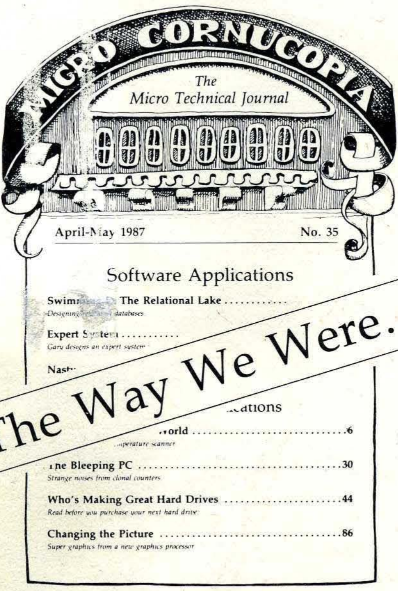
No. 36 June/July 1987 $3.95

THE MICRO TECHNICAL JOURNAL

MICRO CORNUCOPIA


MICRO CORNUCOPIA

The Micro Technical Journal

April-May 1987 No. 35

Software Applications

Swim[illegible] The Relational Lake ....
Designing [illegible] databases

Expert System ....
Gary designs an expert system

Nast[illegible]

...cations

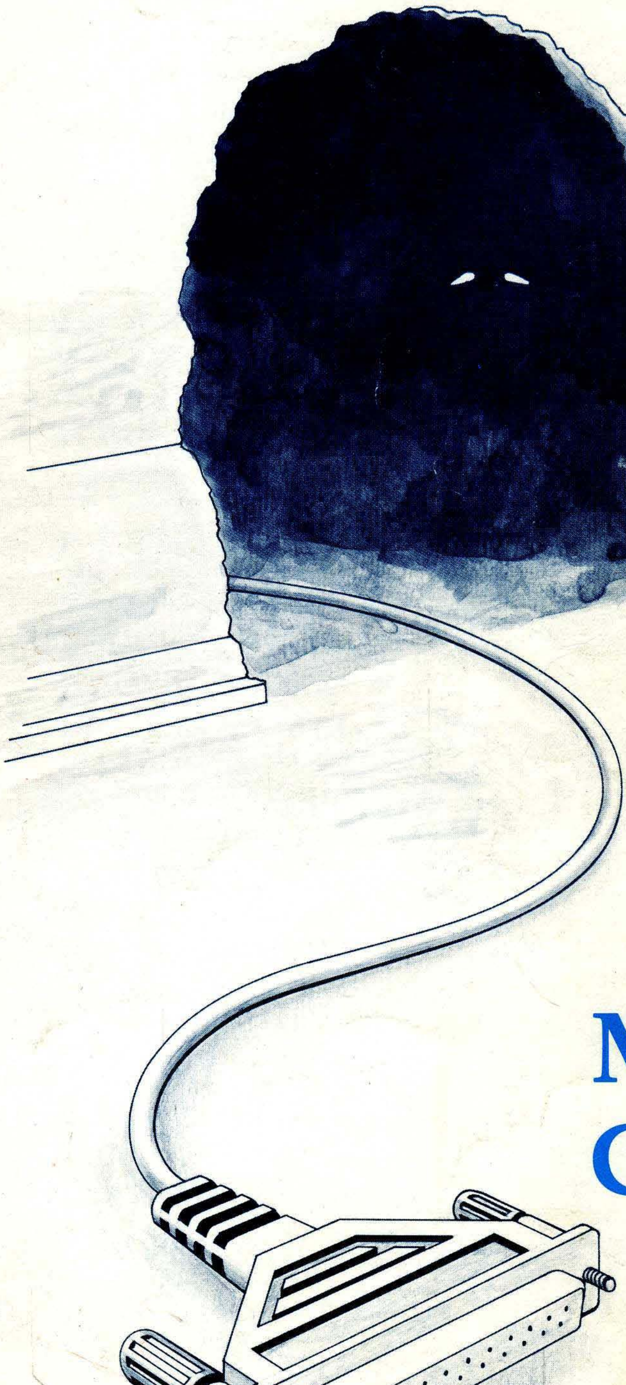
...world 6
...perature scanner

The Bleeping PC 30
Strange noises from clonal counters

Who's Making Great Hard Drives 44
Read before you purchase your next hard drive

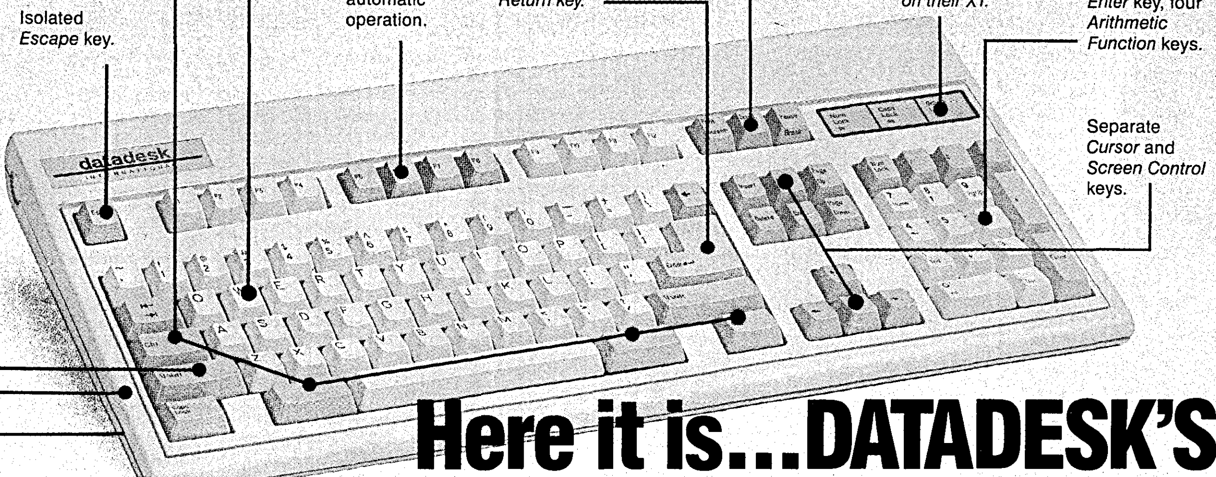
Changing the Picture 86
Super graphics from a new graphics processor

The Way We Were...



