we want. They are:

- multiplication *
- exponentiation ^

In order to calculate the square of a number, you must multiply it by itself. To multiply the number 5 by itself in the BASIC language, you would write:

5 * 5

A second method, called exponentiation, allows you to raise a number to any power. The power two means writing the number down twice and multiplying it as we did above. The power three would multiply the number three times, thus calculating the cube of the number. It would be the same as writing:

5 * 5 * 5

To use the exponentiation method (you'll see why I prefer it later), you would calculate the square of the number 5 by writing:

5 ^ 2

That little arrow pointing toward the sky is an up caret and is located above the number six (6) on your keyboard. All this information allows us to write the following program. As you key it into your computer, remember that the comma will place our output in columns, and the ^2 will calculate the square of each number. Here's our program:

```
5 REM THIS IS THE FILE NUM02.BAS
10 REM PRINT NUMBERS 1 TO 10 AND SQUARES
20 PRINT 1,1^2
30 PRINT 2,2^2
40 PRINT 3,3^2
50 PRINT 4,4^2
60 PRINT 5,5^2
70 PRINT 6,6^2
80 PRINT 7,7^2
90 PRINT 8,8^2
100 PRINT 9,9^2
110 PRINT 10,10^2
```

Having run the program, we know that it works and works well. There's one more thing I'd like to point out in the program above. That's lines 5 and 10. Each line begins with 'REM'. That tells us that these are REMARKS. They are there for the programmer to read. The computer ignores them, yet, they're extremely important. The two things which I showed here are:

1. The name of the program
2. The function of the program

However, you can add any notes you want in a REMARK statement. Some important things include the date the program was written and the name of the author. Turn to the REM statement in your BASIC manual and read it. Note that we can use a separate statement as we did above, or add a note to the end of a line by using a single quotation mark. Our final program in this month's article shows how thorough you should learn to be with REMARKS. Even though you may pick this program up several years after having written it, you'll be able to read and to understand it because the REMARKS are there. As our programs become more complex, this principal really increases in importance.

Now that we've done what we wanted by brute force, it's time to introduce a little bit of elegance into our program. Rather than repeat a single instruction ten times (more times if we wanted more numbers), wouldn't it be nice if we could write that instruction once and then use it as many

times as we needed. Indeed we can do that and we'll do it now with a loop.

The loop we'll use is FOR...NEXT. That's right, turn to FOR...NEXT in your BASIC manual and read it. We'll use something called a 'numeric variable' which we'll call 'X'. That is, we'll use the letter X to represent a number. First, we'll tell it to represent the number one (1), and execute the steps required to print that number and its square. When that's been done, we'll tell X to represent the number two (2) and produce another printout. This process will continue until X represents the number ten (10). The program will then be permitted to end. Our new program with the loop (FOR...NEXT) added to it looks like this:

```
10 REM NUM03.BAS
20 REM DEMO PROGRAM TO PRINT 1 TO 10 AND
SQUARES WITH A LOOP
30 FOR X = 1 TO 10
40 PRINT X, X^2
50 NEXT X
```

Type the program into your computer and run it. Note that lines 30 and 50 provide the control for the loop. Line 30 tells us what numbers to use, and line 50 tells we're at the end of the steps we wanted to perform, and should go back to line 30 (the FOR line) with the next value of X. This keeps happening until the value of X reaches the top of the range we specified. That is, when X becomes 10, the program continues after the NEXT statement. If we had more instructions after line 50, the program would execute them at this time.

Now we have a better, more compact program. It was much easier to write. This small five-line exercise could be part of a much bigger program, however. Now, let's suppose that we want to perform this same job several times within that larger program. Must we rewrite these instructions every time we want to use them? No, certainly not! What we can do is tuck them away in a safe spot (that is, another area of the program) and use them any time we like.

When we put a group of instructions together like this, we call them a sub-routine. That is, a small part of the much larger routine which is the whole program. To use this sub-routine, we use the GOSUB...RETURN statement. Look up GOSUB in your BASIC manual and read it. Our sub-routine is written by placing it in a separate area of the program, away from the main set of instructions. We'll put it in the area beginning on line 5000. When we write a sub-routine, we write each line just as we would within the main body of the program. Then we add a last line which reads RETURN.

To use this sub-routine, we add the instruction GOSUB 5000. This tells the program to jump to line 5000 and do what it finds there. When the computer encounters the RETURN statement, it returns to the line immediately after the one on which it found the GOSUB. The real reason for

doing this, and the awesome programming power it offers will become apparent in the last program of this article, but let's not jump too far, too fast. First, we'll get our program to run with a sub-routine in it, then we'll move on.

Our program looks like this:

```
10 REM NUM04.BAS
20 REM DEMO SHOWING A GOSUB TO A LOOP
FUNCTION
30 GOSUB 5000
40 END
5000 FOR X = 1 TO 10
5010 PRINT X, X^2
5020 NEXT X
5030 RETURN
```

Note that lines 10 and 20 are still the REMARKS. Line 30 tells the program to go to the sub-routine found at line 5000 and do what's there. When the sub-routine gets to the RETURN instruction, the program is sent back to line 40. There we encounter a new statement called END. This tells the program to stop here. If we did not have the END statement, when the program came back from the sub-routine, the next line to be executed would be line 5000. From now on, we'll always use the END statement to stop our programs. Oh yes, be sure to read the END statement in your BASIC manual. Now type the program NUM04.BAS into your computer and run it.

You see, the GOSUB worked perfectly. Our simple program found this set of instructions, executed them, and came back to the END statement. Now we can print the numbers 1 to 10 and their squares any time we want, as often as we like. All we have to do is add the instruction GOSUB 5000. But what if we want the numbers 6 to 12 or 5 to 20 and their squares? Do we have to write a separate sub-routine for every possibility? Or, could we somehow modify our sub-routine so that it would be more flexible? Could we make it start and stop with any numbers we want?

We must be able to do just that, or I wouldn't have asked. Let's introduce two new numeric variables into our program. We'll call them 'A' and 'B'. We'll let A be the number we start with, and we'll let B be the number with which we finish. When we write our sub-routine, we'll put the A and the B where the numbers 1 and 10 were in the FOR statement. Then all we have to do is tell the program what we want A and B to be worth, and call the sub-routine as we did before. It looks like this:

```
10 REM NUM05.BAS
20 REM DEMO SHOWING PASSING INFO TO A
FUNCTION
30 A = 1: B = 10: GOSUB 5000 'PRINT 1 TO
10 AND SQUARES
40 END
5000 FOR X = A TO B
5010 PRINT X, X^2
5020 NEXT X
5030 RETURN
```

Notice that line 30 has several instructions on it, rather than the one instruction per line we've used in the past. This is allowed in BASIC if you separate each

instruction with a colon (:). I did it here because setting the values of A and B, and calling the sub-routine were all related to one operation, and I documented that to the right. Notice also that a single quote was used to separate the REMARK from the instructions your computer will execute. So go ahead, type NUM05.BAS and run it. You'll see that it works just fine.

Now some of the power I talked about earlier has become evident. It's possible to write a set of instructions to perform a certain task, to make those instructions flexible, to call them any time we want them, and to give them different sets of values to work with. That's power, and what we've done is STRUCTURED PROGRAMMING. It's not so bad after all. We set up a structure in which our program is controlled by a small set of instructions, and in which other sets of instructions we've called sub-routines that perform certain specific tasks are located in a separate area. Yet they're accessible to the whole program.

Part of the beauty of this style of programming is that the program has become more serviceable. We can read the instructions controlling the operation of the program without wasting time worrying over the details of how each sub-routine works. If we need to work on or to

```

10 REM *****
20 REM NUM06.BAS
30 REM WRITTEN BY DAVID E. WARNICK
40 REM MAY 06, 1990
50 REM *****
60 REM PROGRAM DESCRIPTION
70 REM THIS IS A DEMONSTRATION PROGRAM WHICH SHOWS THE USE OF A STRUCTURED
80 REM APPROACH TO THE WRITING OF PROGRAMS WRITTEN IN THE BASIC LANGUAGE.
90 REM LINES 10 THROUGH 990 ARE ASSIGNED TO GENERAL COMMENTS
100 REM LINES 1000 THROUGH 4990 ARE FOR PROGRAM CONTROL
110 REM LINES 5000 AND UP ARE ASSIGNED TO SUB-ROUTINES
120 REM A LIBERAL USE OF COMMENTS IS USED TO SHOW THEIR VALUE AND TO
130 REM ENCOURAGE OTHER PROGRAMMERS TO DO SO ALSO.
140 REM THE OPERATION OF THIS PROGRAM IS DESIGNED TO PRINT A COLUMN OF
150 REM NUMBERS AND THEIR SQUARES ON THE SCREEN. THIS PROGRAM DEMONSTRATES
160 REM THE USE OF A SUB-ROUTINE TO PERFORM THAT FUNCTION AND THE USE OF
170 REM MULTIPLE GOSUB STATEMENTS TO MAKE SEVERAL CALLS TO THAT SUB-
180 REM ROUTINE
190 REM ROWS OF ASTERISKS ARE USED TO DIVIDE PROGRAM SEGMENTS, THUS
200 REM ADDING TO THE READABILITY OF THE PROGRAM.
210 REM *****
1000 A = 1: B = 10: GOSUB 5000 'PRINT 1 TO 10 AND THEIR SQUARES
1010 A = 6: B = 12: GOSUB 5000 'PRINT 6 TO 12 AND THEIR SQUARES
1020 A = 5: B = 20: GOSUB 5000 'PRINT 5 TO 20 AND THEIR SQUARES
1030 END
5000 REM *****
5010 REM THIS IS A SUB-ROUTINE WHICH PRINTS A SERIES OF NUMBERS AND
5020 REM THEIR SQUARES ON THE SCREEN. THE RANGE OF NUMBERS PRINTED
5030 REM IS DETERMINED BY THE VALUES OF NUMERIC VARIABLES 'A' AND
5040 REM 'B'. THE PROGRAM WHICH CALLS THIS SUB-ROUTINE MUST FIRST
5050 REM SET THE VALUE OF 'A' TO THE FIRST NUMBER TO BE PRINTED AND
5060 REM THE VALUE OF 'B' TO THE LAST NUMBER TO BE PRINTED BEFORE
5070 REM MAKING THE CALL.
5080 REM *****
5090 FOR X = A TO B 'FOR THE RANGE OF NUMBERS SET BY THE PROGRAM
5100 PRINT X, X^2 'PRINT THE NUMBER AND ITS SQUARE IN COLUMNS
5110 NEXT X 'CONTINUE THE LOOP
5120 RETURN 'GO BACK TO THE CALLING PROGRAM
5130 REM >>>>>>>>>>>> END OF SUB-ROUTINE <<<<<<<<<<<<<<<<

```

An Invitation to Join W/Pug, the WordStar Processing Users' Group

W/Pug is the only independent organization DEDICATED to help YOU.

- Help you with tutorials on WordStar and companion programs.
- Help you with "Tips & Tricks" to use WordStar more effectively.
- Help you use WordStar in business and desktop publishing.
- Help you select new products and services to enhance WordStar.
- Help you exchange information with other WordStar users.
- Help you use DOS and Windows.

Membership includes: (1) a free subscription to *Scroll*, our 24 page bi-monthly newsletter; (2) free access to W/Pug's large library of programs which complement WordStar; (3) a free library disk; (4) special discounts on hardware and software.

Send for Information: W/Pug
 PO BOX 16-1443
 Miami, FL 33116-1443

change a sub-routine, we know where to find it (at line 5000, in our case). I cannot overstate the importance of this. It's the singular item I find most lacking in the majority of programs I have read. It's not too bad when you first work on something, or if the task is small, but just come back a year later or try to work on a big program and you'll be lost. The answer is simple. Be consistent, and arrange your program so that it's serviceable.

You've learned a very big and a very important lesson this month. I'll leave you with a final, slightly larger version of this program. It has lots of internal documentation (REMARKS) so it's easy to read. Type and run it, then save it as an example of what we'll do in the future. Maybe sometime we'll come back and add a sub-routine to let the computer operator enter the values at which we want to start and stop our list. Have fun, play with this, and we'll start next month by making our own programmers notebook. Now here's the last program. ✻

Binary Conversion by Division: Why it works

Gil Hoellerich
2617 Country Way
Fayetteville, AR 72703

Early in your computer education, you learned that computers use binary numbers internally instead of the decimal numbers we use.

Your next question probably was: Well, how do I convert from decimal into binary? This question had not one but two answers! There are two different methods of conversion: the subtraction method and the division method.

While both methods convert correctly, there are some differences. The subtraction method produces the left bit (or most significant bit) first while the division method produces the right (or least significant bit) first.

The subtraction method is easier to understand but more difficult to use. For this method you must know or have ready access to a table of the powers of two while converting; so this method becomes difficult when converting large decimal numbers.

The division method is easier to use since you start with the decimal number and successively divide by two until the quotient becomes zero. The remainders of zero and one become the value of the bits of the binary number.

I have never seen any material which explains why the division method works! I have worked through an explanation that satisfies me and perhaps others who like to know why may be interested.

We will start with eight decimal numbers for which we know the binary equivalent and analyze these to determine some rules. Then we will see how the division method satisfies these rules. We will then try the division method on a large decimal number.

To start the process, the eight decimal numbers 8 through 15 and their binary equivalents will be shown. See Figure 1.

You will notice that the binary numbers in the left column end in 0 while the binary numbers in the right column end in

1. Is there a reason for this? Yes! You will notice that decimal numbers of the left column are even (divisible by 2) while the decimal numbers of the right column are odd (not divisible by 2).

Positional Value->		8 4 2 1		8 4 2 1	
8	1 0 0 0	9	1 0 0 1		
10	1 0 1 0	11	1 0 1 1		
12	1 1 0 0	13	1 1 0 1		
14	1 1 1 0	15	1 1 1 1		

Figure 1

Before we look at the next bit of these binary numbers, we need to give each of the 4 bits of the binary number a designator, so you will know which bit of the binary number is being discussed.

We will call the bit at the end or at the extreme right the LSB (least significant bit) and reference the others to that bit. We use the LSB because, as mentioned above, the division method produces the LSB first. We will move to the left from the LSB.

So the bit immediately to the left of this will be LSB+1; the bit to the left of the LSB+1 bit will be the (LSB+1)+1 or LSB+2 bit; and the left bit will be the (LSB+2)+1 or LSB+3 bit. Thus the binary equivalent of decimal 10 is:

LSB+3	LSB+2	LSB+1	LSB
1	0	1	0

So we have our first rule:

Rule 1. If the decimal number is divisible by 2, the LSB is 0; if not, the LSB is 1.

Before we leave the LSB, notice that we divided the LSB by the LSB+1 positional value of 2.

Now lets move to the LSB+1 (the bit immediately left of LSB). To do this we will disregard the LSB and the remaining binary numbers on each line are then equal:

Notice that disregarding the LSB is the same as subtracting zero or one from the

decimal number. This value is shown in parenthesis above. Now we will divide by 4 (the positional value of LSB+2) and if the number is divisible by 4 we enter a 0 in the LSB+1 position; if not, we enter a 1. So both 8 and 9 have a zero while both 10 and 11 have a 1 in that position, etc.

(8)	8	1 0 0 0	(8)	9	1 0 0 1
(10)	10	1 0 1 0	(10)	11	1 0 1 1
(12)	12	1 1 0 0	(12)	13	1 1 0 1
(14)	14	1 1 1 0	(14)	15	1 1 1 1

Figure 2

Rule 2. If the decimal number minus the value of the LSB is divisible by 4, the LSB+1 is 0; if not the LSB+1 is 1. Notice that we divided by the positional value of (LSB+1)+1.

Now lets move to the LSB+2 position. To do this we will disregard the two previous bits. When we disregard the LSB+1, we must subtract 2 from the remaining decimal number, if we entered a 1. So our original table looks like this (the value in parenthesis is the decimal number after we subtract the value of LSB and LSB+1):

(8)	8	1 0 0 0	(8)	9	1 0 0 1
(8)	10	1 0 1 0	(8)	11	1 0 1 1
(12)	12	1 1 0 0	(12)	13	1 1 0 1
(12)	14	1 1 1 0	(12)	15	1 1 1 1

Figure 3

Now we will divide by 8. Using the number in parenthesis the first two rows are divisible by 8 and the last two are not. So we enter a zero for the LSB+2 position in the first two rows and a one in the LSB+2 in the last two rows.

Rule 3. If the decimal number minus the value of LSB and LSB+1 is divisible by 8, the LSB+2 is 0; if not, the LSB+2 is 1. Notice again that 8 is the positional value of (LSB+2)+1!

Now for the left bit or LSB+3. While not shown on our table, the positional value of LSB+4 (LSB+3+1) is 16. So we must divide the decimal number by 16! Also note that when we discard the LSB+2 bit (0 or 1), that all our decimal numbers are 8! Since 8 is not divisible by 16, LSB+3 is 1 for all the numbers.

Rule 4. If the decimal number minus the value of LSB, LSB+1, LSB+2 is divisible by 16 (the positional value of LSB+4), the LSB+3 is 0; if not, the LSB+3 is 1.

We have 4 rules now to guide us in converting a decimal number up to 15 or 4 bits. If we can think abstractly enough, we can form one general rule which will allow the conversion of larger decimal numbers, as well.

General Rules — Each bit is determined in this manner:

1. Subtract from the decimal number the values of the bit times their positional value for each of the bits to right of the bit under consideration.
2. Then divide the result by the positional value of the bit immediately to the left of the bit under consideration.
3. If there is no remainder, the value is 0; if a remainder, the value is 1.

Let's try the decimal number 10 and see if it works:

$$10/2 = 5 \text{ with remainder } 0$$

So this means that it is divisible by 2. So we have the LSB as 0. (Since this is the LSB, there no bits right of the bit under consid-

eration and nothing needs to be subtracted.)

Now we need to get LSB+1. So we subtract the LSB (0) from 10 and still have 10! We now divide 10 by the positional value of (LSB+1)+1 or 4;

$$10/4 = 2 \text{ with remainder } 2$$

So we have the LSB+1 as 1.

Now we need to get LSB+2. So we subtract the sum of LSB (0) and LSB+1 (2) from 10 and we have 8. We divide 8 by the positional value of LSB+3 or 8:

$$8/8 = 1 \text{ with remainder } 0$$

So we have the LSB+2 as 0.

Now for LSB+3. 10 minus the sum of LSB (0), LSB+1 (2), and LSB+2 (0) is 8. Dividing by positional value of LSB+4 or 16 we find that this division is not possible. So we have LSB+3 as 1.

Now for LSB+4. 10 minus the sum of LSB (0), LSB+1 (2), LSB+2 (0), and LSB+3 (8) is

zero! So this tells us that there will be no fifth bit or LSB+4 for the decimal 10.

So the binary of 10 is 1010.

This is almost the division method! We need to make two more observations. First, instead of dividing the original decimal number by the positional value of the bit

Division result	General Rule above	Binary
10/2 = 5 R 0	10/2 = 5 R 0	LSB=0
5/2 = 2 R 1	10/4 = 2 R 2	LSB+1=1
2/2 = 1 R 0	(10 - 2)/8 = 1 R 0	LSB+2=0
1/2 = 0 R 1	(10 - 2 - 0)/16 = 0 R 8	LSB+3=1
0/2 = 0 R 0	(10 - 2 - 0 - 8)/32 = 0 R 0	end

Figure 4

0	R	1	(The zero quotient indicates no more bits.)
2	√	1	
2	√	2	
2	√	4	
2	√	8	
2	√	17	
2	√	35	

Figure 5

immediately to the left, let's try to divide each previous quotient by 2. Then instead of subtracting from the original decimal number, we drop the remainder after using it as a bit.

Let's compare the results which we have above with that of using the division method. You will see that both give the same results. (We will use R to designate remainder of the division.)


To test this let's convert the decimal 35 to binary. So the MSB will be on top, I will start with 35 at the bottom and divide up:

So the 8 bits 100011 should be equivalent to 35. Let's see:

$$(1 \cdot 32) + (0 \cdot 16) + (0 \cdot 8) + (0 \cdot 4) + (1 \cdot 2) + (1 \cdot 1) = 32 + 2 + 1 = 35!$$

Proving that it works is more complex than using it. Isn't this usually the case with a mathematical proof? But you now know why it works! ✨

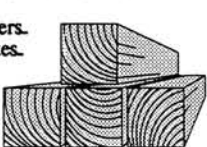
Messages.
Orders.