Mouse Control

Borland's Turbo Lightning™ FREE

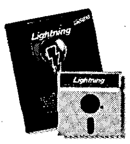

Enlarged *Shift* keys
Two *Ctrl* and *Alt* keys for ambidextrous access.
Positive-tactile, firm-feel keys.
12 *Function* keys for increased automatic operation.
Extra-wide, easy-to-find *Return* key. *IBM changed sizes again by shrinking their Return key.*
Dedicated *Function Control* keys.
Caps, *Num* and *Scroll* Lock indicator lights. *IBM has no lights on their XT.*
Separate dedicated *Numeric Pad* with enlarged *Enter* key, four *Arithmetic Function* keys.
Isolated *Escape* key.
Switch-selectable compatibility with IBM, PC, XT, AT, PCjr, AT&T and all compatibles. *IBM's new Enhanced Keyboard runs only on their new XT's, AT's and ignores their installed base which probably includes you.*
Separate *Cursor* and *Screen Control* keys.
Switch allows you to swap position of *Ctrl* and *Caps* lock. *IBM moved Ctrl to bottom row, you have no choice!*
datadesk

Here it is...DATADESK'S *TURBO-101 ENHANCED KEYBOARD* for the 10 million PC users IBM just ignored!

IBM just announced their new redesigned "standard" keyboard for personal computers. There's only one problem: *it won't work* on your IBM computer if it was purchased prior to June 1986 or on *any* PC compatible purchased at any time!

Not to worry. Our new *Turbo-101* Enhanced Keyboard gives you the layout and enhancements of the IBM with some logical improvements (see above photo). And it works on your existing PC, XT, AT, PCjr, AT&T, Epson and virtually all compatibles!

Get Borland's Turbo Lightning™ For FREE!



To really turbocharge your productivity, we are including, free-of-charge, Borland's red-hot Turbo Lightning software with each keyboard. Now, when using SideKick, WordPerfect, Microsoft Word, 1-2-3 or most popular programs, our Turbo-101 Keyboard will check your spelling *as you type,* gives you instant access to Random House's 80,000-word Concise Dictionary and 60,000-word Thesaurus and much, much more!

"Lightning's good enough to make programmers and users cheer, executives of other software companies weep," **says Jim Seymour of PC Week. Sold separately, Turbo Lightning retails for $99.95!**

The Turbo-101 is the best data entry tool since the pencil!
For users of spreadsheets like 1-2-3, the Turbo-101's *separate* cursor controls and numeric keypad makes entering numeric data into cells and moving from cell to cell as natural as moving your fingers. And for *word-processing,* the *'Selectric'* typewriter layout makes the Turbo-101 as easy to use as a pencil; and with the extra large Enter, Shift & Control Keys, you'll make so few mistakes, you won't even need an eraser!

SPECIAL OFFER!

*ONLY $149.95** FOR BOTH KEYBOARD & SOFTWARE

Includes 30-day money back guarantee and 2 year full warranty. To prove that we don't ignore you or your pocketbook, you get our Turbo-101 Enhanced Keyboard and Borland's Turbo Lightning for an astounding $149.95.* No, you didn't read it wrong. During this amazing ***Introductory Offer*** you get both keyboard and software for less than most software programs by themselves! Now, if you're still feeling ignored, you can always do what you-know-who wants you to do...*and buy a new computer to get their keyboard!*

Up to now, DataDesk International may be one of the best kept secrets, but here's what's being said about our first *end-user* Keyboard/Borland software bundle:

"Who Can Pass Up a Deal? Department. Talk about an aggressive product!"
John C. Dvorak, InfoWorld Mar 86

"It solves all of the problems exhibited by their regular PC/XT keyboard...it's a great bargain!"
PC Productivity Digest May 86

"DataDesk Intl. has designed a sturdy and handsome keyboard that has tactile response...is the hardware bargain of the year" says Charles Humble, Oregonian Jan 86

"The best part of the keyboard is the way it feels. It's ideal! And fast. I've never worked on a keyboard with a nicer touch."
Business Computer Digest Aug 86

"It's a good keyboard. Good feel: the keys have tactile feedback. No mush at all. This is about as good a keyboard deal as you're likely to find...I have absolutely no hesitation in recommending the Model PC8700."
Jerry Pournelle, Byte Magazine Sept. 86

"This keyboard is neat to type on and feels solid. It has tactile feedback keys...I can type much faster on it."
Test Drive Scorecard: DataDesk-10 Key Tronics-9
Teleconnect Magazine May 86

The Only Alternative
datadesk™
INTERNATIONAL
7650 Haskell Avenue
Van Nuys, California 91406 (818) 780-1673

Turbo-101 is a trademark of DataDesk International. Turbo Lightning is a trademark of Borland International. IBM and IBM AT are registered trademarks of International Business Machines, Inc.

credit card orders call
(800) 826-5398
in CA call
(800) 592-9602

*Price does not include adaptor cables required by certain compatibles • A Limited offer—price subject to change without notification.

MADE IN USA

BOTH TURBO-101 ENHANCED™ KEYBOARD AND BORLAND'S TURBO LIGHTNING™ SOFTWARE FOR ONLY:

$149 95* LIMITED OFFER
*Plus $10 Shipping & Handling. California Residents Add $9.75 Per Unit Sales Tax

NO. UNITS: ______
PAYMENT: ☐VISA ☐MC ☐CHECK
AMOUNT ENCLOSED: $ ______

Enter

NAME: ______
ADDRESS: ______ STATE: ______ ZIP: ______
CITY: ______ COMPUTER TYPE: ______
PHONE: ______ EXP: ______
CC NO.: ______

HITACHI O'SCOPE SALE

SAVE $75 to $450

MANY MODELS TO CHOOSE FROM. PLEASE CALL! 3 YEAR WARRANTY!

—IC PROGRAMMERS—

XT/AT COMPATIBLE

ALL USE SINGLE SLOT IN YOUR XT/AT AND INCLUDE EXTERNAL PODS WITH ZERO INSERTION FORCE SOCKET(S)

PAL PROGRAMMER
$299.95

—PROGRAMS MOST 20 & 24 PIN PALS
—VERIFY, PROGRAM, & BURN SECURITY FUSE LINK

8748/49 PROGRAMMER
$199.85

—EPROM PROGRAMMERS—

PROGRAMS 27XX to 27512
12.5, 21, 25V DEVICES
INCLUDES SOFTWARE FOR STANDARD & INTEL HEX FORMATS

SINGLE GANG . . . $109.95
FOUR GANG $169.95
TEN GANG $339.95

HALTED SPECIALITIES offers unique supermarket style shopping for your electronic needs. We stock thousands of parts, from the newest IC's to some of the first transistors. We also have computer accessories, test equipment, tools, R&D supplies and much more. Please call or visit one of our retail stores.