With this board,
you're really talking!

Which is more useful?

Splinters.
termites.



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

Zenith Data Systems



Clearing Up The Memory Mystery

Pat Swayne
3M Software Engineer

Let's see, there's DOS memory, base memory, upper memory, high memory, expanded memory, extended memory, EMS memory, XMS memory, INT 15 memory, and virtual memory. Did I leave anything out?. Confusing, isn't it? Hopefully, by the time you have finished reading this article, it will not be.

What Is Computer Memory?

You probably know that inside your computer is some stuff called "memory" that is used to store information. When you run a program such as a word processor, the set of computer instructions that make up that program are copied from your hard or floppy disk into your computer's memory. As you enter text, it also is stored in memory. When you save your text, it is copied from memory onto your disk, and when you exit from your word processing program, the copy of it in memory is (as far as we are concerned) forgotten.

There is also some memory in your computer that is used to store the characters or images that you see on your screen while you are using your word processor or whatever. And there is a special kind of memory that the computer can take information from, or read, but it cannot store information, or write, to this memory. This is called "read only memory", or ROM, and it also is permanent memory because it does not forget what is stored in it when you turn your computer off. When you turn your computer back on, the information in a ROM is used to tell your computer what to do to get additional information from your disk, until it is "smart" enough to understand your commands at the keyboard. Normal memory that can be both read from and written to is usually called RAM, which means "random access memory". When you turn off your computer, any information stored in RAM is usually "forgotten" unless there is some kind of

battery system to keep voltage supplied to the RAM memory.

No matter what information is in your computer's memory (for example, word processor text or spreadsheet data) it exists in memory as a series of binary digits, or "bits". When memory is quantitized, that is, when we consider "how much", these binary digits are usually arranged in groups of 8 called "bytes". And when we count these bytes, programmers find it easier to count them using a number system that is a multiple of 2, usually the hexadecimal or base-16 number system. We humans, on the other hand, have for many years been using a base-10 number system, because most of us have 10 fingers. So somewhere along the line someone decided to count memory using binary multiples that were close to decimal multiples. Therefore, a kilobyte of memory is not 1000 bytes, but 1024 bytes. And a megabyte is not 1,000,000 bytes, but 1,048,576 bytes.

Just as you have an address that tells others how to locate you, the memory in your computer has addresses. Each byte has its own address, which is a number starting at zero and working upwards. When writers like me want to visually represent the memory in a computer, we usually use a vertical chart with address zero at either the bottom or top. In this article, I will put address zero at the bottom.

The central processor in your computer is the actual "chip" that does the computing. Depending on what kind of central processor you have, it may not be able to directly "access" (read or write) all of the memory in your computer. Or, it may be able to access certain parts of your memory only while it is running in a particular mode of operation. You may have heard of the terms "real mode", "protected mode", and "virtual mode". I will not attempt to define these terms here

except to say that all computers that use MS-DOS can run in the real mode. If your computer has an 80286, 80386, or 80486 (for the rest of this article, I will just say '286, '386, and '486) central processor in it, it can also run in the protected mode, and if it has a '386 or '486 processor, it can run in the virtual mode. In the real mode, the processor can only directly access the first megabyte of memory addresses (note the exception later). MS-DOS runs in the real mode.

Base Memory and DOS Memory

Base memory and DOS memory are synonyms for the same area of memory in your computer. It is the area used by

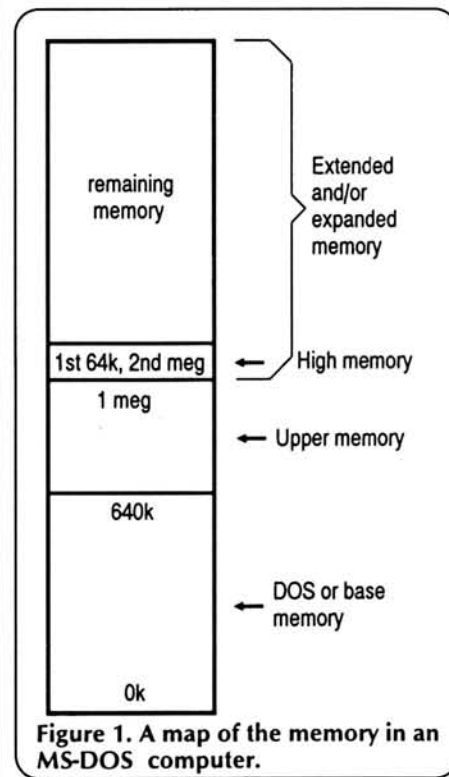


Figure 1. A map of the memory in an MS-DOS computer.

MS-DOS itself (except for part of MS-DOS 5 — more on that later) and by most of the programs that you run from the DOS command line. This memory begins at address zero and continues for the first 640 kilobytes (or 640k) of memory (see figure 1). Not every computer will actually have 640k of memory in this area (an Eazy-PC with no expansion will only have 512k), but if there is any memory in this area, it will normally be base or DOS memory. Base memory can be accessed directly by your computer's central processor no matter what mode it is operating in.

Figure 2 shows how base memory is used. The first section, called "interrupt vectors" contains a table of addresses that point to "service routines", subroutines that accomplish basic computer tasks, such as reading the keyboard or reading and writing a disk. Then there is the "kernel" of the MS-DOS operating system, which has the job of organizing the disk space into files, among other things. If you have any device drivers such as ANSI.SYS installed, they will be in this area. Next is the "Transient Program Area", which is used by programs like your word processor. Some of your TPA may be used by programs that remain in memory after you run them. These programs are called TSR's ("Terminate and Stay Resident"), and are usually either pop-up programs like Side Kick or they do jobs similar to device drivers. At the top of base memory is a place where part of the MS-DOS command processor, COMMAND.COM is stored. This part of COMMAND.COM can be over-written by a program if it needs the space.

Upper Memory

Upper memory is the rest of the first megabyte of memory beyond base memory. Figure 3 shows how it is used. (Each large block is 64k, and each small block is 32k.) As with base memory, the processor can access this memory in any operating mode. This memory was reserved by the designers of the original IBM PC as a place to put ROM and "video memory" (the memory that holds the characters on your screen). Your computer will have at least two ROM areas in it — one for the BIOS ROM and one for the video ROM. The BIOS (Basic Input/Output System) ROM contains most of those "service routines" that we mentioned in the section about base memory. It also contains the instructions that get your computer started when you first turn it on. The video ROM contains service routines for video operations. If your computer only has a CGA video card, then it will not have a separate video ROM, but will instead use video routines in the BIOS ROM. If your computer is an XT-type (they usually have an 8088 processor) and it has a hard disk, then it will have a separate ROM to control the hard disk. Other models have hard disk control rou-

ties in the BIOS ROM.

The BIOS ROM area may not be all ROM, but some of it may be what is called "scratchpad memory" — a small memory used only by the service routines in the ROM.

Some Zenith Data Systems computers place a copy of the video ROM in the area called "video slushware" in Figure 3. That is done because RAM memory is usually much faster than ROM memory — the computer can get data into and out of it faster. Since many programs write directly to video memory and therefore bypass the video ROM, some people consider video slushware a waste of upper memory. However, a device driver is available (from the Zenith Data Systems section of the ZDS-COM1 bulletin board) that can turn off video slushware.

I will explain what a "page frame" is later.

Some computers have RAM memory addressed to the unused spaces in the upper memory area. The UMM.SYS (Upper Memory Manager) device driver that comes with MS-DOS 5 is able to put device drivers and TSR programs into this area, which gives you more Transient Program Area. If you have a '386 or '486 computer, your processor has the ability to "map" memory from the area above 1 megabyte down into this area, so that even if you do not have any memory that is physically addressed to the unused area, you can still have memory there. The EMM386.EXE device driver that comes with MS-DOS 5 can perform this mapping and can serve as an upper memory manager for '386 and '486 computers.

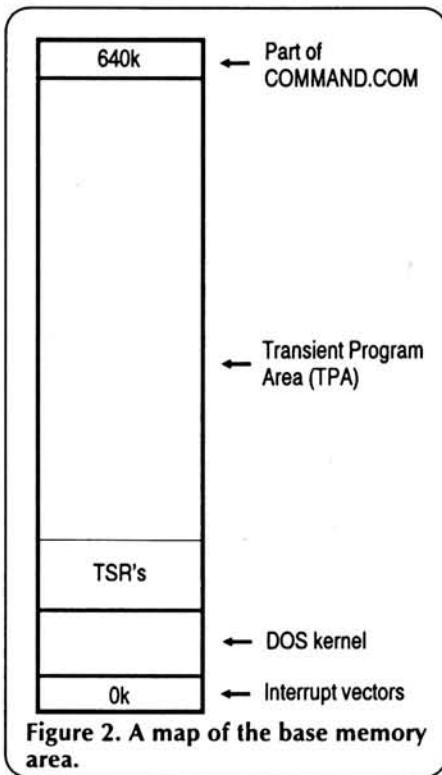


Figure 2. A map of the base memory area.

High Memory

High memory is the first 64k of the memory beyond the first megabyte. If your computer has a '286 or higher processor in it, then it can access this memory directly in the real mode. MS-DOS 5 is able to put most of its kernel here, which gives you more Transient Program Area than you get with other versions of MS-DOS.

Sometimes the term "high memory" is used to describe all memory beyond the first megabyte, but usually it is called extended or expanded memory.

Extended Memory, INT 15 Memory, and XMS Memory

The term "extended memory" can refer to all of the memory in your computer beyond the first megabyte (including the high memory area). It is called extended memory when it is set up so that the processor must be in the protected or virtual mode in order to access it. The original IBM AT computer was the first model to use extended memory. The designers of its BIOS added some service routines that allow MS-DOS programs running in the real mode to have limited use of extended memory. These service routines place the processor in the protected mode, perform some operation on expanded memory, and then return to the real mode. For example, there are routines to copy blocks of memory from the base area to the extended area or from the extended area to the base area. These routines are accessed via what is known as software interrupt 15 (hex), and so extended memory accessed by them is sometimes called INT 15 memory. For an

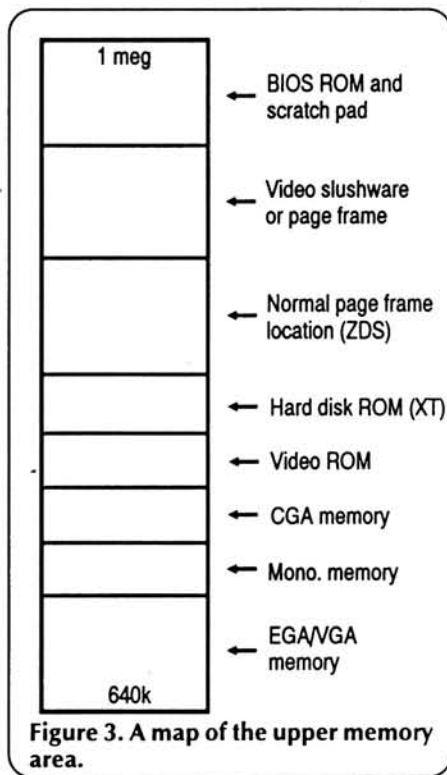


Figure 3. A map of the upper memory area.

explanation of software interrupts, see my series on assembly language in previous issues of REMark.

After a while, it was decided that the INT 15 routines were not sufficient for controlling extended memory. For example, if a program ran in the protected mode, it could not tell what a program running in the real mode and using INT 15 routines had done with extended memory. So if a real mode program was hoping to store something there for later use, it could find it clobbered after the protected mode program (for example, Windows) had been run.

The solution to this problem is the eXtended Memory Specification (XMS). The HIMEM.SYS device driver provides service routines that conform to this specification, which allows programs to access extended memory without conflict. The high memory area used by MS-DOS 5 is also managed by HIMEM.SYS. Normally, HIMEM.SYS "takes over" INT 15 so that it can no longer be used to access extended memory with HIMEM installed. However, the newest version of HIMEM.SYS allows you to specify a portion of extended memory that can be controlled by INT 15. This allows you to use programs which were written before the extended memory specification was developed, but require extended memory.

Expanded Memory and EMS Memory

Even before the IBM AT came along, people decided that 640k of memory was not enough, so some folks at Lotus and other places devised the Expanded Memory Specification (EMS). This specification describes a way of accessing memory above (or outside) the first megabyte by using special hardware to mirror slices of it in the first megabyte. These slices, called "pages", are in an area called the "page frame". A device driver (such as EMM.SYS - "Expanded Memory Manager") is used

to control the pages (for example, what part of the expanded memory they point to). Usually the page frame is placed somewhere in the upper memory area, but in early versions of EMS, it was placed in the base memory, which means that it reduced your transient program area. The page frame is usually 64k in size, but in the "old days" it could be larger and indeed had to be in order to use large amounts of expanded memory. The page frame could really "eat into" your base memory. The current specification, EMS 4, lets you use large amounts of expanded memory from a 64k page frame.

A program like Lotus 1-2-3 can use expanded or EMS memory for its data (spread sheet in this case) while the program itself still uses base memory. The program can thus handle more data, even on an 8088-based machine. When the program writes to a page, the EMS hardware causes a copy of the data to be placed into expanded memory. When the program reads from a page, the hardware copies data from expanded memory to the page before the read is performed.

Normally expanded memory is provided by special hardware (either a separate expanded or EMS memory board or a provision to control existing extended memory as expanded memory), but on a '386 or '486 machine, it can be emulated in software. The EMM386.EXE program provided with MS-DOS 5 can turn any designated amount of extended memory into expanded memory. It also maps a portion of extended memory into the upper memory area for use as a page frame. When you run Windows on a '386 or '486, since it does not require expanded memory, it signals EMM386 to turn most of the expanded memory back into extended memory while Windows is running. The expanded memory is restored when you exit from Windows. The beauty of a '386 or '486 computer is that you can

have enough base memory, upper memory, extended memory, and expanded memory for just about anything you are likely to run.

Virtual Memory

Simply put, virtual memory is making a computer seem to have more memory than it has by using some of the disk space as memory. This is actually a very old concept (as computers go), even for micro computers. The old CP/M version of WordStar was able to edit files larger than the minuscule amount of memory available on 8-bit computers. The user could scroll anywhere through a large file, perform search-and-replace operations throughout it, etc., as if it were all in memory. The WordStar program handled the job of shuffling pieces of the file between disk and memory as needed, and the whole operation was transparent to the user. What's new with regard to virtual memory is that the '386 and '486 processors have the ability to do this shuffling act built into the chip. So far, the only MS-DOS program I know of that takes advantage of this is Windows version 3. The SWAPFILE.EXE program that you run to set up a "swap file" on your hard disk actually reserves a portion of your disk space to be used as virtual memory. Any Windows program is able to use this virtual memory without knowing a thing about it.

Virtual memory has nothing do with the virtual mode I mentioned previously. The virtual mode is a special mode of operation of '386 and '486 processors that permits them to emulate multiple real mode (8088-type) processors. When you run a real mode MS-DOS program under Windows 3 (while it is in the "enhanced 386" mode), you are making use of the virtual mode. That is why you are able to have more than one real mode program running at a time. *



HEPCAT Patch

Pat Swayne
ZUG Software Engineer

There is a potentially serious bug in HEPCAT versions 2.0 and 2.1. To fix the bug in 2.0, create a file called HEP20.SCR that contains these lines

```
E159 2E 80 3E 13 01 03 75 03 E9 2A 2E E9 98 2D 2E 80
E169 3E 13 01 03 74 03 E9 1C 2E CF
E2F87 E9 DD D1
E32BF 98 CE
E32D5 59 01
E3513 59 01
```

W
Q

For version 2.1, create a file called HEP21.SCR that contains these lines

```
E159 2E 80 3E 13 01 03 75 03 E9 31 2E E9 9F 2D 2E 80
E169 3E 13 01 03 74 03 E9 23 2E CF
E2F8E E9 D6 D1
E32C6 91 CE
E32DC 59 01
E351A 59 01
```

W
Q

To patch 2.0, put HEP20.SCR and HEPCAT.COM in the same directory, make it the default, and enter
DEBUG HEPCAT.COM <HEP20.SCR

To patch 2.1, put HEP21.SCR and HEPCAT.COM in the same directory, make it the default, and enter
DEBUG HEPCAT.COM <HEP21.SCR

These patches are for the PC version of HEPCAT only.

Note: After the patch is made, HEPCAT 2.0 or 2.1 can pop up in DOS when the Windows mode is set, but do not attempt to remove it from memory (F4 key) while the windows mode is set. To do so will lock the computer. The current version of HEPCAT is 2.2. If you have 2.0 or 2.1, you can send me your HEPCAT disk c/o ZUG, and I will copy 2.2 onto it and return it. If you have 1.x, send your HEPCAT disk and \$10.00 to Lisa c/o ZUG to upgrade. *

Developing Applications Programs for Windows 3

Harold C. Ogg
357 W. Diversey Avenue
Addison, IL 60101-3508

Microsoft Windows version 3 is a platform whose time has come. The software's popularity has exploded, and even reluctant developers are jumping on the bandwagon to write programs to run under Windows. Zenith Data Systems has its own version of Windows 3 and has packaged the utility and documentation as a bundle deal with many of its ready to run computers. With some setups, Zenith has also included a copy of the Asymetrix ToolBook program and development software, providing a complete Windows programming environment. Zenith assumes that its customers are serious about Windows and certainly has taken steps to promote the program to the nth degree.

The marriage of Heath/Zenith computers and Windows is not new. When Heathkit introduced its first PC compatible computer kit, included was a copy of MS-DOS version 2.0 (along with version 1.25, if you wanted to use it), GW-BASIC, and a coupon for Windows. The first issue of Windows was vaporware at its finest. Finally, Heath/Zenith grew tired of customer inquiries regarding coupon redemption due to Microsoft's now (in)famous delay in releasing the software. In place of Windows, Zenith offered coupon bearers a \$100 credit toward the purchase of merchandise. The actual release windows didn't happen until some months later. As it is with many pieces of software technology, Windows got off to a slow start. It also appeared for awhile that a PC running Windows would be no more than a Macintosh mimic.

Many programmers first considered Windows no more than a plaything. Indeed, even Microsoft itself created its own competition with the OS/2 Presentation Manager. Perseverance ensued, and Windows evolved into Windows/286 and even Windows/386. These were the so-called "versions 2," which were significant improvements over the original issue. For

versions 2, there was a tool for programming, the Windows Development Kit. It was expensive (around \$600) and required an intimate knowledge of C language programming for placement on the screen of grab bars, buttons, frames and all the other niceties of the Windows environment. For the average programmer, Windows was too complicated a product on which to spend hundreds of hours coding a specific application. Windows programs were still the domain of the "big" software houses, and only the more affluent end users could afford utilities written proprietarily for the Windows platform.

Compared to previous versions, Windows 3 was a revelation. There are now sufficient, reasonably priced utility programs and tools for the average or even casual programmer to create applications in a realistic amount of time. If you wish, you can still use the current release of the Windows Software Development Kit (SDK, now retailing at \$500) and write program code from scratch. Power programmers and systems developers will, in fact, most likely prefer this alternative for its control factor and ease of melding with existing source code. Microsoft recognizes and supports this, making available numerous reference books to supplement the Kit and its several linkable libraries.¹ A good way to get a low-cost "sneak preview" of the Kit's elements is with Petzold's *Programming Windows*.² In it, you'll note the techniques of making system calls, incorporating dynamic link libraries (DLLs) in your C language code, and using the Graphics Device Interface (GDI). Those persons already versed in creating applications for the MS-OS/2 operating system will note a remarkable similarity in the processes for manipulating that operating system's Presentation Manager.³

However, in this article we want to look at those facilities and tools where much of the drudgery has been removed.

In other words, we'll consider some user friendly developmental programs that can make your life a bit easier if you must maintain a Windows-based department or workstation group. No games will be reviewed, however; if you or your clients tire of Solitaire or Minesweeper or the limited number of toys in the Windows Entertainment Pack, you'll have to find other amusements yourselves—or write your own article!

Windows Development for the Common Person

For a while, Zenith Data Systems was packaging copies of Asymetrix' ToolBook program⁴ with its bundled computers. This was true of machines that ZDS was promoting in its educational discount program, which also shipped with copies of Windows version 3. In other words, if you purchased certain ones of the ZDS package deals, you automatically became outfitted to engage in Windows developmental programming. Note that if you have a copy of ToolBook from one of ZDS' bundled deals, that software works only on Zenith equipment, in other words, on 80386/80386sx/80486 computers containing the Zenith ROM BIOS. However, this restriction does not affect ToolBook applications created for other brands of PCs; it's just that if you intend to write ToolBook applications on non-Zenith machines, you'll have to obtain a more generic version of the Asymetrix software. This seems like a rather feeble attempt by ZDS to invoke a kind of copy protection, and it thwarts attempts by honest programmers to legitimately transfer the program to others if the purchaser of the bundled deal isn't interested in Windows development. But sermonizing aside, the Zenith version of the program is identical to the off-the-shelf copy, and what follows applies to whatever version you might use.

What is ToolBook? The cover on the

documentation proclaims it a "Software Construction Set for Windows," which is an apt description. More accurately, it is an interface to an interface, in other words, a collection of sophisticated tools for tapping the internals of Windows 3 itself. Windows holds considerable dormant potential in the form of building blocks. They're always available; it's merely a matter of knowing how to summon them. ToolBook performs this job for you.

ToolBook organizes everything into books and pages. It allows you, the programmer, to decide what action to take when a "radio button" is pushed, to determine what the computer will do when you click the mouse pointer on a "hotword" in the middle of a screenful of text, and to establish the procedures for handling error and exception conditions. You'll need to know a little about programming and work flow to take full advantage of ToolBook's (and Windows') full potential. But the rewards will be excellent, and the impressions you make on your clients and users will be lasting ones.

What is the potential here? With ToolBook, you define your own screens and text boxes. You decide whether your Windows 3 application will be only for show (as in a sales presentation) or interactive (as for creating a point-and-shoot educational program). The choice of graphics, positions of the selector buttons and control of peripheral output is yours. The program can be automatic (self running) or manual (wait for input). The look and feel of your demonstration or application is limited only by your imagination.

ToolBook operates at the most basic level from a proprietary language called OpenScript. This language is deeply committed to object oriented programming (OOP), and can either be written from scratch or generated using the Script Recorder utility. For those persons who have used SmartCom III to generate communications dialogs, the process is quite similar. In ToolBook, while you perform the various steps within the Script window, your actions are noted and the appropriate OpenScript code is generated. This code can be edited should you wish to fine tune any aspects of the program, and the Asymetrix software comes with a 614 page reference manual outlining the specific elements of the OpenScript language.

ToolBook Applications

So now that Asymetrix is a member of your computing arena, you need to justify its use and purchase with tangible results. You can create applications from scratch, using ToolBook and its OpenScript language, or you can begin more gradually by using "canned" applications. There is a growing number of the latter on the market, and many are inexpensive. A represen-

tative selection follows.

There is a quotation that "all a writer needs to be successful is something to say." This is true of programmers as well, with variation: a successful programmer needs "something to display and to perform work." Many of the commercial Windows programs are nothing more than standard issues of word processors, spreadsheets, etc., written for access within the special Windows platform. There's nothing wrong with this activity, except that it doesn't exploit the true power of the interface. The opportunity to create slide shows, business presentations, repeating visual attractions, diversions, games, and interactive educational schemes are all possible through Windows utility programming. The beauty is that you don't have to "buy the same real estate twice"—because of its Graphics User Interface (GUI), Windows 3 has preempted construction of the basics for you.

But just what is available, and what ideas have been generated upon which you can build more sophisticated presentation software? Probably the most difficult element in creating impressive applications is the inclusion of artwork. Those persons blessed with artistic talent have a decided advantage in that they can breeze through software-based drawing programs and make the screens sparkle. However, for the rest of us who exhibit ten thumbs when it comes to freestyle artwork, there has to be a resource. Steve Shubitz has wrapped a handsome collection of clip art for ToolBook in a package called ClipBook⁵ (formerly Scrap Book), itself offered in a handy ToolBook format. Steve has arranged the artwork in a slide show presentation so that the developer can page through the catalog either manually or automatically to determine his/her program's needs.

ClipBook is an excellent example of a ToolBook application in and of itself. Steve has included a utility for alternative invocation of the standard Windows 3 calculator, editor, calendar, etc., as well as a chapter for the installation of standard program icons. There is also a feature for creating system scripts (Custom 1 is demonstrated) which causes ClipBook to generate batchlike line-oriented commands for running programs. Those readers who have experienced the Commando shell in Apple A/UX (Apple Macintosh's UNIX) will note the influence of that operating system on ClipBook and will appreciate its ability to create operating system code "on the fly" for you.

ClipBook includes a runtime version of ToolBook, so you don't need a copy of the full-fledged Asymetrix software to run the demos. This runtime version will also support ToolBook applications written by other developers as well, so if you come across a demo disk which makes available only the .TBK and .DLL files, you'll already

have the appropriate support module. I have used the runtime module just mentioned to run the other programs to be described in this article, and it works quite well with all of them. You can obtain a demonstration version of ClipBook as shareware disks UX 50a.0 and UX 50b.0 from Public Brand Software.⁶

It should be mentioned here that icon use (discussed below) and clip art use, while each involves graphics manipulations, they do not constitute the same activity. The artwork just described is for the ToolBook environment, and further, it is not in the realm of standard image formats such as .PCX (PC Paintbrush) or .TIFF (tagged image file format) used in desktop publishing—at least not yet. Icons are small image files used directly by Windows for visual control of programs. However, you'll doubtless be involved using both Windows art formats in a ToolBook project, and it's a good idea to know when to keep them separate.

Public Brand Software also makes available several other collections of ToolBook applications and utilities. There are currently four disks of utility application available (PBS' UX 52.0 through UX 55.0). Some of the .TBK programs on these disks are free, applications written as demonstration exercises; others are copyrighted and economically priced for registered versions. The short programs include MSCOPE, the ToolBook Microscope utility from Asymetrix and TBK-COMM. The latter is a true reference "book" and explains the various serial port calls from the TBK-COMM.DLL (dynamic link library) in the OpenScript language. There are details on all functions including writeComPort, setComPort, flushComTxBuffer, etc. OpenScript allows power programmers to access up to four serial ports from within .TBK applications.

Other programs on the Public Brand disks are more elaborate. VORDAT is an application for pilots which organizes data on VOR flight procedures. A demo version of Hi, Finance! from Brightbridge Solutions (\$30 for the registered version) is also included to show the mathematical capabilities of ToolBook in a personal accounting setting.

Programming Tools

While ToolBook is a major player in creating Windows applications, there is a host of so-called "smaller" utilities that present one- or two-task functions. While some of these are no more than cursor enlargers, key click toggles for the speaker, pyrotechnics for unattended terminal displays and alternate editors, many have unusual and useful functions.

GRABIT is a screen capture utility that can dump specified portions of a screen, or the entire screen, in .BMP (bitmap) format. The Magic Screen Saver blanks the display

after a set period of time (as do many utilities designed to prevent ion burn of CRTs' phosphors), and it requires a password to unblank the display. Even a reboot into Windows can't thwart this protection. Further, the program will announce COM:port activity and advise the user when a background program is attempting to activate the PC. And did you ever wonder how to program the middle button of a three-button mouse? WHISKERS allows you to do so, and you can program the right button to represent the Esc(ape) key. The registered versions of these programs are \$15-\$30. Demo versions of the aforementioned utilities are available on Windows Tools 2 from Public Brand Software as disk UX 16.1.

Windows Tools 3 (disk UX 17.0 from Public Brand) contains utilities for more advanced users. Bar Code uses the Windows Clipboard to create standard UPC or "Zebra" labels. All the usual formats are supported: UPC-A and UPC-E, 2 of 5 and 3 of 9, Codabar and Code 128. These can be printed or transferred to other programs. HotKey can assign as many as 32 programs to single keystrokes for immediate invocation. This utility also makes it possible to assign Windows functions themselves to hotkeys. CPU Usage Meter and MEM keep track of your CPU's active processing time and of RAM (memory)/available disk space respectively. The registered versions of the Tools 3 programs cost between \$8 and \$20.

And there are tools for the ToolBook. Public Brand's ToolBook Programming Tools (disk UX 56.0) contains several bannerware programs (copyrighted, but no registration fee imposed) designed to enhance the facilities in the Asymetrix program. The 3D program allows you to display selector buttons as three-dimensional objects. Custom Colors adds a number of hues to the standard ToolBook palette, including yellows, cyans, and magentas not part of the basic ToolBook capabilities. MicroScope, a utility which later became known as BookLook, lets you examine the contents of books. You can dissect all contents of Asymetrix' books, including fields, objects, properties, and scripts. Finally, IMPORT, a replacement for ToolBook's import verb, gives you the latitude to include comma delimited and columnar database information in your software creation.

Public Brand Software isn't the only purveyor of shareware or of utilities for Windows. On the West coast, The Software Labs⁷ offers its own repertoire of shareware programs and demonstration packages, which includes many utilities and accessories for Windows. The only drawback is that TSL's catalog doesn't publish costs for fully registered versions of the annotated programs as Public Brand does.

Nonetheless, The Software Labs contains a number of programs not found in Public Brand's catalog (and vice versa), with sufficient information to evaluate Windows accessories before you invest in the full-fledged versions.

The Software Lab's disk #1487, the Windows Programmer's Toolbox, contains some rather high-powered software which will appeal to the serious Windows developer. In addition to another substitute for the Windows Notepad, WINEDIT, a demo version of TOOLS:W⁸ is featured. TOOLS:W assists you to write programs using the Windows Software Development Kit, and, of course, you'll need a Microsoft C compiler to process the code. The demonstration module utilizes the Btrieve record manager, of which a TSR (terminate and stay resident) version is included, and features a database application of relatively high complexity. The writeup claims that as much as ninety-five per cent of normal coding time can be saved using TOOLS:W, leaving you with more time to fine tune and debug your royalty-free applications.

TOOLS:W, however, is for a very focused clientele. If you're going to develop the kind of Windows based database applications TOOLS:W supports, you'll need to purchase the full-fledged version of Btrieve from Novell. And you'll need to be conversant with Windows' Dynamic Data Exchange (DDE) processes. But you can also develop for OS/2's Presentation Manager with TOOLS:W, as well as include dialog box input validation, a multi-user database interface, a graphical display of database records, and a report generator in your creations. With the registered version, the publisher includes a copy of a finished application to demonstrate the power of TOOLS:W. There is also available an accessory module called Version Tool which allows data bases created under previous applications to work with the most recent versions of compiled programs. Source code for the callable libraries is available as an extra cost option.

If you'd rather start your Windows programming on a slightly more elementary level, The Software Labs' disk #1487 also includes a demo of WinBegin, a program that generates simple C code for Windows applications. WinBegin is an excellent experimenter's tool, and it also serves more advanced program developers as a utility for creating skeletal Windows programs. WinBegin's Make Program menu selection prompts for a program/directory name, a brief description of the intended program, and your (i.e., the author's) name. After typing the particulars, you get a Windows 3 application complete with menu bar commands and two dialog boxes. You can then vary and build on the program by adding another menu item and dialog box. The registered version costs \$35, and, ob-

viously, you must own a C compiler to generate the final code version.

The Software Labs has thirteen disks entitled MS Windows Accessories in its catalog, each containing between three and eight utility programs. Most of these are of interest primarily only to end users, and the programs include personal finance utilities, electronic card files, disk managers, and label makers. However, the contents of Accessories #2 (TSL disk #4040) is of particular interest to programmers: Winbench version 1.1, developed for PC Magazine Laboratories, is a benchmark program that displays performance information data about Windows display drivers and graphics display boards. Unlike generic test programs, the tests in Winbench target hardware specifically accessed by Windows and programs running under that platform. This is a particularly useful program to diagnose and prevent potential problems in the course of the development of a Windows application. Also included on disk #4040 is WinPcx, a utility that allows you to view graphics images in standard PC Paintbrush (.PCX) format while running Windows. Windows literature is a bit thin on the use of graphics image formats other than those specifically created for Windows and those in .ICO (Windows icon) and .BMP (bitmap) format. Source code is included for developers who would experiment with this medium.

The Software Labs' disk #1741 contains a demo version of UNICOM 2.0,⁹ a communications program capable of executing data transfers in the background while other Windows programs are running. This is similar to Lattice Software's SideTalk program, an older, non-Windows application. UNICOM was written with programmers in mind, and has features attractive to the developer who has little time to spare attending to uploads, downloads, and incoming bulletin board inquiries. UNICOM supports all the popular transfer protocols, including X-, Y-, and ZMODEM, Kermit, Checksum and ASCII, and it can emulate color ANSI-BBS, VT52 and TTY terminals. Programmers will appreciate UNICOM's 35-command script language with trace mode for debugging, as well as its capability to execute as many as eight script files through the Script Scheduler facility. It would be nice if future releases would support more terminal emulations, but UNICOM is still a bargain for its variety of features. It's a good replacement for the standard communications utility found in the basic release of Windows.

Finally, there is also a batch language interpreter for Windows. The Software Labs offers a demo of WinBatch version 2.0¹⁰ as disk #4092. WinBatch is an extension of the DOS batch language (obviously) that includes windows-specific functions. It appears to be a hybrid collection of functions

similar to those found in other utility programs such as the Window Boss and Hyperkinetix' Builder (see my October 1990 REMark article, page 27, for details on batch file utilities). As such, the WinBatch language elements resemble DLL calls. The verbs and modifiers of the language are intuitive, and functions such as FileOpen, StrReplace, AskYesNo, and Beep require little explanation. With WinBatch you can send keystrokes to applications, perform arithmetic and string operations, make branching and jumping decisions, and display information to users in several formats. WinBatch is a good alternative to writing interfaces and operations code in C language, and it lets you provide your Windows clients a more user friendly environment in a hurry.

Icons and Wallpaper

If there's any one thing available in abundance for Windows 3, it's icons. Many Windows users proclaim that icons are what give Windows its unique "personality." To be sure, end users who are not technically inclined rely psychologically on finding identifiable icons on the screen to get their applications started. And, icons allow Windows developmental programmers a bit of latitude for creativity. After all, on a conventional (bare DOS) setup, you can't easily substitute Bart Simpson's face or Bill the Cat for the C> prompt!

And it's easy to incorporate third-party .ICO images into your Windows environment. For a non-Windows program, you need only invoke the Program Manager and attach as a "New Program Object." You'll then be given an opportunity to "Change Icon," at which time you can select an appropriate icon by name and enter a typed description for a caption that will appear beneath the .ICO graphic. The rest is a matter of saving your changes; the icon and its related program are available for invocation immediately. You can return later and attach a different icon if you wish. This procedure is very clearly detailed in the Microsoft Windows 3.0 User's Guide furnished with the Zenith release of the software. In regards to appearances, icons are the "little touch" that can make the biggest difference to the clients for your Windows applications.

"Cheap and plentiful" best describes icon libraries. Public Brand Software offers three collections as disks UX 42.0, UX 43.0 and UX 44.0. Altogether, there are more than 2,350 icons spanning these disks; represented are nearly all the major applications programs of vendors from Ashton-Tate to Zortech. Many of the major utility programs and games are depicted in .ICO files, too. The Software Labs also offers three collections as disks #4085, #4086 and #4087. TSL doesn't squeeze as many icons on its disks as Public Brand, but TSL's

individual disks are cheaper. I'll leave the economics of the situation to the reader since both vendors offer quality collections.

Despite the availability of a couple of thousand icons in the public domain, it is recognized that there are developers and programmers who will want to create their own, custom .ICO files. A variety of tools exist for this purpose. Public Brand Software offers a demo disk (UX 41.1) entitled Windows Icon Tools, containing several utilities for icon manipulation. IconDraw version 1.1 works like a standard draw program to create 16-color 32 x 32 pixel graphics in standard .ICO format. The usual rectangle, line, ellipse, and pencil tools are available in the program. Icons Management is also provided on the disk, and it displays .ICO files in screens of fifty icons. Icon Master allows capture of icon-sized portions of the screen, making the creation of new icons considerably faster than starting from scratch. And IconDLL allows C programmers to change .ICO files into .DLL files; the latter require less disk space. Registrations are required only for IconDraw (\$15) and Icon Master (\$20).

The Software Labs offers a similar disk (#4043) which also includes the IconDraw utility (above) and Icon Library to manage your collections. Icon Library version 1.0 also includes 330 of its own icons. PBIcon is also included on TSL's #4043 disk. This program lets you create icons using Windows Paintbrush, an advantageous facility since you have all the drawing tools of Paintbrush at your disposal for the special task of editing icon graphics. PBIcons works by opening/saving files in a format accessible by the Program Manager, and by using the Clipboard to transfer files back and forth between itself and Paintbrush—the latter edits graphics in a bitmap (.BMP) format. A \$10 registration fee is required.

Another one of The Software Lab's thirteen accessories disks (MS Windows Accessories #8, TSL disk #4078) is devoted to icon manipulations. Icon Tamer and ListIco are handy management utilities for viewing, copying and deleting libraries of icons. But the unique tools on this disk are Gif2Ico and WinGIF, two programs that allow conversions from CompuServe's .GIF graphics format into Windows icons. WinGIF additionally allows some dithering of images as well as clipping/trimming of objects and alteration of the icons' color palettes. Graphics image conversion programs (e.g., Hijaak) have been on the market for some time, but most have ignored the .ICO format. You need this program if you're really into icon creation. WinGIF requires a \$15 registration fee.

The matter of wallpaper is similar to that of icons, inasmuch as it is stored on disk as a bit pattern. If you tire of your wallpaper's plain vanilla pixel pattern, you

can display various designs or even pictures as your desktop background. As with any graphics, you should be frugal with the number of wallpaper variants stored on disk since each one requires a moderate amount of storage space. But changes of wallpaper can add interest to your Windows environment, and unique patterns can delight, and perhaps inspire, clients for your written applications.

The Software Labs has three wallpaper utility demos in its catalog. Clipboard Tiler (i.e., TILER, disk #4045) lets you use tiles (obviously) as the bases for wallpaper patterns. Using the Windows Paintbrush program you create a tile, copy it to the Clipboard, and view it as wallpaper without having to convert. Sixteen tiles are included, and the disk also includes 1.3 megabytes of finished wallpaper. Registration is \$20. WallBlaster (disk #4094) uses .ZIP archive processes to conserve hard disk space; with it, desktop wallpaper is selected from elements of an image library. Personal WallBlaster costs \$20 to register; Professional WallBlaster, which also includes a library of callable C language functions, is \$40. Zipaper (disk #4095) selects wallpaper from a bitmap .BMP image and allows you to store it as a compressed .ZIP file. Wallpaper is displayable in resolutions up to 800 x 600 pixels. A registered version of Zipaper version 1.2 is \$10.

Book-Based Programming Tools

There are several excellent paperback books to support Windows 3 programming; most are of a supplemental nature. Two are reviewed here because they each contain bonuses—disks with the utilities discussed in the text.

*Windows 3 Power Tools*¹¹ is split about 50/50 on Windows use and Windows development. The programming section really begins with the chapters on memory management and performance fine tuning, together with customizing the .INI files. The four chapters concluding the book discuss in detail four programs offered as shareware: Oriel, Command Post, Aporia, and IconDraw. The latter is the same as the draw program in the Icons section immediately above; the remaining three will be outlined briefly.

Oriel is similar to ToolBook, except somewhat scaled down and sporting a command set of thirty-three operators. Its power is the fact that it allows the programmer direct access to the Graphics Device Interface (GDI), and, as such, offers some features not directly available with the Windows Software Development Kit. The draw commands are executed from English words contained in ASCII files, which are reminiscent of the BASIC draw functions and verbs used to direct a pen plotter. Your "canvas," of course, is the CRT, and you can include menus and message boxes as targets for

the library of shapes, colors and patterns available. Distribution of your applications is royalty free, but you'll need a runtime module¹² that complements shareware Oriel.