—FLOPPY DRIVES—

ALL DRIVES NEW WITH 6 MO. TO 1 YR. WARRANTY

TOSHIBA 3.5 . . . . . $139.00

—720K FORMATTED
—DOS 3.2 COMPATIBLE OR USE OUR HIGH DENISTY CONTROLLER IN YOUR XT
—INCLUDES UNIVERSAL 5¼" MOUNTING KIT FOR XT/AT AND COMPATIBLES

TOSHIBA 1.2M . . . $139.95

—1.2M HIGH DENSITY
—AT COMPATIBLE
—USE ONE IN YOUR XT WITH OUR HIGH DENSITY CONTROLLER

FUJITSU 360K $89.95

—½ HEIGHT
—IBM COMPATIBLE

TEAC 360K $109.95

—½ HEIGHT
—IBM COMPATIBLE

SHUGART 8" SA801R
$149.95

—SINGLE SIDED DOUBLE DENSITY
—600 KB FORMATTED CAPACITY

—HARD DRIVES—

SEAGATE ST225 . $299.95

—20 MEG
—½ HEIGHT

MMI 212 . . . . $219.95

—10 MEG
—3.5" SHOCK MOUNTED IN 5¼" ½ HEIGHT FRAME. LOW POWER. GREAT FOR PORTABLES

SEAGATE ST4038
$599.95

—30 MEG
—AT COMPATIBLE
—39 MS ACCESS

CONDOR SWITCHER
$35.00

+ 5V @ 9.0 AMPS
+12V @ 3.0 AMPS
5.0 AMPS PEAK
-12V @ 1.0 AMP
+15V @ 1.5 AMP

—FLOPPY CONTROLLERS—

NEW!
FLOPPYMASTER . . $89.95

—XT COMPATIBLE
—CONTROLS 1.2M, 720K, 360K DRIVES BOTH 3½ AND 5¼ INCH

DUAL FLOPPY CONTROLLER
$19.95

—IBM COMPATIBLE
—48 & 96 TPI DRIVES
—DOES NOT INCLUDE CABLES & BRACKET

—HARD DISK CONTROLLERS—

XEBEC 1210A $79.95

—FULL LENGTH
—CONTROLS 10 & 20 MB
—CUSTOM CONFIGURATIONS AVAILABLE

OMTI 5520 . . . . $124.95

—WITH CABLES
—SHORT SLOT CARD
—CONFIGURABLE BIOS
—RUNS ANY 3" OR 5" HARD DISK
—SOFTWARE
—SUPPORTS DRIVES UP TO 320 MB!!

OMTI 5527 . . . . $169.95

—RLL CONTROLLER
—WITH CABLES & SOFTWARE

KONAN KDC 230 . . . $89.95

—SHORT SLOT CARD
—LSI SURFACE MOUNT TECHNOLOGY
—CONTROLS MOST 5" HARD DISKS
—MENU DRIVEN CONFIGURATION

—FLOPPY DISKETTES—

STANDARD 5¼" DSDDBOX OF 10 . $4.50
SHUGART DSDD 8" BOX OF 10 . . $12.50
VERBAT!M SSSD 8" BOX OF 10 . . . $4.95

—SASI HOST ADAPTER—

PC/XT COMPATIBLE
$14.95

SYNC-SEPARATOR KIT

USE WITH ANY 3 LINE TTL MONITOR ALLOWS YOU TO RUN 3 LINE MONITOR FROM A COMPOSITE VIDEO SOURCE!
$12.95

HALTED SPECIALTIES CO., INC.

827 E. EVELYN AVE., SUNNYVALE, CA 94086

MAIL ORDERS CALL: (408) 732-1573

3 CONVENIENT LOCATIONS

HSC Electronic Supply of Santa Rosa
6819 S. Santa Rosa Ave.
Cotati, CA
(707) 792-2357

HSC Electronic Supply
5549 Hemlock Street
Sacramento, CA
(916) 338-2545

Store Hours:
Mon.-Fri. 8:00-7:00
Saturday 9:00-5:00

WE SHIP C.O.D.

MasterCard VISA AMERICAN EXPRESS

TERMS: Minimum order $10.00. California residents add 7% sales tax. Prepaid orders sent freight C.O.D. or call for charges, shipping will be added to credit card and C.O.D. orders. Prepaid orders over $100.00 use money order or certified check. Please do not send cash. Some items limited to stock on hand. Prices subject to change.

MICRO CORNUCOPIA

The Micro Technical Journal

Editor & Publisher
David J. Thompson

Associate Editors
Gary Entsminger
Cary Gatton
Rebecca Ozrelic

Technical Department
Larry Fogg

Director of Advertising
Laura Logan

Director of Marketing and Graphics
Julie Hastings

Circulation Manager
Tracey Anthony

Accounting
Sandra Thompson

Order Department
Tammy Westfall

MICRO CORNUCOPIA (ISSN 0747-587X) is published bi-monthly for $18 per year by Micro Cornucopia Inc. 155 NW Hawthorne, Bend, OR 97701. Second-class postage paid at Bend, OR and additional mailing offices. POSTMASTER: Send address changes to MICRO CORNUCOPIA, PO Box 223, Bend, OR 97709.

SUBSCRIPTION RATES:

1 yr. (6 issues)	$18.00
2 yr. (12 issues)	$34.00
3 yr. (18 issues)	$48.00
1 yr. (Canada & Mexico)	$22.00
1 yr. (Other foreign)	$30.00

Make all orders payable in U.S. funds on a U.S. bank, please.

CHANGE OF ADDRESS: Please send your old label and new address.

MICRO CORNUCOPIA
P.O. Box 223
Bend, Oregon 97709

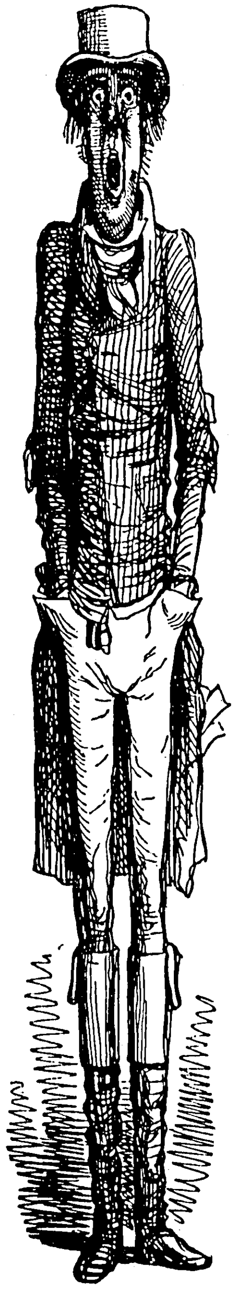
CUSTOMER SERVICE: for orders & subscription problems call 503-382-5060, 9 am to 5 pm, Pacific time, M - F.

For technical help call 503-382-8048, 9 am to noon, Pacific time, M - F.

RBBS - 24 hrs. 300-1200-2400 baud
8 Bits, No Parity, 1 Stop Bit
503-382-7643

AROUND THE BEND

Don't Panic



Publishing a magazine is a lot like reading Douglas Adams (he wrote *The Hitchhiker's Guide To The Galaxy*), especially since we picked up a secondhand model B-13 Infinite Improbability Drive (IID).