Command Post is an alternate shell to the Program Manager and File Manager. The chief attribute of Command Post is its facility to allow text files to define custom menus which then appear on the Executive Window's menu bar. On the surface, Command Post resembles the shell interface introduced with DOS version 4.0. As an extension, the Command Post Menu Language (CPML) is a powerful file manager, its one hundred+ operators and functions permitting creation, deletion, renaming, and copying of files while displaying them in cascaded or tiled arrangements. The CPML acts as a very powerful batch language to give you a wide latitude of controls over appearances and manipulations. A coupon in *Power Tools* allows you a \$9.95 savings over the regular \$49.95 registration fee.

There is also included a shareware copy of Aporia. This program also functions in a manner similar to the Program Manager and File Manager, and is very Macintoshlike in appearance. Its platform is even called a "desk," and, while it doesn't offer a programming or batch language as do the other utilities in *Windows 3 Power Tools*, it offers a true iconic interface for a \$50 registration fee.

For those persons who would rather just experiment with Windows 3 development, there is an alternative to the Asymetrix ToolBook. Michael Hyman has written *Windows 3 for BASIC Programmers*,¹³ a title which is somewhat misleading since his subject is actually the presentation of a language toolkit for building and executing Windows programs. He calls his project the Realizer Limited, a program which allows some manipulation of Dynamic Link Libraries, along with facilities for object oriented programming, multitasking, menu

creation, and event driven programs. The book is formatted as a tutorial and includes a disk, two attributes which followers of Zenith Data Systems' continuing education courses will find highly appealing. If what you want to do is learn Windows programming fundamentals and enrich your PC programming knowledge, a purchase of this paperback is money well spent.

Programs and References Mentioned

¹*Microsoft Windows Guide to Programming*, \$29.95 *Microsoft Windows Programmer's Reference for Version 3*, \$39.95 *Microsoft Windows Programming Tools*, \$24.95. Redmond, WA: Microsoft Corporation, 1990. These three paperbacks constitute the Microsoft Windows Programmer's Reference Library and are the definitive answers to the Windows Software Developer's Kit (SDK).

²Petzold, Charles. *Programming Windows: The Microsoft Guide to Writing Windows Applications for Windows 3*. Redmond, WA: Microsoft Press, 1990. Paper, 944 pages, \$29.95.

³MS-OS/2, version 1.21, part no. OS-31-6 (3 1/2" disks) or part no. OS-51-6 (5 1/4" disks) from Zenith Data Systems, P.O. Box 1000, St. Joseph, MI 49085. \$339.00 plus your state's sales tax and \$4.50 S/H.

⁴ToolBook version 1.5, available from Asymetrix Corporation, 110 - 110th Avenue N.E., Suite 717, Bellevue, WA 98004. \$395.00.

⁵ClipBook, version 1.9B, available from Published Perfection!, 7486 La Jolla Blvd., Suite 552, La Jolla, CA 92037. \$39.95 plus \$5.00 S/H.

⁶Public Brand Software, 3750 Kentucky Avenue, Indianapolis, IN 46241. Disks are \$5.00 each (either 3-1/2" or 5-1/4" format), plus \$5.00 S/H per order.

⁷The Software Labs, 3767 Overland Avenue #112-115, Los Angeles, CA 90034. Disks are \$3.69 each in 5 1/4" format and \$4.69 in 3 1/2" format, with reduced prices for multiple disk purchases, plus \$4.00 S/H per order.

⁸TOOLS:W, available from The Software Factory, P.O. Box 2639, Gainesville, FL 32602. Basic Tools, \$495.00 (\$995.00 for source code); DDE Tools (including source), \$295.00; Version Tool (including source), \$295.00.

⁹UNICOM 2.0, available from Data Graphics, P.O. Box 46354, Seattle, WA 98146. \$45.00.

¹⁰WinBatch, available from Wilson WindowWare, 2701 California Avenue SW, Suite 212, Seattle, WA 98116. \$69.95.

¹¹LeBlond, Geoffrey, LeBlond, William B., and LeBlond, Jennifer L. (a.k.a. The LeBlond Group). *Windows 3 Power Tools*. New York: Bantam Books, 1991. Paper, 664 pages, \$49.95 (includes a 3 1/2" 1.44 MB disk).

¹²Oriel for Windows Runtime, available from The LeBlond Group, P.O. Box 247, Soquel, CA 95073-0247. \$250.00.

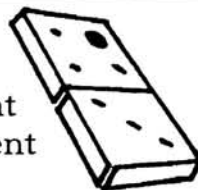
¹³Hyman, Michael. *Windows 3.0 for BASIC Programmers*. Reading, MA: Addison Wesley, 1991. Paper, 316 pages, \$29.95 (includes a 5-1/4" disk).

About the Author

Harold C. Ogg is Chief Information Officer and Director of Academic Computer Facilities for Roosevelt University in Chicago. His specialty is Library and Information Technology, and he has authored numerous articles on the use of small computers for information delivery.



Z-89 & Z-100 Software
(New) \$50 each package
Parts for all Zenith Data Systems Equipment
Repairs for all Zenith Data Systems Equipment
Batteries for '171, '181, '184, etc.
Demo Equipment



Since 1980

TURBOSPORT
\$1295 Demos
While supplies last

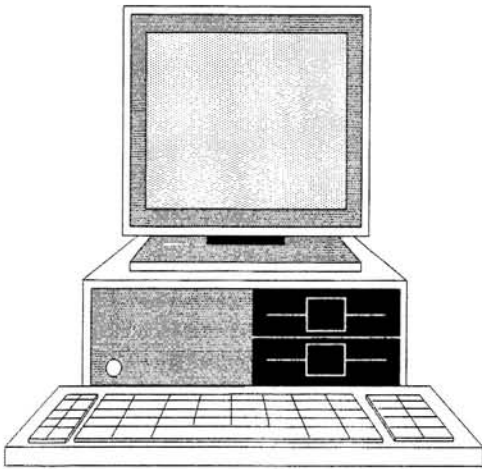
**DOMINO
COMPUTERS**
2801 Touhy Ave. - Unit "A"
Elk Grove Village, IL 60007-5300
708-299-4422 EXT 653



MOVING?
Don't miss a single issue of REMark!

Please let us know 3-4 weeks
before you move. Call
(616) 982-3463 or write:

Zenith Data Systems
P.O. Box 217
Benton Harbor, MI 48023-0217



Budget Desktop Publishing Is Here!

A Look at Key Publisher and the Hewlett Packard DeskJet 500

*Patrick Swayne
ZUG Software Engineer*

Believe it or not, this entire article was typeset using a printer that cost \$416 including shipping and COD charge, and it was prepared using software that cost \$14.40 including shipping charge. The printer is a Hewlett Packard DeskJet 500, and the software is Key Publisher.

Key Publisher

Key Publisher is a clone of another publishing program called Publish It!. It is marketed by Softkey Software Products, Inc., a company that buys the licenses to products made by other companies and sells them under a different name at a discount price. The Selective Software catalog lists a number of Softkey products (including Key Publisher) at \$39.95 each, but recently DAK Industries offered Key Publisher at only \$9.90 plus \$4.50 shipping. Their motive in doing this, according to their catalog, was to get the names of computer users on their mailing list. Well, I don't care if I get more junk mail (I can always make the recyclers busier), so I ordered Key Publisher from DAK, and what a pleasant surprise it turned out to be! I don't know if the DAK offer will still be good by the time this article is printed (call them at the number listed at the end to see), but even at \$39.95, Key Publisher is a good investment for someone who wants to experiment with desktop publishing without much expenditure.

Key Publisher runs under GEM (Graphic Environment Manager), a graphic interface from Digital Research. It is so much like the GEM

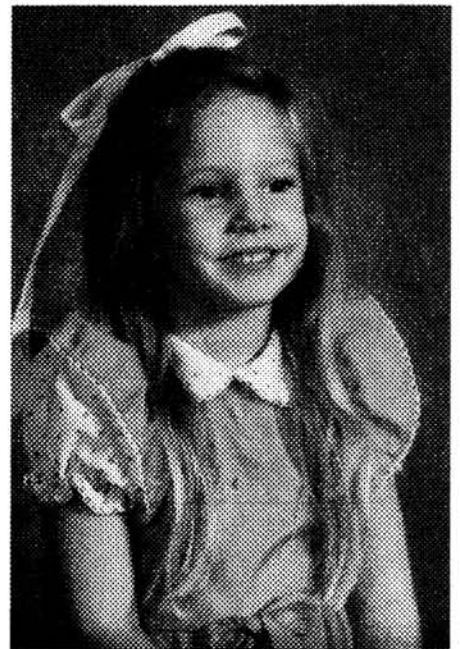
version of Ventura Publisher that a person familiar with Ventura could probably use Key Publisher without reading the manual. Of course, it does not have all the features of Ventura, but it has a few features that Ventura could use, including more page views and more graphic tools, such as the ability to edit imported bit map graphics.

Key Publisher Features

Like Ventura, Key Publisher has four operating modes that control different aspects of compiling a publication — the frame, paragraph, text, and graphic modes. In the frame mode, you draw rectangles on your page that will hold any text or graphics that you intend to place there. You cannot put anything on a page without first drawing a frame for it. The frame borders can be invisible, or they can be lines of various thicknesses. Key publisher is even more strict in its use of frames than Ventura. In Ventura the page itself can be considered a frame, so you can put text or non-imported graphics on a page that has no frames. In key publisher, however, nothing can be placed on a page without a frame to put it in. Even the lines between the columns in this article are in their own frames. In Ventura, you can divide any frame into columns, but in Key Publisher, you can only divide the whole page into columns, and each column must have a frame drawn on it before it can contain text. To make it easier to draw these frames, you can turn on a "snap" features that causes

the frame lines to tend to "stick" to the correct places as you draw them. All this frame drawing business may sound tedious, but if you are going to use the same setup more than once, you can make a "template" containing the frames, graphic lines, and even headers and footers (such as the page numbers and date in this article) that you will require, and use it for each publication that you prepare. So unless you are going to use a different layout each time, you will only have to draw most of your frames once.

After you add your frames, you can



Key Publisher can import scanned photographs as well as other graphic images.

fill them with text and pictures from external sources. Key Publisher can import WordStar, Microsoft Word, and Word Perfect format files, as well as plain ASCII files and a few other word processor formats that I am not familiar with. It can import line (vector) graphics in the .GEM and Lotus .PIC formats, and bit mapped graphics in the .IMG, .PCX, .TIF formats, and two other formats that I am not familiar with. It can also import encapsulated PostScript (.EPS) files, but only for the purpose of passing the information on to a PostScript printer. It cannot display an .EPS picture or print it on a non-PostScript printer.

(You have probably noticed by now that the font on this page is smaller. One drawback of Key Publisher is that it does not have a 9 point font size, which is what we use in REMark. So I did the first page in 10 points, and the rest in 8 points. More on fonts later.)

The paragraph mode is probably the best feature of Key Publisher (and Ventura Publisher, which it was obviously modeled from). It is used to assign "paragraph styles" to blocks of text. In this article, the title is assigned one style, the topic headings another style, and the "body text" is another style. If I want to change the type face of all of the headings, I just have to select the paragraph mode, click on one heading, and then change the typeface. The other day I watched an associate trying to change the point size of just the body text in an article in PageMaker (without affecting the headings or anything else), and I noticed that it took him much more effort to do it than it would have taken me in either Key Publisher or Ventura.

The text mode is used for entering and editing text, and for making typeface or point size changes to individual characters or words. Like most desktop publishing programs, Key Publisher has minimal word processing capabilities built in, so that you can prepare short documents without using an external word processor. However, I found that it responded sluggishly to fast typing, which is probably why the Key Publisher manual recommends that you use an external word processor and then import your text whenever you can. Unlike some desktop publishing programs, Key Publisher includes search and search-and-replace commands.

The graphic mode is used when you want to draw lines, boxes, circles, and ellipses. In addition to the straight line drawing capability found in most desktop publishing programs, Key Publisher also has a freehand line drawing mode. You can fill your boxes, circles, and ellipses with any of several patterns, and you can automatically add arrow heads to your lines. The only problem with Key Publisher in this mode is the limited selection of line widths. In Ventura, you can specify just about any width, using several measurement systems (inches, metric, points, etc.). In Key Publisher, you have to select the width from a menu.

Key Publisher supports 9 and 24 pin Epson-compatible dot matrix printers, Hewlett Packard and PostScript-compatible laser

printers, and, of course, the DeskJet series inkjet printers. If you have a PostScript printer, it will use the type faces available in your printer, and it can also use Adobe Type 1 soft fonts. For all other printers, Key Publisher usually generates the characters itself and operates the printer in its graphic mode. However, it can use soft fonts with most Hewlett Packard LaserJet printers. With the DeskJet, you are limited to using the built-in fonts in Key Publisher. These include two book type faces called Dutch (like Times Roman) and Swiss (like Helvetica), which are provided in several sizes from 6 to 36 points. Some display typefaces are also provided, including

⇒ **Rockface** in 10 and 18 points (18 shown)

⇒ **Ravinia** in 9 and 18 points (18 shown)

⇒ **DRUDY LANE CAPS** in 10 and 18 points (10 shown)

⇒ **Madison** in 12 and 18 points (12 shown)

A special typeface of bullets, such as those above, is also provided in 7, 10, 14, and 18 points.

For some reason (probably to save memory) neither true bold face nor italic versions of the fonts are provided. Instead, to make bold face, the normal typeface is sort of double struck, so that it looks like WordStar-type bold print on a Diablo printer. To make italic type, the normal type face is obliqued.

it it it

Bold face is formed by double striking the normal typeface, and italic is formed by obliquing it.

One drawback of Key Publisher is that it uses only what is normally called "DOS memory", and it requires quite a bit of it. It does not use any extended or expanded memory. If you have a '286 or '386 computer with more than 1 megabyte of memory and MS-DOS 5, however, you will have plenty of memory to work with. The first thing that suffers when you run short of memory is the font generating part of Key Publisher. When you print your document, fonts that you had set at 24 point size, for example, may be printed in 10 point size if you are short of memory.

Other Inexpensive Publishing Software

There are actually many high quality inexpensive desktop publishing and related programs to choose from, with many new ones becoming available quite recently. The "parent" of Key Publisher, Publish It!, is now available in version 2 (Key Publisher is version 1.2), which is said to have many improvements, including scalable fonts. And speaking of fonts, Adobe Type Manager is a program

that has impressed me so much that I intend to do an article about it in the future. Adobe Type Manager is a sort of device driver for Windows that allows just about any Windows application to use Adobe Type 1 scalable typefaces for both the screen and printer. Even the Windows Write program (the editor that comes with Windows) can do fancy things with Type Manager. And fonts for Type Manager are beginning to appear on bulletin boards — yours for the downloading. All that is needed to go with Type Manager is an inexpensive publishing program for Windows, and Microsoft has just released one called Microsoft Publisher that sells for under \$160. There are also some word processing programs for Windows that can do desktop publishing. Adobe Type Manager itself can be purchased for under \$60 from many mail order companies. Desktop publishing "for the rest of us" is becoming a reality.

The Hewlett Packard DeskJet 500 Printer

The Hewlett Packard DeskJet 500 printer is a plain-paper inkjet printer with a resolution of 300 dots per inch. It retails for around \$800, but I was able to get one from a mail order company called Allon Computer for \$395 plus \$21 COD and shipping. The man on the phone said that the unit was "like new" so I figured that it must not be "brand new" in some way. However, he said that Allon Computer would warranty it for 90 days, so I took the chance and ordered it. When it arrived, it was in the original carton, but I could tell that it had been opened and re-packed. However, inside were all of the manuals that go with the printer, still in shrink wrap, and the manufacturer's warranty card was included in that pack. So I knew I was double-covered in case something went wrong.

Actually, there was a minor problem to contend with. Inside the box was a small rubber grommet under the printer. I figured that it must have come from the printer, so I opened its case and found that it was one of the grommets used to shock-mount the print mechanism. It was a snap to replace, and literally a snap to re-assemble the printer, because the case is held together with plastic tabs that snap in place.

The DeskJet 500 is about 17 inches (44 cm) wide, 8 inches (20 cm) high, and 15 inches (38 cm) deep. It comes equipped with both a serial and a parallel interface, so it should work with just about any computer you have. The serial interface can use either XON/XOFF or DTR handshaking. I have only tried the parallel interface myself, and the owner's manual recommends it because data can be sent to the printer faster. The power supply for the DeskJet is in a separate box, which can be placed on the floor beside your desk or printer stand. The power supply was probably put in a separate box to reduce the weight of the printer and to avoid heat problems.

The "print head" in the DeskJet 500 is a

replaceable ink cartridge that snaps into place in a carrier on the print mechanism. The paper handler consists of a lower input tray and an upper output tray. The input tray can hold about 50 sheets of average weight paper. Hewlett Packard recommends that you use copy machine paper. (Even though the DeskJet 500 is an inkjet printer, they specifically recommend that you do not use inkjet paper.) There is also a special slot in the upper tray that accepts a standard business sized envelope. When two of the buttons on the printer panel (marked with a figure of an envelope) are pressed together, the envelope is pulled into position for printing.

I tried my DeskJet 500 out with Key Publisher right away, and discovered that it worked fine except for some white streaks through some of the characters. According to the manual, this is caused by dirty contacts on the inkjet cartridge. Since the actual ink jets are on the replaceable cartridge, it must make electrical contact with the rest of the printer. Actually, I figured out that dirty contacts must be the problem before I read the manual, and when I cleaned them (with a paper towel moistened with alcohol) the printer worked fine. The manual recommends that you use plain water (on a cotton swab) to clean them, but the alcohol did not seem to harm anything.

The DeskJet 500 comes with a number of built in fonts, including Courier and Gothic (typewriter-style fonts) in a number of sizes, and Roman in 6 and 12 points. It also has two cartridge slots that can accept either font cartridges or memory cartridges. If you add memory, you can download soft fonts into the printer. The manual indicates that the printer comes with one font cartridge, but mine did not include one. I have not even bothered to call Allon Computer about this, because only one of the programs I have used with the printer, WordStar, uses the printer's own fonts. I cannot see any purpose in buying either memory or font cartridges for the printer, since most programs provide their own fonts and run the printer in the graphic mode. And unlike a laser printer, the DeskJet does not require memory to do graphics. (The one exception to not getting a cartridge would be the Epson compatibility cartridge, which makes the printer Epson compatible, for those times when you must use software that only supports Epsoms.) The only drawback is that printing in the graphic mode is slower, with some programs taking quite a while to compose the information before sending it to the printer. Key Publisher, however, is almost as fast at filling a page with 12-point Roman text as WordStar (which uses the printer's own fonts) is. And Windows Write using Adobe Type Manager is almost as fast. These programs tend to bog down if you include pictures in your text, but that happens with any printer.

The DeskJet 500 supports what Hewlett Packard calls Level III of their Printer Command Language (PCL). This is the same

"language" (it is not really a language, just a set of control codes) that Hewlett Packard LaserJet printers use, although some models use higher levels. Because of this, the DeskJet 500 can work with some applications that list the LaserJet as one of their supported printers, but not the DeskJet. I tested two shareware programs, CompuShow and Graphic Work Shop, which support LaserJets, but do not even mention the DeskJet in their documentation. Both programs worked perfectly on the DeskJet 500. Probably any program that supports a LaserJet Plus and runs in the graphic mode will support a DeskJet 500. A program that uses LaserJet font cartridges or soft fonts will not work with the DeskJet 500.