It was a very unlikely event, finding the model B-13. Even more unlikely was finding one we could afford. And, most unlikely of all was finding a working one, since the factory has never shipped a working IID, the independent service organization has never successfully repaired a non-working one, and neither has heard of this model. All of which, of course, explains everything.

Take this issue's cover. It might be very much like last issue's. It might be very different. There might not be a cover at all. (You might not be reading this.) That's the kind of thing that an Infinite Improbability Drive brings to a publication.

If you like whatever it is you have or haven't seen, you probably won't say anything. If you don't like it, you'll no doubt send us letters describing in excruciating detail our most private shortcomings. The IID will probably ignore these letters, but we won't (it's one of our shortcomings).

A lot of you really liked our old style. You told us. But, just for once, we're doing something for ourselves. We've been secretly dreaming of putting out a real, honest to gosh: "in your eye Madison Avenue here comes Micro C" kind of publication.

Meanwhile, the content won't change. At least it won't change any more than it's been changing. (Probably.)

Doing Your Own Thing

Speaking of content, this issue is crammed with "On Your Own" information. If you're interested in writing and marketing your own software, check out the information on shareware in the "On Your Own" Column. If you're more interested in hardware, Bruce Eckel and Larry Fogg have their usual great stuff.

Working for yourself versus working for a large corporation is like free enterprise versus welfare, or piloting your own small plane versus riding in an airliner. It's the freedom of making your own decisions, the status of entrepreneurship, and the financial benefits of winning the lottery all rolled up into one. If you're lucky.

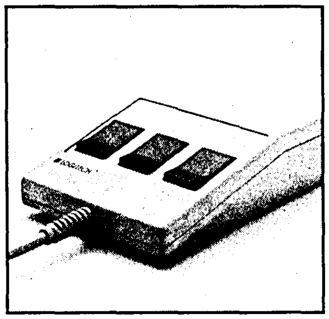
Overall, the odds of starting a successful business are about one in ten. If you pick your product carefully and do enough of a business plan to know that it can be sold successfully and profitably, then the odds improve to one in two. That's much better than any lottery.

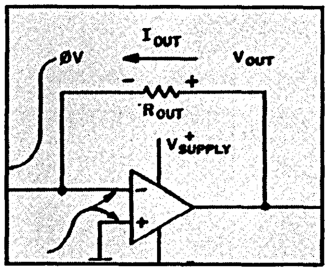
(continued on page 90)

June-July 1987
Issue No. 36

CONTENTS

Features





PD32 Status Report

In October, 1986, the "finished" PD32 prototype was handed over to Definicon Systems (DSI) by designers George Scolaro and Dave Rand. DSI was to manufacture and market the kit for a small markup over the cost of parts. At this point, we had a two-layer board with half a dozen cuts and jumpers. One "tidy-up" revision was suggested and the project was ready to go.

DSI had ten boards assembled. Not one of them worked. Swapping chips and hanging bypass caps everywhere brought a few boards to life, but obviously the "finished product" was far from finished. We decided that a four-layer board was required to solve the problems.

The board went into layout (back to step one), and prototypes were assembled (wait two weeks). Now things were looking a lot better -- all the four-layer prototypes worked on power up. A production run of the board began (wait five weeks). To avoid future headaches, we decided to offer only wave soldered kits. The kit parts and blank boards were shipped to an assembly house (wait ten days) and now they're READY. Honest.

The die-hard hacker who wants to solder his own board can still buy a blank. But please don't send it back to us if it doesn't work. Blank boards sell for $50 plus shipping. The kits go for $370 to $695 depending on speed (6 or 10 MHz) and RAM (1 or 2 MBytes). Get the $500 UNIX from DSI or directly from Dave Rand in Alberta.

Steve Hope, Senior Engineer
Definicon Systems
21042 Vintage St
Chatsworth, CA 91311
(818) 889-1646

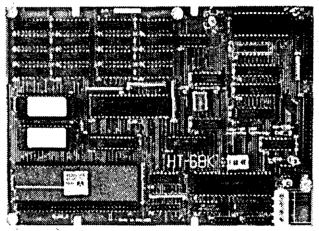
Seagate Poo-Poo

I, too, have noticed that recent Seagate drives have been poo-poo. I have purchased several from dealers in the Washington, D.C., area. All sounded like Woody Woodpecker ("whack, whack, whack") when trying to home the heads.

I contacted Western Digital. After a lot of ranting and raving, a real tech came on the line. He said that controller cards should go out with the W6 jumper on pins 2 and 3. This sets the controller for low current, driving a maximum of eight heads. Three out of three controllers I bought had pins 1 and 2 jumpered (high current, 16 heads). Apparently whole bunches of controllers were shipped that way.

Suspecting other problems, I called Seagate on my nickel. It's sad that when you spend your money to call for service, you get to hear a long recording advertising their 800 sales line.

68000 SINGLE BOARD COMPUTER
$395.00
32 bit Features / 8 bit Price



-Hardware features:
* 8MHZ 68000 CPU
* 1770 Floppy Controller
* 2 Serial Ports (68681)
* General Purpose Timer
* Centronics Printer Port
* 128K RAM (expandable to 512K on board.)
* Expansion Bus
* 5.75 x 8.0 Inches Mounts to Side of Drive
* +5v 2A, +12 for RS-232
* Power Connector same as disk drive

Add a terminal, disk drive and power, and you will have a powerful 68000 system.

-Software Included:
* K-OS ONE, the 68000 Operating System (source code included)
* Command Processor (w/source)
* Data and File Compatible with MS-DOS
* A 68000 Assembler
* An HTPL Compiler
* A Line Editor

ASSEMBLED AND TESTED ONLY **$395.00**

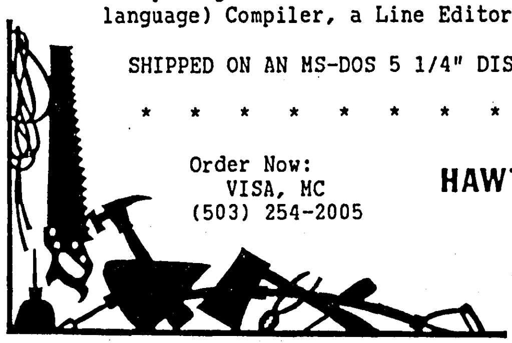
* * * * * * * * * * * * * * * * *

K-OS ONE, 68000 OPERATING SYSTEM

For your existing 68000 hardware, you can get the K-OS ONE Operating System package for only $50.00. K-OS ONE is a powerful, pliable, single user operating system with source code provided for operating system and command processor. It allows you to read and write MS-DOS format diskettes with your 68000 system. The package also contains an Assembler, an HTPL (high level language) Compiler, a Line Editor and manual.