I don't think I have to say anything about the quality of print produced by the DeskJet 500 or Key Publisher. What you are seeing speaks for itself. If you are thinking of going into desktop publishing and have been considering a laser printer, you should also consider the DeskJet 500. It enjoys wide support by many programs (including many that specify only the LaserJet), and the initial cost is lower. On the negative side, the DeskJet 500 is slower than a laser (mine does about 1 page per minute with Key Publisher), and the operating cost is higher. The ink cartridges retail for about \$20, and the lowest street price I have found is about \$17. They last for about 300 pages, which makes the cost about 5.7 to 6.7 cents per page, less the paper. However, refill kits have recently been introduced by third party vendors. One of these is the CompuJet Ink Refill System, which costs \$12.95 for a two-refill pack. If these work acceptably, then the cost would lower to about 2.2 cents per page if the cartridge could be refilled indefinitely. This is close to the cost of operating a LaserJet IIP, as mentioned in William Will's article in the October 1991 issue of REMark (page 35). Some people have tried refilling cartridges using ordinary fountain pen ink, but I would not recommend that. It is reported that the ink jets wear out after a few refillings. It remains to be seen if that will happen with a commercial refill kit.

The slowness is not objectionable if you are going to print one master of a document and then take it to a copy machine or printer, or if you are only making a few copies.

Other Inkjet Printers

Before I purchased my DeskJet 500, I considered other inkjet printers, specifically the Bubble Jet series made by Canon. The street price of some of these is less than what I paid for my DeskJet (I have seen the BJ10e at \$285), and it works at a maximum resolution of 360 dots per inch. However, the Bubble Jet printers are actually inkjet versions of 24-pin dot matrix printers, and their "native" resolution is 180 dots per inch. They are probably driven in a "double density" mode to make 360 dots per inch, which means that the print head must make two passes over the paper for each band of printed

material. This could affect speed as well as quality. Also, they are not as widely supported as the DeskJet, although they can be operated in a mode that emulates an IBM Proprinter, which is fairly widely supported. If anyone reading this has a Bubble Jet printer, I would appreciate it if you could send me some sample output for comparison. Meanwhile, I think I'll keep my DeskJet.

Products Mentioned

The companies mentioned with each product are selected suppliers.

Key Publisher (DAK version) \$14.40
Order number 5919 for 5.25" disks, or 5920 for 3.5" disks.

(this price includes shipping)
DAK Industries
8200 Remmet Avenue
P. O. Box 7120
Canoga Park, CA 91304-9955
800-325-0800

Key Publisher \$39.95
Order number X90235 for 5.25" disks, or X90233 for 3.5" disks.

Selective Software
2515 East 43rd Street
P. O. Box 182217
Chattanooga, TN 37422-7217
800-423-3556

Publish It! (version 2) \$130

Telemart
8804 N. 23rd Avenue
Phoenix, AZ 85021
800-521-1973

Microsoft Publisher \$159.99
Egghead Software
(various retail locations)
800-EGGHEAD

Adobe Type Manager \$58
Order number 08259
Multiple Zones International
18005 NE 68th Street, Suite A110
Redmond, WA 98052-9904
800-528-2088

Hewlett Packard DeskJet 500 \$395
Allon Computer
211 Marshall Street, Suite 200
Redwood City, CA 94063
415-366-5554

CompuJet Ink Refill System 2 PAK \$12.95
Order number 3572
Lyben Computer Systems
1150 Maplelawn
P. O. Box 1237
Troy, MI 48099
313-649-4500



LANtastic

Jeff Babcock
121 Green Street
Plymouth, IN

Artisoft's LANtastic is a DOS-based Network Operating System that allows peer-to-peer sharing of hard disks, printers, and CD-ROM drives. It has a low memory requirement and provides much of the capabilities of higher priced networks. Version 4.0 of the LANtastic operating system was released in June of 1991. It added greater support for Windows, faster overall performance, and some new features that users have been wishing for. This article describes some of the capabilities of LANtastic and presents a case study of my first installation.

Unloadable from Memory

LANtastic v4.0 is unloadable from RAM memory, unlike the earlier versions that forced you to reboot the PC to halt the server. The server can send out a message to any users logged on that a shut down has occurred.

Remote Program Execution

With this release they added a NET RUN command that allows users to execute remote batch commands on a server from their machine. So you could use idle servers to perform some time consuming jobs such as program compiles, database searches, etc. while allowing you to use your local machine for other work.

UPS Support

UPS (uninterruptable power supply) support has been added with automatic and remote shutdown capability as well as additional advanced utilities. If the UPS sends a signal to a server saying that a shutdown is imminent, LANtastic will warn all the network users to logoff. Then after a short pause, LANtastic closes any open network files so that no one will lose any data. The UPS must support a connection to a monitoring board in your PC. Some manufacturers who support this are American Power Conversion (800-541-8896), TrippLite (312-329-1777), and Sola Electric (800-879-7652).

Speed

Artisoft has reworked the functioning of their NETBIOS and Network Operating System in this newest version of LANtastic. It is reported to be up to 300% faster than their previous version under certain conditions. I have not noticed a significant increase in the speed of our LAN, but I have yet to run a good benchmark test on it.

A server can now be run as dedicated or Peer-To-Peer. With the introduction of the ALONE program you can setup a server to run in dedicated mode and switch back to Peer-To-Peer at any time. Using the ALONE program increases the performance of a server while allowing it to process more simultaneous user requests.

Artisoft has upgraded LANcache, their network disk caching program, to be faster and more compatible with Microsoft Windows. When LANcache detects that Windows is running, it will release its own memory to Windows to increase its performance. Because LANcache was written especially for LANtastic, it can greatly increase network performance. It has the ability to cache both reads and writes. LANcache's background writing operations allow the server to perform user requests in the foreground while writing to the disk in the background. If you use the background writes I would recommend putting the server on a UPS in case a power failure occurs before a write to the disk can be completed.

Windows

LANtastic will now support Microsoft Windows 3.0 in real, standard, and enhanced modes on both non-dedicated servers and workstations. Most network vendors have been scrambling to provide true support for Windows and capitalize on its current popularity. Artisoft has released LANtastic for Windows. This version will allow you to access all the functions of your network without having to leave Windows. In addition, LANtastic for Windows can handle multiple tasks and multiple win-

dows. For example, you could setup a window to monitor the status of every server on your network while you chat online to other users. LANtastic for Windows requires LANtastic NOS version 4.xx and Microsoft Windows 3.xx. It currently retails for \$299 per network.

Cost

Artisoft has several different LAN starter kits. They offer a 2 Mbps Starter Kit using two proprietary Cards or a 10 Mbps Kit that contains two AE-2 cards with can be set for industry standard NE2000 emulation. The kits come with 25' of RG58 thin net cable, two 50-ohm terminators, two BNC T connectors, and the NOS with manuals. Current street prices for the kits run about \$349 for the 2Mbps, and \$499 for the 10Mbps. If you already own LANtastic you should upgrade to v4.0 which is available to all registered end users through Artisoft or any authorized Artisoft dealer. The upgrade will cost \$50 per network for networks using Artisoft's 2Mbps or AE-2 Ethernet Adapters. LANtastic/AI (adapter independent NOS) v3.xx single-node users can upgrade to v4.0 for \$99 per network or \$50 for one station.

New Products

I regularly logon the Artisoft BBS (602-293-0065) that is setup to provide technical support, Patches, and product bulletins. I discovered some of the following new product information.

At the PC EXPO on June 25, 1991 Artisoft announced a version of its LANtastic operating system that will provide Novell NetWare users with LANtastic's peer-to-peer features. The software will run on top of Novell and allow the sharing of one nodes resources such as programs, printers, and CD ROMS with any other node. LANtastic for NetWare will also allow Netware users to use Artisoft's Sounding Board adapters to send Voice Mail across the network. It retails for \$499 per network.

Artisoft has announced on August 16, 1991 a low-cost 10Base-T Ethernet Card called the AE-1/T. It supports the use of unshielded twisted pair cabling for Ethernet and is compliant with the IEEE 802.3 standard. They have scheduled to ship it starting in September 1991 for \$199. This card should sell very well due to its low price and the current migration of networks to 10Base-T cabling. Some specifications of the card are as follows:

- Eight-bit ISA bus card for use with PC/XT/AT/386/486 computers
- Can connect to any standard 10Base-T hub with unshielded twisted-pair cabling by its external RJ45 connector
- Has a link-integrity LED to show if proper connection to the hub has been made
- Includes a special internal connector for linking internally to Artisoft's new 10Base-T Peer-Hub
- 10Mbps data transfer rate, supports 8K of on-board RAM
- Network compatible with Novell's NE1000 adapter

Artisoft has announced a new product call Central Station. This hardware allows quick and easy connection to an existing LANtastic or Novell network. You can connect your Laptops or notebooks to the LAN by just plugging in a cable. There is no reconfiguring needed for the network. Some the features are:

- Connects to an existing Artisoft LANtastic or Novell network
- Supports thin and twisted-pair Ethernet cable
- Connects to PC and printer via parallel or serial ports
- Includes two serial, one parallel and one PC interface port
- Allows communication between PC and network
- Allows printing from PC and network to printer
- On-board RAM
- LED displays show power, port activity, and twisted-pair cable link integrity

LANtastic Case Study

One of our divisions had four CAD stations running Autocad 11/386 that they wanted to link together. Each station would need to plot on a JDL Autoplotter and Calcomp 1043 Plotter. They also wanted to have a centralized unattended backup system and the ability to exchange drawing files. I also had a PC in my office downstairs that I wanted to use to perform the administration of the LAN. We chose LANtastic due to it's low cost and good reviews in several popular computer magazines.

We ordered the 10Mbps (Megabytes Per Second) Ethernet Starter Kit and additional cards for all the stations. Artisoft provides you with a complete user and reference guide to their software that has detailed instructions on how to install the

software. Of course I took the manly approach and went ahead without reading the manual. While this has caused me slight trouble in the past with other software, LANtastic's install program was a breeze. It used pop up windows and offered on-line context sensitive help at every step of the process.

A word of warning about installing the software, use the install program. The files are compressed and will not function correctly if just copied over. Since I was using Artisoft's AE-2 10Mbps Ethernet cards, I did not need to buy and load their adapter independent driver software (LANtastic/AI) which is sold separately (at \$99 per network node). I question the practice of charging a fee for using third party cards. This could deter some customers from moving to LANtastic who already own a large number of them and cannot afford the AI drivers.

I began the installation on our spare station. After the install program loaded its files on to the hard drive, it asked me if I was setting up a workstation or a server and what unique name it should be called. I was then given the option of installing default resources and users accounts for the machine (highly recommended). The program then prompted me for confirmation as the current contents of my config.sys file and the lines to be added were displayed. And finally, I was given the option of having the install program create a sample network startup file (also highly recommended). The startup.bat file on my SPARE server looked like this:

```

PATH %PATH%;C:\LANTASTI
C:
rem BRING up AE-2 on adapter 0
C:\LANTASTI\AE-2
C:\LANTASTI\AILANBIO

C:\DOS\SHARE.EXE
C:\LANTASTI\REDIR SPARE LOGINS=10
C:\LANTASTI\SERVER
C:\LANTASTI\NET LOGIN \\SPARE USER
C:\LANTASTI\NET lpt timeout 10

```

: END

Each CAD station was setup as a server in order to share its resources. The network Server program took up 40K which I was able to load into high memory using QEMM 5.1. The spare station would only be needed part time so I used it for the print server. I needed to set up each station to redirect its printer ports to the SPARE server's print queue. Every time a user booted their station they would need to log to all the other stations. So rather than have the user issue the logon and printer redirection commands each time, I included the following command lines in their autoexec.bat files for each server.

```

NET LOGON \\SERVER1 USER
NET ATTACH/VERBOSE \\SERVER1

```

```

NET LOGON \\SERVER2 USER
NET ATTACH/VERBOSE \\SERVER2

```

```

NET LOGON \\SERVER3 USER
NET ATTACH/VERBOSE \\SERVER3

```

```

NET LOGON \\SPARE USER
NET ATTACH/VERBOSE \\SPARE

```

```

NET USE LPT1 \\SPARE\@JDL
NET USE COM1 \\SPARE\@CALCOMP
NET LPT TIMEOUT 10

```

The security features in LANtastic are quite good and can control when a user is allowed to logon and exactly what resources can be accessed. There are also provisions to setup audit trails for each user to record their activity. On this LAN security was not considered vital, so I setup the system with no passwords. The NET ATTACH/VERBOSE statement in their autoexec files will go out and find all available physical drives connected to the server you are logging on to. LANtastic will assign them to the next available drive letter on your machine. This seems good until I needed to determine ahead of time what drive letter on a particular station would be redirected to the SPARE servers hard drive.

The problem was in the way I was

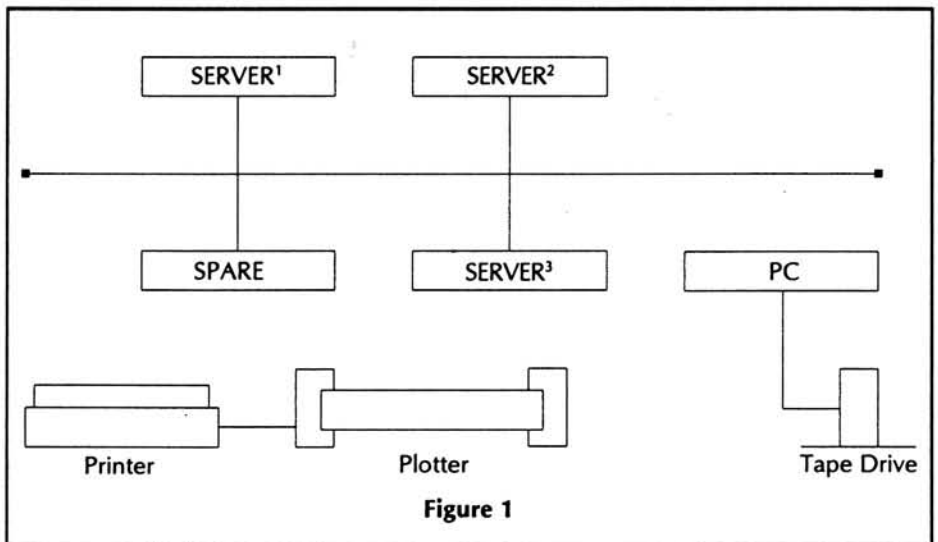


Figure 1

FBE

EaZy PC: EZM128 128K Memory Expansion, \$95; EZCOM Serial Port \$85; EZCOMBO Memory Expansion and Serial Port, \$145

SmartWatch: No-slot calendar/clock module. Software included. For all H/Z PC's, \$32

H/Z-148: ZEX-148 1-1/2 Card Expansion Bus, \$79.95; ZP-148 704K Memory PAL, \$19.95

H/Z-151: VCE-150 removes existing video card, allows use of EGA/VGA card, \$49.95; ZP640+ PAL modifies existing memory card to 640/704K using 256K RAM chips, \$19.95; ULTRA-PAL modifies existing RAM card to 640/704K plus 512K EMS/RAM disk, \$39.95; COM3 kit changes existing COM2 to COM3, allows internal COM2 modem, \$29.95

H/Z-100: ZMF100a modifies old motherboard for 768K memory, \$75; ZRAM-205 converts Z-205 card into 768K RAM disk, \$39

H89: H89PIP two port parallel printer interface card, \$50; Printer cable \$24; SLOT4 adds extra expansion slot to right-side bus, \$39.95

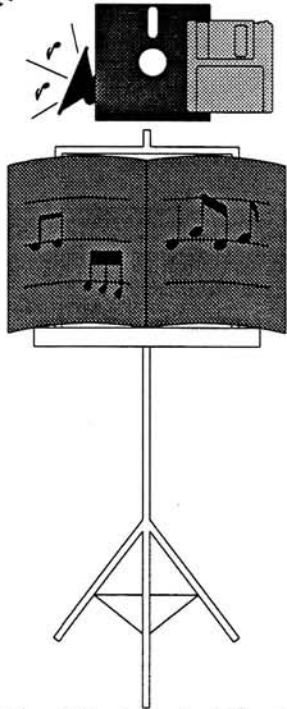
Call Or Write For Additional Information

Order by mail, phone or FAX. VISA/MasterCard/AMEX accepted. No charges for UPS Ground or USPS shipping. WA residents add 8.2% tax. Hours: M-F 1-5 PM Pacific. We return all calls left on our answering machine!

FBE Research Company, Inc.

P.O. Box 68234, Seattle, WA 98168
206-246-9815 Voice/FAX TouchTone Selectable

New Software Product!



The Electronic Clavier
P/N 885-6016

attaching to the various servers. When LANtastic started executing my login and attach commands, I assumed that all servers would be up and running all the time. When LANtastic was unable to find a server it skipped to the next command. This meant that my drive letters were not always assigned in the same order. My drive C may be G to you one day and H to you the next. This was extremely bothersome when trying to write batch files to move data to a particular station on the LAN or setup Autocad to send plot files to the spare station. The solution I discovered was to specify the drive letter I want redirected. For example:

```
NET LOGON \\SERVER3 USER
NET USE F \\SERVER3\C-DRIVE
```

```
NET LOGON \\SPARE USER
NET USE G \\SPARE\C-DRIVE
```

Initially I wanted to set up the spare station with a tape drive to handle the backups. I found out later that the station could not backup itself when the server program is running. This fact, along with my desire to store the backup tapes physically away from the station in case of fire, prompted me to move the tape drive downstairs to a PC in my office and run their backups from there. I would change the tape each day and once a week send them off site according to a rotation schedule with the disk packs from our mini computer.

One of the constants in the computer universe seems to be the speed at which your printer prints. In order to avoid waiting on the printer to complete your output, LANtastic like most other networks will redirect your print output to a file in the print server's queue. This worked alright in the beginning, but as time went on I noticed a delay if two or more stations were spooling at the same time to the print server. The hard drive in the print server was a 120 MB IDE. It had a fairly quick access time (16ms), however it could only process one write request at a time and consequently slowed down the stations sending data to the print server. When the spare station that was used as the print server was needed to run Autocad there was also a noticeable slow down during any disk access as other stations sent over their spool files.

In order to avoid this hard drive bottleneck I set up the SPARE station with a 4 MB RAM Drive to store the plot files until they could be plotted. This was surprisingly easy due to LANtastic's flexibility in locating its print queue. When LANtastic's server is run it will automatically create the appropriate directories on the RAM drive for the print queue. I tried a 3 MB RAM drive size but ran into trouble as it soon filled up. A typical Autocad print file for this system was 350,000 -550,000 K. The following command to create the RAM drive was

placed in the spare stations config.sys file.
DEVICE=C:\DOS\RAMDRIVE.SYS 4096 /A

When you setup your network, you should run net_mgr on the machine you will use as the print server in order to set the number of printer tasks that can be run at the same time (i.e. devices hooked up, set the number at two). The maximum number you can set is five (which represents LPT1, LPT2, LPT3, COM1, and COM2).

The network has been running for almost a year now without much trouble. The times when I did encounter difficulty it turned out to be a cable-related problem. If you install your own cables you will quickly learn to take your time and make each termination perfect. Because if you don't you will be plagued by intermittent connection problems in the future. I used RG-58A/U 50-ohm cable for the network. In order to make your life as a network administrator go smoothly, be sure to check on the rules for installing ethernet cable. If your cable is too close to a power cord or florescent light, you are asking for trouble.

Most of the changes I make to LANtastic now are done to try and optimize its performance. If in the future I need to switch to a different Network Operating System or connect to a Novell LAN, the AE-2 cards from Artisoft can emulate a Novell NE2000 Card. DOS based LANs run slower than ones that have their own multitasking operating systems such as Novell. However, DOS based LANs like LANtastic are less complex to setup, will deliver good performance, and are less expensive. Considering what the Artisoft starter kit costs and the immediate benefits you can receive from a LAN I would recommend LANtastic to anyone.

For more information about the Artisoft product line, contact:

Artisoft, Inc.
Artisoft Plaza
575 E. River Road
Tucson, AZ 85704
Telephone 602-293-6363

Or some of the following Artisoft dealers:

Connecting Point Computer Center
61045 U.S. 31 South
South Bend, IN 46614
(219) 291-1196

Reese Equipment Co.
523 E. Jefferson Street
Plymouth, IN 46563
(219) 936-2523 *

Son of

Small Computer System Interface

Nathan E. Baker
Technical Writer
Zenith Data System

The SCSI-2 specifications are the second round in the ongoing evolution of the small computer system interface. Those of you who are avid REMark readers will recognize this article as a continuation of my last article, in which I explained SCSI-1. In this article I will explain the SCSI-2 specifications, and since SCSI-2 is so closely related to SCSI-1, I'll explain some of the commands, messages, and status information that is common to both interfaces.