SHIPPED ON AN MS-DOS 5 1/4" DISK. **$50.00**

* * * * * * * * * * * * * * *



Order Now:
VISA, MC
(503) 254-2005

HAWTHORNE TECHNOLOGY
8836 S. E. Stark
Portland, Or 97216

Anyway, after numerous attempts, I managed to talk to a real tech at tech support and he verified that the 225s have stepper problems.

This was after trying to call the company president. Naturally, I only got his secretary. The worst part was the classic "not my job" attitude. I did try to point out that there was once a drive company named CMI...

Mike Rutkoski
9909 Dameron Dr.
Silver Spring, MD 20902

Moonlit Software

My congratulations to Cecil Stump for his "On Your Own" column in issue #35. I am in a similar position with C.C. Software. It brightened my day to hear that others have gone through some of the same problems I have. His points on customer relations were especially well taken.

One point Mr. Stump didn't elaborate on was advertising. Because costs are very high (Micro C excluded of course), the startup business will not be able to advertise heavily, if at all. Ads are nice, but customers want to hear what others think about a product -- if only to find out that the company is legitimate.

So product reviews are very important. Software houses live and die by reviews. Magazines do publish reviews of various products but their space is limited. And getting their attention amid thousands of press releases can be very difficult. If your products are as unusual as mine, the chance becomes even smaller. Creating an awareness of your product will take persistence.

Clark A. Calkins
C.C. Software
1907 Alvarado Ave.
Walnut Creek, CA 94596

Z80MR Source

I would like to know if source code is available for the Z80MR macro assembler (Micro C User Disk K25). The assembler has a bug in its print routine that could be fixed if source were included.

Also, a hex dump of Z80MR shows that the commands ASEG, CSEG, EXTERN, LOCAL, NAME, and PUBLIC exist. Use of these commands produces an OBJ file instead of a HEX file. With the assembler source to show the OBJ format, a linker could be written.

Peter J. Hall
2703 Newton St.
Wheaton, MD 20902

Editor's note:

Some time back we tried (unsuccessfully) to obtain the Z80MR source. If anyone out there has done a disassembly, we'd sure like to have a copy to add to the Z80MR disk.

■■■

NEW LOWER 16 BIT PRICE

SPEED — POWER — VERSATILITY

EXPRESS 2.2

FULL SCREEN EDITOR

ONLY $29.95

Most EXPRESS users are converts from the GIANTS of WORD PROCESSING ......
......and you couldn't pay them to go back...HERE'S WHY

- FULL ACCESS TO CP/M USERS AREAS (up to 32) with any editor command!
- NAMED DIRECTORIES supported on MSDOS VERSION!
- MEMORY MAPPED VIDEO AVAILABLE FOR IBM PC and PC clones!
- BUILT-IN CP/M LIKE COMMANDS (RENAME, COPY, ERASE, TYPE, DIR, LOG)!
- POWERFUL KEYBOARD MACRO'S (as many as you want)!
- FULLY RECONFIGURABLE COMMAND KEYS (emulate any other editor if you like)!
- FAST, FAST, FAST SEARCH!
- TERMINAL DATA BASE...INSTANTLY configure for over 50 predefined terminals!
- FILES LARGER THAN MEMORY handled with ease!
- CONTROL and HIGH BIT CHARACTERS may be entered and edited!
- DYNAMIC WORD WRAP/UNWRAP • FULL CURSOR CONTROL (and then some)
- EASY to SET TAB STOPS • GLOBAL/SELECTIVE/LITERAL/IGNORE CASE REPLACE
- FULL BLOCK INSTRUCTIONS including PRINT, SAVE, INCLUDE, MOVE, COPY, DELETE
- VARIABLE SPEED (FAST!) BI-DIRECTIONAL AUTO SCROLL • GOTO PAGE N / LINE N
- COMPACT on disk and in RAM. (even the 16-bit versions are only 25k bytes)!
- NOT COPY PROTECTED!
- AFFORDABLE... High performance at a fair price is our motto
- EXPRESS 1.0 SAMPLER available FREE on your Local Bulletin Board or for $10.00 from TCI

GENTLEMEN ... I'm ready to step up to EXPRESS... please send:
______ copies EXPRESS 2.2 at $29.95 + $5.00 shipping and handling
______ copies EXPRESS 1.0 at $10.00 + $2.00 shipping and handling
My computer uses: PCDOS — MSDOS — CPM-86 — CCPM-86 — CPM-80 (Circle One)
Disk format: Kaypro II — Kaypro 2X,4,10 — Osborne 1 SSDD — EPSON QX-10
(Circle One) Morrow MD2 — Morrow MD3 — MS-PCDOS STD 5" DSDD — XEROX 5" DSDD
IBM CPM-86 5" SSDD — Zenith Z90 — 8" SSSD — Apple 35 TRK CPM
Other 5" Computer ______________ Sides ______ Den ______

Send Check or Money Order to:
TCI
17733 205th Ave. NE
Woodinville, WA 98072
Washington residents must add 8.1% Sales Tax

Name ______________
Address ______________
City ______________ State ____ Zip ______
Phone Day ______________ Night ______________
Mastercard/Visa ______________ Exp ______

MASTER CARD **PHONE ORDERS 1-206-788-9732** VISA

TOMORROW'S COMPUTING INNOVATIONS

Programming A Laboratory Mouse

By Earl Brabandt
Intel Corporation
1900 Prairie City Rd. FM2-66
Folsom, CA 95630

Teaching A Rodent New Tricks

If you've purchased a mouse for your PC, you've undoubtedly used it to paint pictures. Never mind that you could have done a better job in less time at a drafting table, and that anything short of a high-priced laser printer adds a decidedly digital flavor to your analog rendering. This article is about creating your own mouse interface. It's not hard once you understand mouse talk.

Maybe you've discovered you like to take mouse in hand and drag cute icons around the screen, even when you're just copying, deleting, or printing files. Maybe you've installed one of the mouse-driven word processing programs so you can cut and paste text with a flick of your wrist.

This article should serve as a reasonable tutorial on making your code respond properly to mouse movements. Hopefully this will encourage you to try projects similar to the ones which follow. (I assume you have a Microsoft Mouse User's Guide or the equivalent. The manual is usually included with the mouse.)

And, let's face it, working with a mouse is fun, and for some applications, the point and click operation and hand-eye interface is very natural. Often, data is most easily evaluated and manipulated when represented spatially or graphically.

Driving A Liquid Crystal Lens

I recently used a mouse to control an experiment in an optics laboratory. The task was to manipulate the 64 voltages that drive a liquid crystal lens. If you don't understand all of the hardware and physics stuff that follows, don't worry; I'll be getting to the programming aspects shortly.

A liquid crystal lens is similar to a liquid crystal display. A birefringent material (a material possessing an index of refraction which varies with the direction of polarization of the incident light) is sandwiched between glass plates covered with transparent electrodes.

The liquid crystal molecules rotate when an electric field is applied across the electrodes. Thus, the liquid crystal's index of refraction changes as the electric field changes. (Assuming you start with polarized light.)