The three key objectives in developing SCSI-2 where:

- Provide host computers with device independence within a class of devices, so different drives (tape, disk, optical) and devices can be connected without changing the hardware requirements of the host computer.
- Remove device-dependent intelligence from SCSI-2 devices. Software or firmware in the host computer can get all the required information from the device, such as type of device, characteristics, and changeable parameters (if any). Further requests to the device can be made to determine other characteristics. Data that is not required to run the operating system or initialize the host are kept within the device itself.
- Provide compatibility with SCSI-1 devices. SCSI-1 and SCSI-2 devices, initiators and targets, can share the same bus and will not erroneously respond to commands and messages out of their genre.

SCSI-1 vs SCSI-2: Differences and Advantages

SCSI-2 is basically an expanded version of SCSI-1. The main differences and advantages are as follows:

- A common command set (CCS) has been formally approved. A subset of SCSI commands, the CCS further defines and extends commands for mass

storage devices, scanners, optical disks, and communication devices.

- Command queuing is implemented.
- Contingent allegiance and extended contingent allegiance methods are implemented.
- Asynchronous event notification is supported.
- Data bus parity and arbitration are now mandatory.
- Support for more messages is mandatory.
- The data path is 32-bits wide, which allows for the wide SCSI option.

Before discussing these subjects, I'd like to lay the groundwork for the rest of this article by explaining the SCSI-2 hardware, and refresh your memory about how SCSI operates.

Basic Hardware Information I: The Cables

The single-ended and differential cable configurations that are used in SCSI-1 are also part of SCSI-2. SCSI-2 uses two cables, an A cable and a B cable, which are both required if the system is going to use the wide-SCSI option. The two cables have the following descriptions:

A Cable: 50-conductor flat or 25-conductor twisted-pair.

B Cable: 68-conductor flat or 34-conductor twisted-pair.

Note: For differential configurations, the twisted-pair cables are strongly recommended.

The pinouts for the connectors are given in Tables 1 and 2.

Basic Hardware Information II: The Devices

The cable configuration dictates the type of devices that can be connected to the SCSI bus. Single-ended SCSI devices can only be used with a single-ended cable; differential SCSI devices can only be used

with a differential cable.

Besides the SCSI cables, a complete SCSI system requires a host computer (which contains a SCSI processor, sometimes called a host adapter) and at least one SCSI device connected to the bus. There may be up to eight SCSI devices (the host computer plus seven SCSI peripherals) connected to the same bus.

The bus provided by the host computer is sometimes called the primary bus. If a SCSI peripheral contains a SCSI processor, it may provide a secondary SCSI bus. There could be up to seven secondary SCSI busses attached to the primary bus. Each secondary bus can connect with up to seven SCSI peripherals, so the maximum number of SCSI peripherals controlled by a single host computer is 56. All the SCSI peripherals are controlled directly (if they are attached to the primary bus) or indirectly (if they are attached to a secondary bus) by the host computer.

It is possible to have multiple host computers connected to the primary bus. Optional extended SCSI commands can then be incorporated, and up to 2048 devices (or routines) can be controlled by a single device under control of one of the host computers.

When the bus is in use, only two SCSI devices can communicate with each other at a time. SCSI devices are classified as initiators or targets. Initiators send commands; targets execute the commands (which are usually I/O processes). Some devices may serve dual roles.

Each target can contain up to eight logical units, beginning with logical unit number zero. Logical units are usually mapped directly to a device. In some cases logical units can be made up of portions of several different devices.

Additionally, each target can provide up to eight target routines, which are processes that execute directly on the target

and are not associated with a particular logical unit. The first target routine is always target routine number zero.

Basic SCSI Operation

SCSI-2 uses eight bus phases to perform an operation (note that these are the same phases used in SCSI-1). These eight phases are:

Bus free — Indicates no SCSI device is accessing the bus. This phase always follows a reset.

Arbitration — Allows SCSI devices to compete for access to the bus. This is an optional phase in SCSI-1, but mandatory in SCSI-2. For an explanation of how arbitration works on the SCSI bus, refer to my previous article.

Selection — Allows an initiator to connect to a target.

Reselection — Allows a target that has disconnected from the bus to reconnect to the bus.

Command — Allows the target to request a command from the initiator.

Data I/O — Allows data transfer between target and initiator.

Status — Allows the target to send status information to the initiator.

Message (out/in) — Allows messages to pass between the target and the initiator. A message out phase sends information from the initiator to the target. A message in phase sends information from the target to the initiator. The message out/in phase can be optional or mandatory, depending on the SCSI implementation, the target peripheral, and the operation involved.

Each bus phase is distinct and separate. The eight phases interact with each other in different ways, depending on the SCSI operation being performed, and if messages are used. A typical SCSI operation uses bus phases in the following sequence:

1. Bus free
2. Arbitration
3. Selection
4. Message out
5. Command
6. Data I/O
7. Status
8. Message in
9. Bus free

Refer to my last SCSI article for timing criteria and a more in-depth explanation of how these bus phases interact.

The Common Command Set (CCS)

The CCS is made up of commands that are mandatory, optional, vendor specific, and reserved.

- Mandatory commands must be supported by initiators and targets if they are to be considered SCSI-2 compatible.
- Optional commands can be implemented when necessary.

Table 1
SCSI-2 A Cable Pinout
(Single-Ended and Differential Configurations)

Pin Number	Single-Ended ¹	Differential
1	Ground	Shield ground
2	-DB0	Ground
3	Ground	+DB0
4	-DB1	-DB0
5	Ground	+DB1
6	-DB2	-DB1
7	Ground	+DB2
8	-DB3	-DB2
9	Ground	+DB3
10	-DB4	-DB3
11	Ground	+DB4
12	-DB5	-DB4
13	Ground	+DB5
14	-DB6	-DB5
15	Ground	+DB6
16	-DB7	-DB6
17	Ground	+DB7
18	-DB(P)	-DB7
19	Ground	+DB(P)
20	Ground	-DB(P)
21	Ground	DIFFSENS
22	Ground	Ground
23	Ground	Ground
24	Ground	Ground
25	— ²	TERMPWR
26	TERMPWR	TERMPWR
27	Ground	Ground
28	Ground	Ground
29	Ground	+ATN
30	Ground	-ATN
31	Ground	Ground
32	-ATN	Ground
33	Ground	+BSY
34	Ground	-BSY
35	Ground	+ACK
36	-BSY	-ACK
37	Ground	+RST
38	-ACK	-RST
39	Ground	+MSG
40	-RST	-MSG
41	Ground	+SEL
42	-MSG	-SEL
43	Ground	+C/D
44	-SEL	-C/D
45	Ground	+REQ
46	-C/D	-REQ
47	Ground	+I/O
48	-REQ	-I/O
49	Ground	Ground
50	-I/O	Ground

Notes

1. A minus symbol (-) in this column indicates an active-low signal.
2. Pin 25 is left open to protect the host computer's power supply. If a single-ended peripheral is accidentally plugged into a differential SCSI bus, it will not short TERMPWR (terminal power) to ground.

- Vendor specific commands can be used for unique functions or processes in target peripherals, depending on the intent of the peripheral vendor.
- Reserved command codes are intended

for possible future implementation, and should not be used.

The SCSI-2 specifications contain many more commands than were previously defined in SCSI-1. Many commands are

Table 2
SCSI-2 B Cable Pinout
(Single-Ended and Differential Configurations)

Pin Number	Single-Ended ¹	Differential	Pin Number	Single-Ended ¹	Differential
1	Ground	Ground	35	Ground	Ground
2	Ground	+DB8	36	-DB8	-DB8
3	Ground	+DB9	37	-DB9	-DB9
4	Ground	+DB10	38	-DB10	-DB10
5	Ground	+DB11	39	-DB11	-DB11
6	Ground	+DB12	40	-DB12	-DB12
7	Ground	+DB13	41	-DB13	-DB13
8	Ground	+DB14	42	-DB14	-DB14
9	Ground	+DB15	43	-DB15	-DB15
10	Ground	+DB P1	44	-DB P1	-DB P1
11	Ground	+ACKB	45	-ACKB	-ACKB
12	Ground	Ground	46	Ground	DIFFSENS
13	Ground	+REQB	47	-REQB	-REQB
14	Ground	+DB16	48	-DB16	-DB16
15	Ground	+DB17	49	-DB17	-DB17
16	Ground	+DB18	50	-DB18	-DB18
17	TERMPWRB	TERMPWRB	51	TERMPWRB	TERMPWRB
18	TERMPWRB	TERMPWRB	52	TERMPWRB	TERMPWRB
19	Ground	+DB19	53	-DB19	-DB19
20	Ground	+DB20	54	-DB20	-DB20
21	Ground	+DB21	55	-DB21	-DB21
22	Ground	+DB22	56	-DB22	-DB22
23	Ground	+DB23	57	-DB23	-DB23
24	Ground	+DB P2	58	-DB P2	-DB P2
25	Ground	+DB24	59	-DB24	-DB24
26	Ground	+DB25	60	-DB25	-DB25
27	Ground	+DB26	61	-DB26	-DB26
28	Ground	+DB27	62	-DB27	-DB27
29	Ground	+DB29	63	-DB28	-DB28
30	Ground	+DB29	64	-DB29	-DB29
31	Ground	+DB30	65	-DB30	-DB30
32	Ground	+DB31	66	-DB31	-DB31
33	Ground	+DB P3	67	-DB P3	-DB P3
34	Ground	Ground	68	Ground	Ground

Note:

1. A minus symbol (-) in this column indicates an active-low signal.

tailored to certain families of devices, but some commands are common to many device types. Table 3 lists the four mandatory commands for all SCSI-2 devices, as well as some commands that are common among the majority of devices.

The Command Descriptor Block

Initiators transfer commands to targets by sending a command descriptor block during the COMMAND phase. (Commands are always sent across the 8-bit data bus, DB0 - DB7, to maintain compatibility with SCSI-1 systems.) For some commands, the command descriptor block is followed by a list of parameters which are sent during the DATA OUT phase. The length of the command descriptor block can vary (to a maximum of twelve bytes), but the first byte is always the operation code, and the last byte is always the control byte.

The operation code, shown in Table 3

and defined in Table 4, contains two important pieces of information; the group code and the command code.

The group code indicates the size of the command, while the command code specifies the actual operation. (A list of all command codes and explanations is too big for this article. Look for that information in upcoming articles.)

Commands fall into certain groups, indicated by the group code. There are six-byte commands, ten-byte commands, twelve-byte commands, and vendor specific commands. The initiator indicates what group the command is by the group code, as shown in Table 5. Note that some group codes are reserved for future use.

The following paragraphs explain a simple command descriptor block for a six-byte command.

Byte 0 — The first byte is always the operation code, which contains the group

code and the command code.

Byte 1 — This byte is the logical unit number byte. It is the same for six-, ten-, and twelve-byte commands, and is explained in Figure 1.

LUNTRN: Logical Unit Target Number — These bits indicate which logical unit number is the target for this command (if LUNTRN = 0), or they can be used as part of a logical block address (see the following explanation for bytes 2 and 3).

LUNTRN: Logical Unit Target — If 1, the command is directed at a target routine. If 0, the LUNTRN field indicates what logical unit number the command is directed at.

DISCPRIV: Disconnect Privilege — If 1, the target may disconnect from the bus. If 0, the target cannot disconnect until the I/O transaction is completed.

IDENTIFY: Identify Message — If 1, this bit indicates that the accompanying descriptor block is actually an Identify mes-

Table 3
Common Commands for all Devices

Command	Operation Code	Support
Change Definition	40H	Optional
Compare	39H	Optional
Copy	18H	Optional
Copy and Verify	3AH	Optional
Inquiry	12H	Mandatory
Log Select	4CH	Optional
Log Sense	4DH	Optional
Read Buffer	3CH	Optional
Receive Diagnostic Results	1CH	Optional
Request Sense	03H	Mandatory
Send Diagnostic	1DH	Mandatory
Test Unit Ready	00H	Mandatory
Write Buffer	3BH	Optional

Note:

1. This command can be sent in a number of ways. The only mandatory support required is the self-test option. All SCSI-2 targets must support the send diagnostic command when the self-test option is set. Other support is optional.

Table 4
Operation Code

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
X	X	X	X	X	X	X	X
[—Group code—]			[—Command code—]				

sage, and not a true command. This message is invalid if either RESERVED bit is 1, or the LUNTAR bit is 1 and the target does not implement target routines.

Note: It is mandatory for SCSI-2 compatible devices to support the Identify message.

Byte 2 and Byte 3 — These bytes are combined with bits 0 - 4 of byte 1 to indicate the logical block address. Bit 4 of byte 1 is the most significant bit, and bit 0 of byte 3 is the least significant bit.

For logical units, or within a partition on device volumes, the logical block address begins with block zero and is contiguous up to the last logical block on that logical unit or within that partition.

For a six-byte command the logical block address is 21-bits long (the sixteen bits of byte 2 and byte 3, plus the five bits from byte 1).

For ten-byte and twelve-byte commands, bits 0 - 4 of byte 1 are ignored (they are considered to be reserved), and the logical block address is 32-bits long. (Byte 2 and byte 3, plus an additional two bytes that are part of the command descriptor block.)

Note: The logical block address is sent only if required. The exact length of the logical block address can be altered for

twelve-byte commands, the information can be transferred in multiple bytes (up to four bytes for a twelve-byte command — byte 4 plus an additional three bytes).

Transfer length specifies the amount of data (usually in blocks) to be transferred. When the transfer length is specified by a single byte (for six-byte commands), up to 256 blocks of data can be transferred by one command. A value of 00H indicates 256 blocks are to be sent; 01H - FFH indicates 1 - 255 blocks are to be sent.

When the transfer length is specified by multiple bytes (for ten-byte and twelve-byte commands), a value of zero indicates

Table 5
Group Codes

Bit 7	Bit 6	Bit 5	Command Category
0	0	0	Six-byte command
0	0	1	Ten-byte command
0	1	0	Ten-byte command
0	1	1	Reserved
1	0	0	Reserved
1	0	1	Twelve-byte command
1	1	0	Vendor specific command
1	1	1	Vendor specific command

specific purposes, depending on the command.

Byte 4 — This byte indicates the transfer length, the parameter list length, or the allocation length. What information is transferred depends on the command. For six-byte commands,

the information is transferred in a single byte. For ten-byte and

no blocks of data are to be sent. A value greater than zero tells the target to send that number of data blocks (0001H for one data block, 0002H for two data blocks, 0032H for 50 data blocks, etc.)

Parameter list length specifies the number of bytes sent during a DATA OUT phase. This information is typically used in command descriptor blocks that send mode parameters, diagnostic parameters, or other parameters needed by the target. A value of zero indicates no data is transferred.

Allocation length specifies the maximum number of bytes the initiator has allocated for returned data. A value of zero indicates no data is transferred. The target will terminate the DATA IN phase when all the available data has been sent, or the maximum allocation length has been reached, whichever occurs first.

Byte 5 — The last byte of a command descriptor block is always the control byte. It is the same for six-, ten-, and twelve-byte commands, and is explained in Figure 2.

LINK: This bit is used to continue the I/O process across several commands. When 1, the target enters the COMMAND phase again upon successful completion of the current command.

FLAG: This bit is optional, and can be used to tell the target to send certain messages. This bit should be 0 if the LINK bit is 0. If the FLAG bit is 1 and the LINK bit is 0, the target will return a CHECK CONDITION status.

When LINK is 1 and FLAG is 0 — the command completes successfully — the target sends the LINKED COMMAND

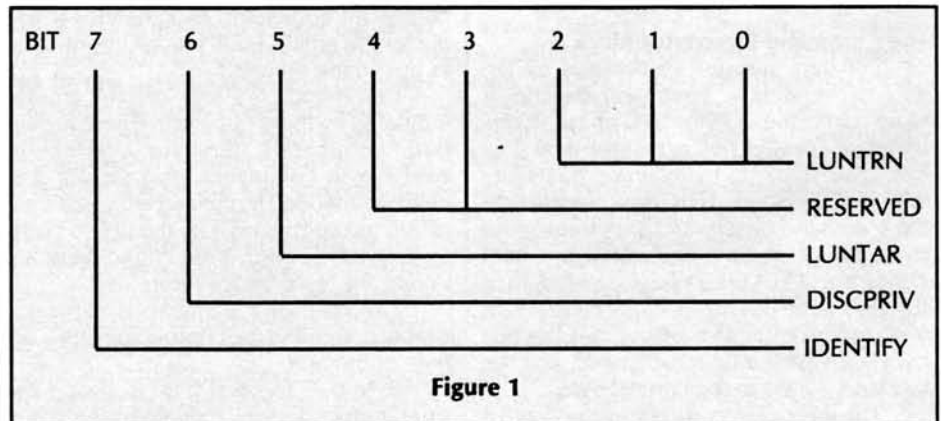
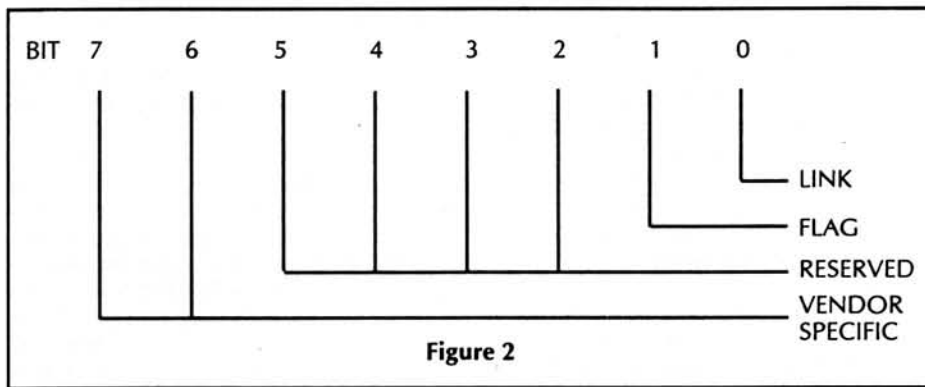


Figure 1



COMPLETE message.

When LINK is 1 and FLAG is 1 – and the command completes successfully – the target sends the LINKED COMMAND COMPLETE (WITH FLAG) message.

Command Queuing

The SCSI-2 specifications provide the option of command queuing. Command queuing allows a target to accept more than one command, and then execute them at a later time.

Two methods of command queuing can be implemented; tagged or untagged.

During untagged queuing, a target can accept a command from an initiator for each logical unit or target routine, while the command from another initiator is being executed. For this method of command queuing, only one command for each logical unit or target routine can be in the queue at a time. (Some SCSI-1 compatible devices may also support untagged queuing.)

During tagged queuing, a target can accept multiple commands from the same or different initiators for each logical unit or target routine, until the target's command queue is full. If the target device utilizes an optimization algorithm, it may rearrange the commands so that they execute in the most efficient order.

Contingent Allegiance and Extended Contingent Allegiance

When an error or unexpected result occurs in a target device, a contingent allegiance condition can occur. When such a condition occurs, the initiator sends a Request Sense command. The target responds by sending a block of sense data that contains information about the target's condition. The initiator can use the data for error correction, if possible.

The target preserves the sense data as long as the contingent allegiance condition exists. If the target is unable to preserve separate groups of sense data, any attempted connection by another initiator will be refused (the target will indicate a busy status). Execution of queued commands in a target routine or a logical unit will be suspended until the contingent alle-

giance condition is cleared by further commands from the original initiator, or a hard reset, an abort message, or a device reset message.

Supporting extended contingent allegiance is optional. An extended contingent allegiance condition occurs when the target sends an Initiate Recovery message to the initiator after a Check Condition Status or a Command Terminated status code. If another initiator tries to connect to the target, the target responds with a busy status.

During an extended contingent allegiance condition, the target will only respond to untagged I/O processes that come from the initiator that sent the Initiate Recovery message. If the initiator sends a queue tag message the target responds with a Queue Full status code.

Execution of queued commands in the target is suspended until it receives a release recovery message from the original initiator.

An extended contingent allegiance condition can also occur during asynchronous event notification.