A conventional lens varies in thickness, thus refractively focusing the rays passing through it. The classic converging spherical lens, for instance is thicker in the center than at the edges so the center's optical path length is longer.

Optical path length is essentially a measure of the time light takes to travel through an object. You can achieve the same effect by varying the index of refraction of the lens material. The liquid crystal lens operates in this manner.

Focusing capability is produced by varying the index of refraction spatially in the lens. So, by increasing the index of refraction at the center of the lens with respect to the edges, the center region becomes optically "thicker."

Unlike a conventional lens, the liquid crystal version can be controlled or adjusted by electrically changing the index of refraction.

The focal length of the lens may be changed simply by changing the voltages on the electrodes. Also, fast image transforms could make real time image processing or adaptive image correction possible.

Using A Mouse

Getting back to the mouse. We use a PC in the optics lab to control the liquid crystal lens via a custom 64-channel digital to analog (D/A) converter..

(Editor's note: If you're not quite sure about D to A, check out Bruce Eckel's article in this issue.)

We can write a user interface program in Turbo Pascal to supply several modes of operation low level routines for direct memory access (DMA) transfers to the D/A board, and two ways to talk to the mouse.

A user can view an image produced by the lens and adjust it with the mouse. He can also randomly access and set any of the 64 electrode voltages in text mode. In graphics mode, the PC's monitor displays the resulting image.

The DMA data transfers from memory to the D/A board ensure that commands to change electrode voltages will be acted on quickly. In fact, as the user changes the levels with the mouse, his efforts are virtually unaffected by the intermittent data transfers to the D/A. The user simply changes the voltage levels as desired, and milliseconds later, the new voltage levels are output to the appropriate electrodes.

Humans are very adept at operations involving qualitative visual correlations between a control display and a visual image produced by the system. Essentially, during interactive operation, a human closes the lens control system's feedback loop. And, thanks to the mouse, it's fun!

With Turbo Pascal

You can use the Turbo Pascal code in Figure 1 to set up screens and manage mouse operations for the liquid lens control program. Other modes of operation and the management of things like the DMA and D/A are specific to the hardware used in the lens experiment (and would just complicate this lesson), so I've omitted that code.

Turbo Pascal's built-in "intr" procedure and its Graphix Toolbox really helped me write this code. (If you don't have the Graphix Toolbox, you won't be able to run the graphics display mode. But, you will be able to run the text mode.)

Graphics Support

I originally wrote the program for systems with Hercules monochrome graphics cards (HGC). However, I've commented out the Hercules code in Figure 1. As the program now stands, it will run on the standard IBM color graphics adapter (CGA) or compatible.

If your computer has the Hercules monochrome card, simply delete the CGA code and remove the comment delimiters protecting the HGC code. You'll need the higher resolution of the HGC to see some of the detail in the graphics mode. But whichever graphics card you have, be sure you've properly notified the Turbo Graphics Toolbox.

Compatibility

A note about compatibility all PC video cards are not alike. Neither are all versions of Turbo Pascal and Turbo Graphix Toolbox. When Gary (Entsminger) and I tested the code, we found inconsistencies between Turbo Pascal versions 3.00B and 3.01A, and Graphix Toolbox versions 1.01, 1.05A, and 1.07A. The code as published works correctly when compiled with Turbo Pascal 3.01A (the latest) and Graphix Toolbox 1.05A. The code works almost correctly with Graphix Toolbox 1.07A (the latest), but writes the coordinate scale at the top rather than the bottom of the screen (a minor matter).

It's also necessary to rename a variable found in the Turbo Graphix Toolbox "typedef.sys" and "graphix.sys" files. See the comment at the top of Figure 1.

Interrupting For A Mouse

The Microsoft compatible mouse is accessed through DOS interrupt 51 (33H). A software driver comes with your mouse, so see your mouse user's guide for installation instructions before running the code printed here.

The Microsoft User's guide details the use of the eighteen mouse functions available through INT 33H. (See Table 1.) Some Microsoft compatible mouse

(continued next page)

Figure 1 - Mouse Control in Pascal

```
program LabMouse;
{
Note: due to the use of the reserved word ``window'' in the Turbo
Graphix Toolbox files ``typedef.sys'' and ``kernel.sys'', you'll have
to do a little work on these files before trying to run this, or
you will get an assignment compiler error. It appears that our
friends at Borland pulled a good one and declared a ``Window''
variable in the Toolbox routines.


Unfortunately there's already a ``Window'' procedure in standard
Turbo Pascal. For this reason, it really should be a reserved
word. The fix is to do a search/replace (^QA) in the Turbo editor
for the string ``window:'' and the string ``window :'' in the
typedef.sys file. Replace them with ``WindowArray:'' (leave out the
quotes but keep the colon in there). Type GNU at the options
prompt to be certain of changing all occurrences.


Then, do a search/replace for the string ``window['' in the
kernel.sys file. Replace it with the string ``WindowArray[''
(again, leave out the quotes but keep the [ sign). Use the GNU
option to change them all.


This isn't a problem if you don't use the ``Window'' procedure in
programs that use the Toolbox, but this code uses both the
Toolbox and the built-in ``Window'' procedure. }


const
   NumLines = 7;        {CGA scan lines numbered 7 at bottom to 0 at top}
   {Next line for Hercules Video}
   (*
   NumLines = 13;       {Herc scan lines numbered 13 at bottom to 0 at top}
   *)


type
   Table = array[1..64] of Integer;  {array to store electrode voltages (mV)}
   CursorMasks = array[0..31] of integer;  {mouse graphics cursor masks}

   RegPack  = record
                AX,BX,CX,DX,BP,SI,DI,DS,ES,Flags : Integer;
              end;


var
   Regs                    : RegPack;
   CellNum,
   Enable,
   Count                   : integer;
   InputTable              : table;
   Selection               : char;
   OK                      : boolean;


{$I typedef.sys}        {type definitions from Graphix Toolbox}
{$I graphix.sys}        {graphics routines from Graphix Toolbox}
{$I kernel.sys}         {graphics kernel from Graphix Toolbox}
{$I mouse.sys}          {mouse routines}


procedure CramBuffer  (AX, BX, CX, DX: Integer);
{Allows left mouse button to act like a keyboard return by using
 mouse interrupt capability.  Register contents AX-DX are not used
 by the routine because there is only one condition which causes an
 interrupt. (left button released)}
```

(Figure 1 continued next page)

(continued from page 7)

drivers, such as the Logitech driver, have additional functions. Obviously, if you use the extended functions in your programs, they will not be supported by the Microsoft driver

Table 1 - Mouse Functions

Function	Purpose
0	Mouse installed flag and reset.
1	Show cursor.
2	Hide cursor.
3	Get mouse position and button status.
4	Set mouse cursor position.
5	Get button press information.
6	Get button release information.
7	Set horizontal position range.
8	Set vertical position range.
9	Set graphics cursor block.
10	Set text cursor.
11	Read mouse motion counters.
12	Set user-defined subroutine input mask.
13	Light pen emulation mode on.
14	Light pen emulation mode off.
15	Set mickey/pixel ratio.
16	Conditional off.
19	Set double speed threshold.

Figure 1 contains the mouse routines used in the lens control program. The simplest routines merely load the function number into the CPU's AX register. Other parameters are loaded into the BX, CX, and DX registers before the software interrupt is placed.

"TextCursor" sets the mouse cursor to the hardware cursor mode. A software text cursor is also available to change the attributes of a character under the cursor. Some or all of the cursor scan lines may be displayed in the cursor. This is done by writing the numbers of the first and last lines to be displayed in the CX and DX registers, respectively. The cursor may be turned off by placing the number of the bottom line (7 for CGA, 13 for HGC) in the CX register and a smaller number in the DX register. The cursor will "wrap around" and turn off.

"SetXLimits" and "SetYLimits" restrict the movement of the mouse to a limited range on the screen, and "GetPosition" returns the mouse cursor position and the status of the mouse buttons. "CursorOn" and "CursorOff" turn the graphics cursor on and off.

The graphics cursor must be turned off to write to the screen in graphics mode. Otherwise the cursor will inter-

Figure 1 - Continued

```
type
   pointer = ^byte;
var
   BuffPtr :   integer absolute $0:$41C;  {determines head pointer in queue}
   BufferPointer : pointer;               {head pointer in queue}
begin
   BufferPointer := Ptr(0,(BuffPtr+$400));  {pointer to current queue position}
   BufferPointer^ := $0D;                   {cram carriage return into queue  }
   BufferPointer := Ptr(0,(BuffPtr+$401));  {pointer to next queue position   }
   BufferPointer^ := $1C;                   {cram linefeed into queue         }
   if BuffPtr = $3C then                    {reset position pointer so that   }
      BuffPtr := $1E                        {BIOS will read carriage return   }
else                                        {and linefeed.                    }
      BuffPtr := BuffPtr + 2;               {increment queue head pointer     }
end;


procedure Beep;       {allows your choice of duration and frequency}
begin
   Sound(440);        {frequency}
   Delay(500);        {duration}
   NoSound;
end;


procedure ShowInputTable(InputTable: Table); {updates screen with values (mV)}
var
   X,Y:        integer;

begin
   for Count := 1 to 64 do begin            {64 voltages to update}
      NormVideo;
      X :=((Count-1) mod 8)*8+21;           {screen X coordinates}
      Y :=((Count-1) div 8)*3+2;            {screen Y coordinates}
      Window(X,Y,X+3,Y+2);                  {use window to restrict write}
      ClrScr;                               {clear window}
      GotoXY(2,1);                          {position cursor in window}
      write(Count);                         {write heading}
      LowVideo;                             {low video for voltage display}
      GotoXY(1,2);                          {position cursor in window}
      write(InputTable[Count]);             {write voltage (mV)}
   end;
Window(1,1,80,25);                          {reset window to full screen}
end;


procedure SetScreen;                        {sets up screen}
type
     SmallStr = string[17];

procedure WriteBlk(X,Y :integer; Heading :SmallStr);   {writes headings}
begin
   GotoXY(X,Y);
   Window(X,Y,X+18,Y+2);                    {use window to restrict write}
   ClrScr;                                  {clear window}
   GotoXY(2,2);
   Write(Heading);
end;


begin {procedure SetScreen}
   TextCursor(NumLines,1);                  {no cursor}
   TextBackground(0);                       {underline bright video}
   TextColor(9);
   GotoXY(22,1);
   write('LIQUID CRYSTAL LENS CONTROL PROGRAM  type ''q''or ''Q'' to Quit');
   TextBackground(7);                       {reverse video}
   TextColor(0);
   WriteBlk(1,10,' E: EDIT TABLE');
   WriteBlk(1,18,'  G: GRAPHICS');
   TextBackground(0);                       {normal video}
   TextColor(7);
   Window(1,1,80,25);                       {reset window to full screen}
   ShowInputTable(InputTable);              {update screen}
end;
```

(Figure 1 continued next page)

Figure 1 - Continued

```
procedure GetInput(CellNum : Integer; var InputTable : Table);
                          {gets user entry for an output (1 to 64) }
var
   Voltage     : Integer;
begin
   LowVideo;
   ClrScr;
   TextCursor(NumLines-1, NumLines);          {underline cursor}


   repeat
      {$I-} readln (Voltage); {$I+}
      OK := (IOresult = 0);
      GotoXY(1,1);                     {calling routine has defined window}
      ClrScr;                          {clear window}
      if (NOT OK) or (Voltage > 5000) or (Voltage < 0) then
         Beep;
   until OK and ((Voltage <= 5000) and (Voltage >= 0));


   write(Voltage);
   InputTable[CellNum] := Voltage;    {update tables}
   Window(1,1,80,25);                 {reset window}
   NormVideo;
   TextCursor(2,NumLines-2);
end;


procedure Display( GraphMin, GraphMax : integer; var InputTable : table);
                   {Scales and generates graphical display of data}


var
   Step,
   LabelPos,
   RightSide                         : integer;
   Text                              : string[4];


begin
   Step := Round((GraphMax-GraphMin)/10);      {step scaling for graph}
   DefineWindow(1,0,0,XMaxGlb,YMaxGlb);        {define graphics window}
   DefineWorld(1,0,70,5000,0);
   SelectWorld(1);
   SelectWindow(1);
   SetClippingOn;
   SetLineStyle(0);


   (*
   {code commented out for CGA use}
   for Count := 1 to 64 do begin              {display mV on left side}
       str(InputTable[Count],Text);
       DrawTextW(0,Count,1,Text);
   end;
   *)


   DefineWindow(2,3,0,XMaxGlb,YMaxGlb);
   DefineWorld(2,GraphMin,70,GraphMax,0);
   SelectWorld(2);
   SelectWindow(2);


   (*
   {code commented out for CGA use}
   for Count := 1 to 64 do begin
       if InputTable[Count] > GraphMax then     {do clipping check--Turbo}
          RightSide := GraphMax                 {clipping is unreliable here}
       else RightSide := InputTable[Count];
       DrawLine(GraphMin,Count,RightSide,Count); {line to represents voltage}
   end;
   *)
```

(Figure 1 continued next page)

fere with the write operation.

The general sequence to write to the screen in graphics mode is to get the mouse position and button status with "GetPosition." Then, when the user pushes the correct button, you update the screen by turning off the cursor, calling a graphics procedure to modify the screen, and finally, turning on the cursor again.