Asynchronous Event Notification (AEN)

Asynchronous event notification (AEN) informs the processing device that an asynchronous event has occurred. AEN is optional, and can be used for the following functions:

1. To inform the processor device (an initiator) that an error has occurred after the target completed a command.
2. To inform all processor devices on the SCSI bus that a newly initialized device is ready for operation.
3. To inform all processor devices on the SCSI bus of unit attention conditions, or a miscellaneous asynchronous event has occurred.

Note: Target routines should not use AEN.

An example of the first case, informing a processor device of an error after a command completed, could occur in a mass storage device that implements a write cache. If the data is written to the cache, but for some reason can't be written to the media, the device could use AEN to

notify the initiator that an error has occurred. This condition may result in an extended contingent allegiance, in which case some kind of error correction method could be implemented.

AEN can also be used when there are multiple processor devices or host computers on the bus. The processor device (or host computer) informs the bus that a new device is ready, or a unit attention condition exists, or an asynchronous event can occur, by using commands that specify AEN. Example of such commands are Send and Inquiry, which are explained in the following paragraphs.

Send (operation code 0AH) – An initiator uses this command to tell a processor device to accept a data transfer. The Send command is implemented in one of two ways, depending on the state of the AEN bit, which is bit 0 of the second byte of this command. If AEN = 0, the data transferred is vendor specific. If AEN = 1, the command conforms to the AEN data format, which indicates the following:

- If the asynchronous event occurred on a target routine or a logical unit.
- Which target routine or logical unit experienced an asynchronous event.
- Error codes, segment numbers, command specific information, and other information that may be required, depending on the processor device or command.

Inquiry (operation code 12H) – An initiator uses this command to request target parameters and information about peripherals attached to the target. The target responds by sending a packet of data to the initiator. The data packet can be up to 96 bytes long. The information contained in the packet varies, depending on the target's vendor and capabilities. The first 36 bytes of the data packet are defined in the SCSI-2 specifications. The remaining bytes, if used, contain vendor specific information (bytes 37 - 55), or are reserved for future standardization (bytes 56 - 95). If the target is a processor device and supports AEN, bit 7 of the fourth byte in the data packet is 1. If it does not support AEN, bit 7 is 0. (If the target is not a processor device, bit 7 is reserved.)

Operations that occur after an AEN differ, depending on the peripheral, the operation being performed, and subsequent commands.

Messages and Status

Messages communicate information between the initiator and the target for the purpose of managing an operation. The STATUS phase indicates the result of an operation. The STATUS phase transfers a single status byte to the initiator, whereas the MESSAGE IN/OUT phase can transfer multiple bytes of information.

During the MESSAGE IN phase, the

target can send a message to the initiator. During the MESSAGE OUT phase, the target requests a message from the initiator. One or more messages can be sent during a message phase, but a message cannot be split up over multiple message phases.

Messages vary in length, depending on the information they convey. The first byte of a message indicates how long the message will be (except for extended messages).

One-Byte Messages— These messages are indicated by the first byte being 00H, or if it falls in the range 02H - 1FH or 80H - FFH.

Two-Byte Messages— These messages are indicated by the first byte being in the range 20H - 2FH.

Extended Messages— These messages are indicated by the first byte being 01H.

The second byte indicates how long the message is, in bytes. The third byte contains the extended message code, and the following bytes contain message arguments.

Reserved Messages— These messages are indicated by the first byte being in the range 30H - 7FH.

The following list of messages is for reference only. An explanation of each one is too long for this article, and is therefore not included.

The status byte, sent during the STATUS phase, is normally transferred after the completion of a command, unless the command is terminated by one of the following conditions:

- An abort message
- An abort tag message
- A bus device reset message
- A clear queue message

- A hard reset
- An unexpected disconnect.

The status byte contains information about the status of an I/O process, and conforms to Figure 3.

SBC: Status Byte Code— This five-byte code indicates the status condition. The codes are shown in Table 6 and the conditions are explained in the following paragraphs. Codes not shown are reserved.

Good— This code indicates that the command has been successfully completed.

Check Condition— This code indicates that a contingent allegiance condition exists.

Condition Met— This code indicates that the conditions for a requested operation are satisfied.

Busy— This code is used to indicate that the target is busy. This code is returned when a target is unable to accept a command from an otherwise acceptable initiator.

Intermediate— This code is returned for every successfully completed command during a series of linked commands (except for the last command), unless the command is terminated by a Command Terminated, Check Condition, or Reservation Conflict status code. If the Intermediate status code or the Intermediate-Condition Met status code is not returned, the series of commands is terminated and the I/O process is stopped.

Intermediate-Condition Met— This code is returned for every successfully completed command during a series of linked commands (except for the last command) that require certain conditions to be satisfied, unless the command is terminated by a Command Terminated, Check Condition, or Reservation Conflict status code. If the Intermediate-Condition Met status code or the Intermediate status code is not returned, the series of commands is terminated and the I/O process is stopped.

Reservation Conflict— This code is returned if an initiator tries to access a logical unit or a process in a logical unit that is already reserved with a conflicting reservation type for another SCSI device.

Command Terminated— This code indicates that the target has terminated an I/O process after receiving a Terminate I/O Process message. A contingent allegiance condition

**Table 5
Messages**

Message	Code	Support ¹	
		Init.	Tar.
Abort	06H	O	M
Abort Tag	0DH	O or M ²	O or M ²
Bus Device Reset	0CH	O	M
Clear Queue	0EH	O or M ²	O or M ²
Command Complete	00H	M	M
Disconnect	04H	O	O
Identify	80H - FFH	M	O or M ³
Ignore Wide Residue	23H	O	O
Initiate Recovery ⁴	0FH	O	O
Initiator Detected Error	05H	M	M
Linked Command Complete	0AH	O	O
Linked Command Complete (with flag)	0BH	O	O
Message Parity Error	09H	M	M
Message Reject	07H	M	M
Modify Data Pointer	xxH ⁵	O	O
No Operation	08H	M	M
Head of Queue Tag	21H	O	O
Ordered Queue Tag	22H	O	O
Simple Queue Tag	20H	O	O
Release Recovery	10H	O	O
Restore Pointers	03H	O	O
Save Data Pointer	02H	O	O
Synchronous Data Trans. Req.	xxH ⁵	O	O
Wide Data Transfer Request	xxH ⁵	O	O
Terminate I/O Process	11H	O	O
Reserved	12H - 1FH		
Reserved	24H - 2FH		
Reserved	30H - 7FH		

Notes:

1. O = optional, M = mandatory.
2. Support is mandatory if tagged queuing is implemented.
3. If the target implements a MESSAGE OUT phase, support for this message is mandatory. Codes 80H - FFH are all used for various identify messages, which are all one-byte long. The format for the identify message byte matches byte 1 of the command descriptor block (see previous section).
4. During a MESSAGE OUT phase, this message is valid only if AEN is implemented.
5. These are extended messages, which all start with 01H. For an example of an extended message, see the following section, "Wide SCSI".

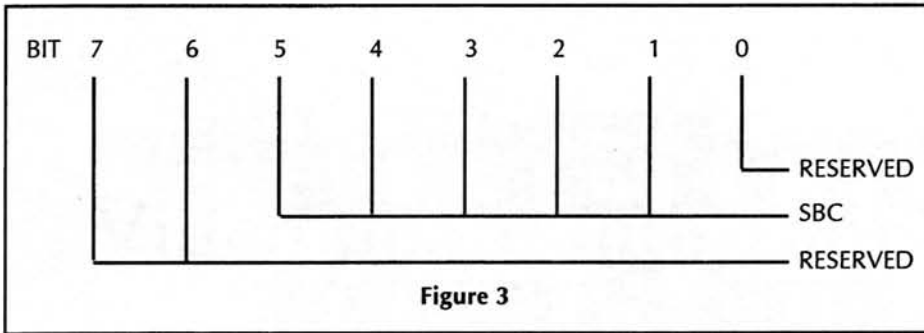


Figure 3

negotiating for synchronous data transfer. If the wide data transfer request occurs after the synchronous data transfer request, an asynchronous data transfer will occur instead.

A SCSI Eye to the Future

I hope the smattering of information I've presented has helped you understand what SCSI can do and how it operates. Future articles will probe more commands, messages, and provide examples of how a an I/O process is carried out, from start to

exists when this status is returned.

Queue Full — This code is used only if tagged queuing is implemented, and is returned when a Simple Queue Tag, Ordered Queue Tag, or Head of Queue Tag message is received and the queue is full. The message is not placed in the queue, and must be sent at a later time.

Wide SCSI

SCSI implementations can be SCSI-2 compatible and use only the A cable. As more manufacturers are taking advantage of the SCSI, many are choosing to implement only the A cable. However, such designs cannot use the option of wide data transfer. To implement wide data transfers, which allow the real speed potential of SCSI-2 to be realized, the A cable and the B cable must both be used.

Negotiation for a wide data transfer can happen whenever a previously arranged data transfer width agreement has become invalid. The negotiation itself occurs during the MESSAGE OUT/IN phase, with the Wide Data Transfer Request message, which is defined in the following paragraphs.

Wide Data Transfer Request Message

Byte 0 — The first byte of this message is 01H, which indicates that it is an extended message.

Byte 1 — The next byte indicates the extended message length, which in this case is 2 bytes long (02H).

Byte 2 — This byte is the wide data transfer request code, 03H.

Byte 3 — This byte specifies the requested width of the wide data transfer. 00H = 8-bits, 01H = 16-bits, and 02H = 32-bits. Values greater than 02H are reserved for future use.

After the Wide Data Transfer Request message is received, the target and initiator communicate back and forth to negotiate a mutually beneficial agreement. During this time the specified transfer width may be altered. The agreement becomes valid when the target exits the MESSAGE OUT phase, indicating that it has accepted the wide data transfer request (which may have been reset to 8-bits as a result of the negotiation).

If a SCSI device implements the syn-

Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Status Condition
0	0	0	0	0	Good
0	0	0	0	1	Check Condition
0	0	0	1	0	Condition Met
0	0	1	0	0	Busy
0	1	0	0	0	Intermediate
0	1	0	1	0	Intermediate-Condition Met
0	1	1	0	0	Reservation Conflict
1	0	0	0	1	Command Terminated
1	0	1	0	0	Queue Full

chronous data transfer option as well as the wide data transfer option, then it should negotiate for wide data transfer before

finish. And of course, I'll present SCSI-3 when the ANSI gestation period is over.*



WordStar

Past and Present

William Warren Pitts
Dragonrose Enterprises
Box 315
Bowman, GA 30624

Who Am I?

I am a member of W/PUG, a WordStar users group with more than 18,000 members. I write a monthly column for Scroll newsletter. Our most famous columnist is Dave Ketchum, best known for his role as "Agent 13" on the old TV show "Get Smart." (When I meet him, I'm going to ask how he got into those mail boxes, fire hydrants, and impossibly tight places.)

It's a long way from Dave's Marina Del Ray home, to my office in exurbia in north Georgia, to nearly three million other registered users of WordStar worldwide. Yet there's one thing we share: a dedication to using WordStar to get our writing done. For writing documents of any kind and size, I've never found anything to beat it.

Another true thing about WordStar users is we never tire of searching out even more Tips & Tricks for using it. No one can ever write a manual covering every possible use of every WordStar option and feature. This is where users groups fill a niche. They give folks a place to go with unanswered questions, and a forum for showing off spiffy solutions to tricky problems. For information about joining W/PUG and subscribing to Scroll newsletter, contact:

Dr. David Rafky, editor
Scroll W/PUG Newsletter
P.O. Box 16-1443
Miami, FL 33116
(305) 274-0099/FAX 271-8904

The name of my column is Notes From the Commander's Corner. Dave Rafky drew the name from the fact I've authored a book titled Commanding WordStar. It was my third book on WordStar. I'm currently hard at work on a fourth — an upgrade to cover the next release — so you might say WordStar is my bread-and-butter.

I've written books about other word processors, namely Microsoft Word and

MultiMate Advantage II. The most telling fact about those two programs is I wrote about them using WordStar. I may be a Technical Writer, but I'm not crazy.

Introduction to WordStar 6.0

For those who haven't seen WordStar since release 4.0 or earlier, expect a new world of word processing. WordStar has gone through so many changes, old-time users might feel they don't recognize it. If you're thinking about upgrading, take heart: everything you ever loved about WordStar is still there; the same commands make features work. The changes you'll find are genuine improvements.

WordStar History Revisited

It wasn't long ago when WordStar was the Number One selling word processor. It was a Hard Charger that got the job done, and got it done right. It was an Old Faithful you could depend on. Even today, there are more WordStar users worldwide than most of its competition combined. Yet right at the peak of its glory, bad decisions by management sent MicroPro riding off on new horses — leaving Old Faithful chafing at the bit in the stables.

Those were heady days for personal word processing. The empire of "dedicated" word processing machines was beginning to crumble. WANG and IBM had long dominated with expensive systems — especially IBM's big as desks that only wrote letters, contracts, form documents, and addressed envelopes. They were clunky, and hard to use, but they were WANG and IBM — and back then it was prestigious to own them.

WordStar showed up as a line editor for mainframe computers. It came into its own as THE hot shot word processor for CP/M personal computers. (You remember CP/M, the operating system every-

body loved to hate?) WordStar was one of the first to give on-screen formatting in 80 columns, and display a page break line on-screen. Quick and easy for writing and printing plain documents, folks naturally fell in love with it.

IBM, in its last successful bid to set an industry standard, introduced its own PC. Thinking to dominate word processing software too, IBM downloaded its dedicated word processing software to run in PCs, and renamed it DisplayWrite. For a brief while, IBM succeeded.

At this point, management at MicroPro International made their big mistake. Though WordStar had successfully transferred to MS-DOS and was selling well, they decided the future was in word processors like DisplayWrite. Their fateful decision not to upgrade WordStar nearly destroyed them. They bet the farm on two products: WordStar 2000 and Easy.

WordStar 2000 was a true clone of DisplayWrite: just as gargantuan; just as maze-like and murky. People said "include me out" and stayed away in droves. Both products fell into the black hole of software and disappeared. Easy was only a cut-down version of WordStar, lacking most of its complicated features. It managed to insult most of the word processing market.

Meanwhile, the word processing world was undergoing revolution. Microsoft Word and WordPerfect appeared at opposite ends of the spectrum. They offered entirely new writing environments — and added a growing list of bells and whistles to word processing.

Microsoft originally wrote Word for the Macintosh, and then downloaded it to work on IBM PCs. It lost the Mac ToolKit, the high-resolution Mac screen and Graphic User Interface. Yet it kept the mouse, and you could still point-and-click to your heart's content. Though exceptionally powerful,

Word is the most complicated program you can buy. It is slow, and you can easily get lost in its features.

WordPerfect took the opposite approach, the ultimate in Spartan simplicity. When you boot it up, you get a blank screen, a blinking cursor, and all major commands installed permanently on function keys. Just slap a command template over the keyboard and jump in with both feet. It didn't take WordPerfect long to dance its way to the top of the market.

MicroPro International kept sinking deeper and deeper into mire. Its best WordStar programmers quit in protest, and formed their own company. They released "New Word" — essentially WordStar with improvements they'd demanded. Around 1985, MicroPro bought "New Word," made a few more enhancements, and released it as WordStar 4. A buddy of mine called it "Word processing from God." Legions of fanatical WordStar users sighed in agreement.

From there, it was a long battle back from the cellar. MicroPro International disappeared in a flurry of corporate reorganization to be replaced with WordStar International. Upgrade has followed upgrade, with an unbroken stream of improvements and enhancements, and sales got better and better. Last year, WordStar International shipped 750,000+ units — officially placing it Number Two. They plan to become Number One again in this year.

Changes and Improvements to WordStar

What follows is an overview of changes and improvements available in WordStar with its current incarnation.

Best Things Remain Unchanged

The best thing about WordStar remains unchanged: the program doesn't get in your way as you write. When you run it, the OPENING MENU appears — pick a writing mode or companion program to use. When you work on a document, you have an uncluttered screen. On-screen labels display function key macros — so keyboard templates aren't necessary. If WordStar doesn't do what you want, then customize endlessly and make it work exactly like you need it.

Two Different Kinds of Command Menus

Choose from two different sets of WordStar command menus. For old-hands,



Figure 1. Classic Opening Menu.

WordStar still has its classic menus. (Figure 1) Call them with familiar commands — they look and work the same way. The only difference is they have more features with every upgrade.

Bowing to popular pressure, pull-down menus were added with release 5.0 of WordStar. They seem to pop-up or pull-down from the top of the screen. (Figure 2) Many new users prefer these menus to the classic ones. They access the same WordStar commands, but arrange features in a different logical order. To be truthful, they do seem to work better if using a mouse with WordStar.

Third-party mouse drivers have been available for WordStar since release 5.0 came out, and using a mouse adds new dimensions to the program.

What's best about two sets of menus is

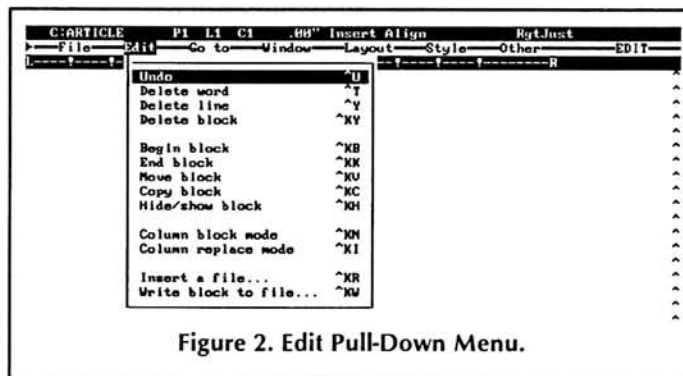


Figure 2. Edit Pull-Down Menu.

you can use whichever set you like best. Toggle between them by changing the Help Display Level. Or make either set the default by customizing with WSCHANGE.

The Star Exchange Conversion Utility

Included with WordStar 6.0 is an improved version of the Star Exchange program. This file conversion utility reads text files from major word processors, then converts them to WordStar files.

This conversion utility supports WordStar's paragraph styles, and automatically recognizes the format of the document to convert. This utility also converts WordStar text files to the formats of popular word processing programs. These include: Word Perfect, Microsoft Word, MultiMate, DisplayWrite, etc.

The Most Powerful Word Processor Anywhere

WordStar 6.0 has more usable word processing features, commands, and options — with more open access for customizing them — than its competitors. Where other word processors leave off, WordStar 6.0 picks up and keeps on running. For writing all documents basic to business and life, there's nothing that can beat it. Okay... I know I keep saying it... but it's true. Read on, and you'll see what I mean.

Flexible Page Layout

Take full control over margins, page sizing, portrait or landscape printing, kerning, etc. (Figure 3) Select between Left, Right, Center, and Full Justification, and WordStar 6.0 automatically reformats the paragraph. Layout standard or "newspaper" snake columns, in vertical or horizontal orientation — with special printing for left and right pages. Even setup for different standard sizes of paper stock.

Automatic Realignment and Text Flow

WordStar now has auto-align, automatically reformatting a paragraph after you make changes to it. This saves time once spent moving the

cursor and issuing reformat commands. When using "newspaper" column format, text formats to the columns as you type it.

Snap-to-grid makes simpler the insertion and use of graphics. When active, text automatically reformats around inserted graphics — even when you reposition them. Move any graphic object and text automatically wraps around it.

WYSIWYG Display With Page Preview

WordStar's Advanced Page Preview gives true WYSIWYG display of documents before printing. Different font types or sizes appear as they print. Change margins, page offset, line spacing or line height, number of columns, or whatever, and Page Preview delivers. Never print a document again without knowing what it is going to

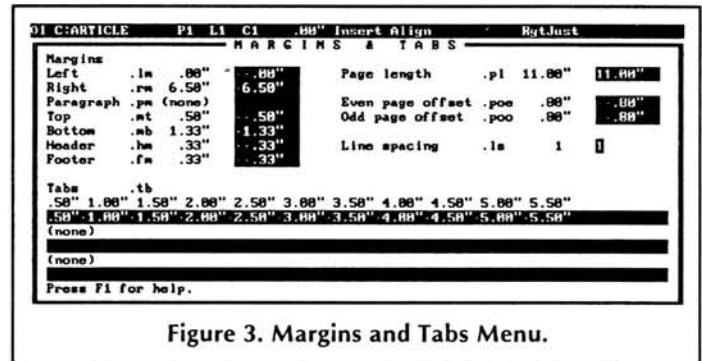


Figure 3. Margins and Tabs Menu.

look like.

Choose between a variety of page views: entire page, facing pages, 2X or 4X enlargement (for fine control over kerning, or touching up graphics), multiple pages, or thumbnail display.

Use Page Preview as an editing and proofreading tool. Use it to find specific pages. With multiple page or thumbnail displays, view a document six or twenty-four pages on-screen at a time. Locate any page visually and then go directly to it for writing and editing.

Stylesheet and Paragraph Tag Customization

WordStar's easy-to-use-and-edit paragraph tagging lets you design your own paragraph styles. (Figure 4) You can also attach a stylesheet to each directory of files, and customize its tags for specific kinds of documents. When you boot WordStar from a directory, all documents automatically have that stylesheet attached.

Tagging a paragraph is as simple as 1-2-3. Put the cursor at the beginning of the paragraph, call the style tags menu, and make your selection. Once tagged, the paragraph uses its preset layout. The special formatting and layout assigned to the tag remains in effect until counteracted by another tag.

Use style tags that come with WordStar, or customize your own. It's a snap to customize style sheets and paragraph tagging to meet your specific needs. Call the EDIT PARAGRAPH STYLE menu. (Figure 5) Move from item to item on it, setting values as you want them.

First Rate Desktop Publishing

Professional layout, design and formatting features, and image-bright WYSIWYG clarity of Advanced Page Preview, give WordStar desktop publishing matched only by program like Ventura Publisher. Typeset your documents as you go, or after you write them. Because it's an industry

documents, tables, and graphs. You do need a printer with the extended character set to use this.

Graphics Importing

WordStar 6.0 can use graphics imported from all popular graphics generation programs, including: Lotus Draw, GEM Draw, GEM Graph, GEM Scan, PC Paintbrush, and more. Cut and Paste images where you need them, then Edit, Scale, Crop, and Touch-up until they're perfect.

Insert graphics using Inset, a resident-memory program that controls printing text and graphics on the same page. "Capture" an image on-screen, store it in a .pix file, then insert a [PIX] tag in the document where the graphic image belongs.

Import grey-scale TIFF files directly; also graphics created with Splash! This makes WordStar 6.0 an even more industry-standard publishing tool.

WordStar 6.0 is compatible with most graphics generators. You can dip into a growing supply of Graphic Libraries on the market. This makes it easy to find clip art you need to make your documents look exactly like you want them.

Improved Laser Printer Support

WordStar 6.0 has special support for LaserJet III features & printing quality. It also supports PostScript printing, the Canon LBP III, Canon LBP-4, the HP DeskJet, and most other laser printers currently marketed. WordStar International intends to dominate in word processing technology for laser printers.

Extended Print-to-Disk Capability has been added. Non-PostScript laser printers are now included in the print-to-disk option. This allows you to print over a network, where printing to disk is part of the process.

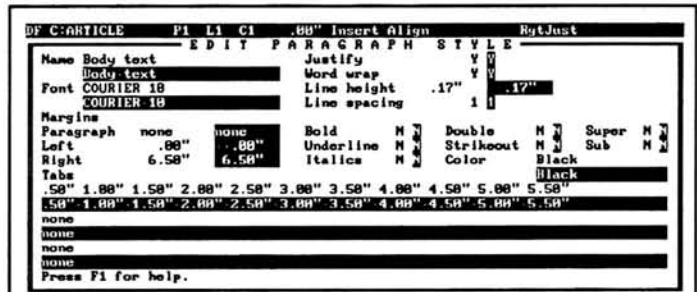


Figure 5. Edit Paragraph Style Menu.

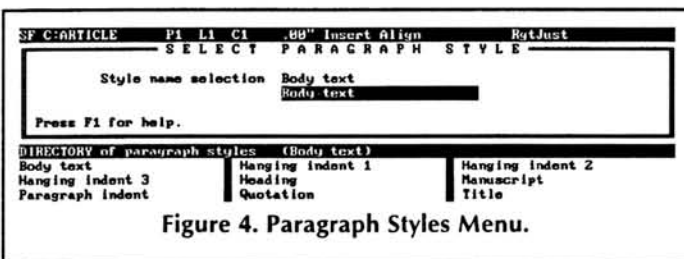


Figure 4. Paragraph Styles Menu.

standard, WordStar documents can go to your print shop on computer disk for rapid setup and professional printing.

Borders and Boxes

With WordStar's built-in line drawing capability you can easily place borders and boxes around and/or between paragraphs, pages, or columns — to spice up ordinary

precise control over space between letters in a word, and over the entire printed line. This improves both appearance and legibility of headlines and body text. It's especially important when working with proportionally spaced fonts.

When using monospace fonts, space between words does not stretch, but remains the same width as other characters

in the font family.

Expanded Type Selection

WordStar 6.0 directly supports four multiple symbol sets: PC-8, 850, Math 8 (including Ventura Math and PostScript Math), and Symbols. Insert characters from these symbol sets while writing documents, and view them with Page Preview.

WordStar 6.0 adds more fonts to all printer drivers. Use down-loadable fonts, or custom install for font cartridges (if your printer uses them), then scroll the list of available typefaces and point sizes. This opens for use an unlimited number of Bitstream and Typografica typefaces.

Besides the standard adobe 35 set WordStar 6.0 recognizes PostScript fonts available from sources like Bitstream. You can import encapsulated PostScript (EPS) files and rotate them in 90 degree increments. This means any of these fonts can be used as either Portrait or Landscape fonts.

Scalable Font Support

WordStar 6.0 is a great word processor to use with scalable fonts. Select any available typeface and specify a point size

from 2 to 999 points (depending on the printer's limitations). Automatic adjustments are made for line length and spacing.

Two main packages of fonts are available directly from WordStar International. These are Bitstream and Type Director 2. With Bitstream, you get a package of 16 basic fonts and the software to build them on your hard drive.

Build them in as many sizes as you wish. The only limitation is the hard drive space available. These fonts consume much storage space, but are more flexible than fonts on cartridges. With cartridges, what you get is what you get. There's no customization possible. If you want bigger or smaller fonts, you must buy another cartridge.

Special Effects Printing

Assign font patterns and grey scaling to any of 16 colors or patterns in WordStar. Use this to create special effects beyond boldface and Italic attributes. Of course, the quality of special effects printing de-

depends on printer quality. You need a color printer for color; and a 24-pin or laser printer for fine resolution. WordStar 6.0 has excellent support for HP LaserJet and PostScript printers.

Hewlett-Packard Fonts and Cartridges

WordStar 6.0 automatically supports internal fonts in the HP LaserJet III. It also supports scalable font cartridges from ProCollection. Add other scalable or bit-mapped fonts to your laser printer with Type Director 2. This package of fonts can create fonts in any size from 4-point to 200-point.

WordStar 6.0 supports HP LaserJet Soft Fonts available from many sources. It works perfectly with almost any font cartridge available for your laser printer.

A Final Word

I've written many words about what WordStar can do for you as a writer. While I have a genuine enthusiasm for this program, I'm also pragmatic enough to realize it may not be right for everyone. Different people always have different likes and dislikes about how computer technology presents itself to them. This is why so many word processors are still on sale and doing well.

Many writers, and business offices, still insist on using typewriters for their daily work. I can understand this, because typewriters held a magical allure for more than half my writing life. I used to dream of better and better typewriters, while pecking away at my Smith-Corona electric. Whenever I used an IBM Selectric, I "oohed and ahhed" over its silky typing smoothness. No matter what machine I used, demons of clumsy foul-ups, awkward fingers, messy correction with erasers and white out still haunted me. Rewrites still meant retyping whole manuscripts.

Word processing eliminates all this in a single stroke. Typing speed goes up as "mistake anxiety" fades. Just delete and retype those typos. Do spelling and grammar checks with the computer. Reorganize and rewrite easier with Cut and Paste. You never have to worry about getting it perfect the first time. Generate each new draft of a manuscript from the preceding one.

As a WordStar "fan" I promote what has done me the most good as a writer. As a writer, first-and-last, my chief support is for word processing itself. I have had the benefit, or misfortune, to learn eight different word processing programs since 1982. I have worked on everything from Apple computers, through Kaypro II's and PC clones, on Wang dedicated machines to IBM Mainframes. The easiest to use were Apple II and Macintosh. By far, the clunkiest system was the mainframe. Yet even the worst systems, with all their quirks and major annoyances, made possible better

and faster writing than I ever could with my old electric typewriter.

I've come to a hard and fast position: writers who don't use computer word processing aren't serious about their craft. This includes every level and kind of writer. I've always kept hand-written journals, first as a learning tool, and now for disciplining my mind. (Writing in longhand requires thinking differently about what one writes.) But the typewriter is a dinosaur. For professional writing, I never consider anything except sitting down to my computer keyboard. It is the only way to write well.

Find and use the computer and word processor that please you the most. Satisfaction with the day-to-day labor of writing is as important to good writing as any other factor. It doesn't matter whether you write business letters eight hours a day, or chip away at your Great American Novel — if you're unhappy with your writing tools, work quality suffers. If your word processor is a pain in the neck, ask for a better one. Get a better one, even if you have to buy it yourself.

You also have a right to expect competent training. Writing with a computer should make writing easier. Used properly, it can lighten your load and speed you on your way. Used poorly, it slows you down and buries you with anxiety. If working in an office without sufficient training, you have a right to ask for better. If your position is strong enough, then demand it. But if neither gets results, find outside training and learn how to get the most out of your word processor.

Unfortunately, good classes on word processing and desktop publishing seem inevitably expensive or unavailable. I've heard all the horror stories. The principle

seems to be caveat emptor — "let the buyer beware." Too many courses on using personal computers are run by engineers or programmers moonlighting for a few extra bucks. These folks are not known for their communication skills. Too often, their students sit through dull-as-dish-water recitations of facts going right over their heads. Maybe they get a few hours hands-on instruction, and then out into the cold.

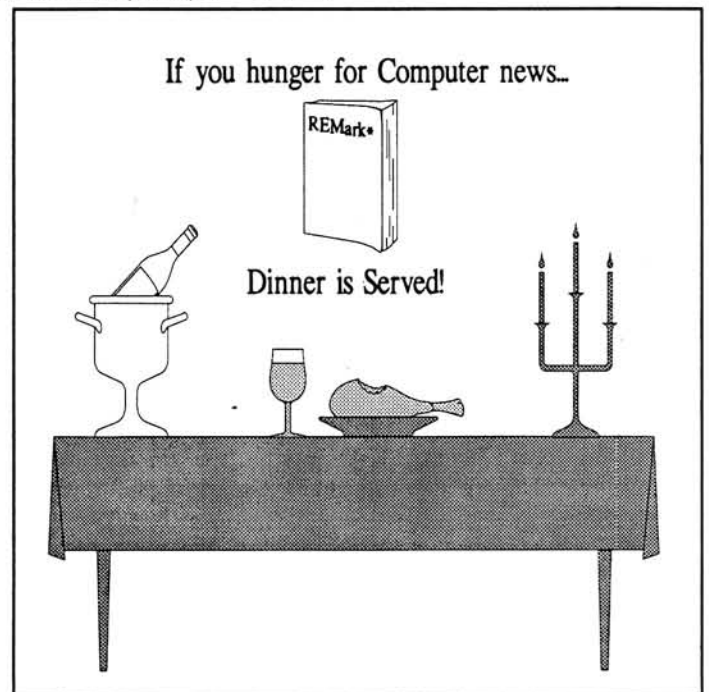
Before you invest in any training course, you better shop around. Make

sure a school is reputable. Make sure your "instructor" has some credentials for teaching, and has written extensively using the word processor in question. It's your money, and your time — and a bad course of instruction is worse than nothing.

In spite of everything, most people still learn word processing all by themselves — with nothing but a badly-written users manual as a guide. Self-taught users invariably waste time and disk space with bad habits and needless redundancy. Most writers don't know what questions to ask. They begin with misconceptions and end by learning only enough to "get by" from day to day.

Despite this, the best learning with computers comes from lots of hands-on experience. The more you use it, the more comfortable and fluent you become. You need to get the broadest overview of word processing as possible. Go to the bookstore and find a manual that speaks to you as you need to hear it. Learn what questions you need to ask. Discover what major features can really do — and learn to apply them to your writing. Don't be afraid to use "trial and error" to pick up any useful feature. Whatever you do, don't DROWN yourself by trying to learn everything all at once. The number one cause of confusion and errors is writers getting ahead of themselves.

Remember: Word processing is a tool, not a cure all. As a better hammer drives nails truer, so your word processor can be used to drive your words home. This hammer still takes a skilled hand to do the work, and lots of practice to strengthen the arm. Execution makes the difference between merely driving nails, and building a cathedral. ✽





When HEPCAT™ sings...

...the other cats get to sing along! And now, HEPCAT does Windows!



HEPCAT, the powerful, versatile pop-up calculator from ZUC, 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 ZUC 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. ✱

A decorative border of stylized yellow and orange flames surrounds the central text.

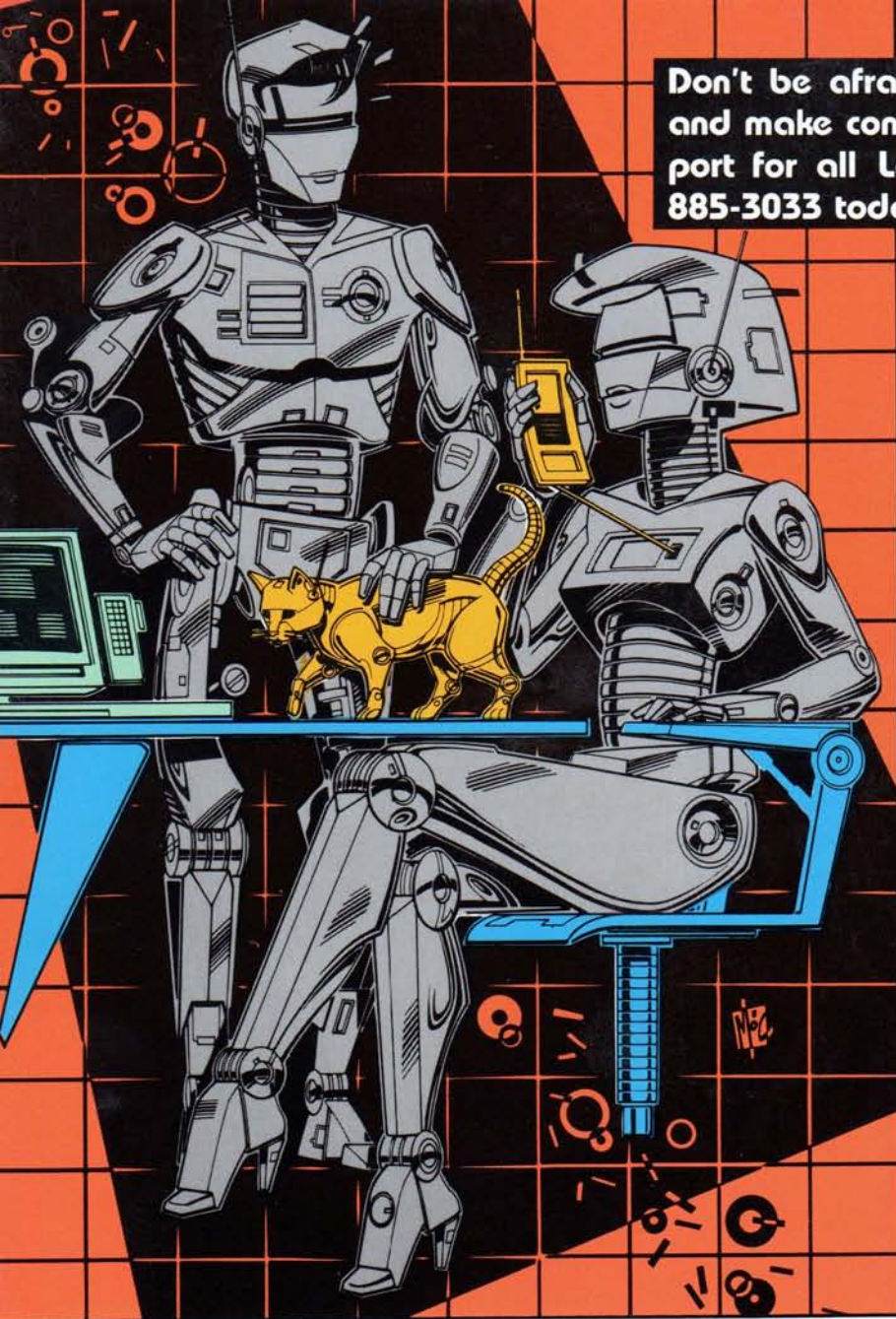
HADES II

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

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

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

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



```
HUGMCP Commands
F1 -- Prints This List, Your Storage Buffer Size, And How Many
     Bytes Are Presently In The Storage Buffer.
F2 -- Allows Sending A Defined Message, Or Character Sequence.
     These Messages Are Entered Using The (F6) Setup Command.
F3 -- Toggles The Storage Buffer On and Off. When The Buffer
     Is On, The (Ctrl) 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 XMODEM,
     Which Optionally Can Be Ignored, Or XMODEM Protocol.
F6 -- Enters The Setup Mode. So This Software Can Be Configured.
F7 -- Clears Out Any Data That May Be In The Storage Buffer.
F8 -- Send Data In Storage Buffer To Printer.
F9 -- Exits Back To MS-DOS.

Storage Buffer = 524288 Bytes
Storage Buffer Usage = 0 Bytes

Select Message (0-0), (F1) To List, Anything Else To Abort --) _
F1=Hly F2=Msg F3=BufR F4=Save F5=Send F6=Cgr F7=Cle F8=Print F9=Exit CM
```

```
HUGMCP Configuration Menu #1
1 This Function Allow The Baud Rate To Be Changed. Depending Upon Which
  Mode You Are Using, It Will Be Set To Either 300, 1200, Or
  2400 Baud. Select Option 1 To A Word. All Will Allow Your Baud Rate.
2 This Function Allow You To Change The Word Parity. Normally, you
  Should Have "No Parity". This Is Acceptable By Most Remote Systems.
  And It Is Also Necessary For XMODEM Protocol To Work Properly.
3 This Function Allow The Changing Of The Word Length. Normally The
  Length Should Be Set To 8 Data Bits. This Value Is Acceptable By Most
  Remote Systems. And Is Necessary For XMODEM Protocol To Work Properly.
4 This Selection Allow You To Enter Messages Which Can Be Automatically
  Sent With The (F1) Key. Up To 14, 70-Character Messages Can Be Entered.
  Selection #1 Is Suggested. If Symbols Contain Your Computer's Name,
  And The Response Selection (0) Is Also Suggested. This Selection Can Auto-
  matically Be Sent When This Program Is First Executed By Initializing The
  Program Option During Setup.

The (F1) Key Is For Help Only. Anything Else To Quit --)
F1=Hly F2=Msg F3=BufR F4=Save F5=Send F6=Cgr F7=Cle F8=Print F9=Exit CM
```

```
HUGMCP Configuration Menu:
0 --)
1 --) Modify Baud Rate
2 --) Modify Parity Type
3 --) Modify Word Length
4 --) Modify Or Add Auto-Messages
5 --) Miscellaneous Functions
6 --) Change Screen Color Assignments
7 --) Display Current Configuration
8 --) Make Changes Permanent

Select 0-C, (F1) For Help, Anything Else To Quit --) _

Baud Rate: 19200
Parity: NONE
Word Length: 8
Duplex: Full
Response In Keyboard Disable: NO
Storage Buffer Data Parity Bit: SET TO ZERO
Send Modem Initialization Text: NO
Delete Character: W0000
Modem Port Set To: COM1

F1=Hly F2=Msg F3=BufR F4=Save F5=Send F6=Cgr F7=Cle F8=Print F9=Exit CM
```

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 885-2143