"MakeGraphCursor" creates a graphics cursor. You may design your own cursor for your own application. Arrows, boxes, pointing hands, and fingers are popular. Two 16-by-16-bit masks are needed. The masks are stored in the array called "Cursor."

The first 16 locations of the array (32 bytes) contain the 16-by-16-bit screen mask. The last 16 locations contain the cursor mask. Visualize these two masks overlapping one another and overlapping the 16-by-16-bit screen image where the mouse cursor is to appear.

The mouse software forms a graphics cursor by first performing a bitwise AND operation of the screen background with the screen mask. Then the software XORs (exclusive-ORs) the result with the cursor mask to make the final image that moves around the screen when you move the mouse.

Another way of looking at it is to imagine the screen mask determining whether the original background pixel has an effect on the final image.

Take one bit position in the 16-by-16 matrix. If the screen mask bit for that position is 0, then the displayed bit will be the cursor mask bit (again, for that bit position). If the screen mask bit is 1, then the original background pixel will be inverted if the cursor mask bit is 1, and it will remain unchanged if the cursor mask bit is 0.

It helps to fill in a couple of 16-by-16 sections on graph paper with 1's and 0's when you're designing masks for a new cursor.

The pixel coordinate on the screen underlying the cursor "hot spot" is used to select location with the mouse. This spot on the cursor can be placed anywhere in the 16-by-16 pixel region through the BX and CX registers.

Interrupt Handling

The procedure "IntSet" sets up an interrupt handler for a user-written in-

(continued next page)

(continued from page 9)

terrupt routine. In this case, the handler is the procedure "CramBuffer." Mouse function 12 allows an interrupt to occur on mouse movement, button presses, button releases, or combinations of the events. In this program, I programmed an interrupt mask of 4 which causes the procedure "CramBuffer" to be executed every time the left button is released.

Whenever one or more of the conditions defined in the mask occur (in this case there's only one condition), the mouse software stops the execution of the program and calls the interrupt handler at the address specified in register DX. The software passes the status of the interrupt to the subroutine in registers AX, BX, CX, and DX. Because there's only one interrupt condition specified in the mask, this information isn't used by "CramBuffer."

Before you can call "CramBuffer," you need to do a little housekeeping to maintain the data segment (DS). Because the interrupt may occur at any time, you must save the DS before calling "CramBuffer," and restore it afterward.

This is handled by executable code stored as constants in "IntSet." Constants are stored in the code segment (CS) in Turbo so the CS: instruction forces the use of the code segment for memory operations.

The "CramBuffer" routine inserts a carriage return/linefeed sequence into the BIOS keyboard queue. Thus, when the left mouse button is released, it's like hitting a keyboard carriage return. This feature is used in the procedure "EditTable." I know, I know, it's not really that useful here, but I thought it would be good to illustrate the mouse interrupt capability as well as the BIOS keyboard queue.

The BIOS maintains a circular queue of 16 two-byte entries for keyboard buffering starting at seg:ofs = 0000:041E. A queue tail pointer exists at 0000:041A. A head pointer is maintained at 0000:041C. When the tail pointer isn't equal to the head pointer, the BIOS knows it has to read some characters starting at the location of the tail pointer. Then it must update the tail pointer.

I like to think in terms of the tail

(continued on page 12)

Figure 1 - Continued

```
    LabelPos := GraphMin;
    for Count := 1 to 10 do begin                {draw scale at bottom}
       str(LabelPos,Text);
       DrawTextW(LabelPos,67,2,Text);
       DrawLine(LabelPos,66,LabelPos,65);
       LabelPos := LabelPos + Step;
    end;
       LabelPos := LabelPos + Step;
       DrawLine(GraphMax,66,GraphMax,65);
end;


procedure GraphMode(var InputTable : table);
              {allows graphical display and entry of date with mouse}


var
     Range,
     M3,M4,
     Voltage,
     GraphMin,
     GraphMax,
     ButtonPush,
     RightLine          :integer;
     VideoMode          :integer absolute $40:$49;  {DOS stores current video mode}
     Cursor             :CursorMasks;
     Text               :string[4];


const
  Scale = 3;
  (*
  {Next line for Hercules Video}
  Scale = 5;
  *)
  HotX = 8;
  HotY = 8;
  HgcPageZero = 6;                               {Hercules graphics mode}


begin
NormVideo;
TextCursor(Numlines-1, NumLines);
GraphMin := 0;                                   {default values for graph}
GraphMax := 5000;                                {dimensions             }
  repeat
     GotoXY(1,25);
     write('Enter Display Minimum: ');
     ClrEol;
     {$I-} read (GraphMin); {$I+}
     OK := (IOresult = 0);
     if NOT OK then Beep;
  until OK and ((GraphMin <= 5000) and (GraphMin >= 0));
  repeat
     GotoXY(35,25);
     write('Enter Display Maximum: ');
     ClrEol;
     {$I-} read (GraphMax); {$I+}
     OK := (IOresult = 0);
     if NOT OK then Beep;
  until OK and ((GraphMax <= 5000) and (GraphMax > GraphMin));
  initgraphic;                                   {Toolbox initialization}
  SetBreakOff;                                   {no breaks during Graphics}


  (*
  {Next line for Hercules Video}
  VideoMode := HgcPageZero;
  *)
```

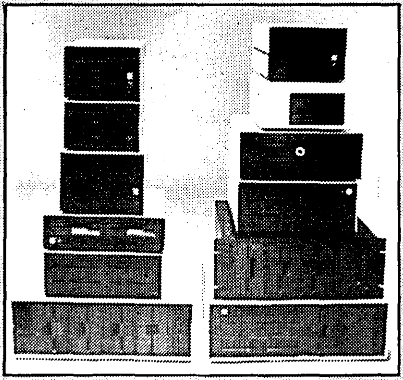
(Figure 1 continued next page)

Figure 1 - Continued

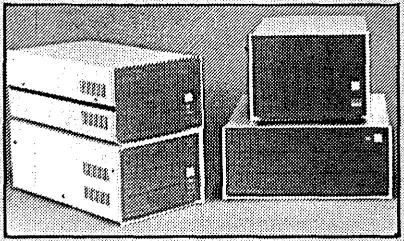
```
Display(GraphMin, GraphMax, InputTable);
MouseReset(Enable);            {Initialize Mouse Driver}
for Count:= 0 to 3 do          {make a nice box for a cursor with masks}
  cursor[Count]:= $FFFF;       {first 16 locations for screen mask}
  cursor[4]:= $F00F;
for Count:= 5 to 10 do
  cursor[Count]:= $F7EF;
  cursor[11]:= $F00F;
for Count:= 12 to 15 do
  cursor[Count]:= $FFFF;
for Count:= 16 to 18 do
  cursor[Count]:= $0000;
for Count:= 19 to 20 do
  cursor[Count]:= $1FF8;
for Count:= 21 to 26 do
  cursor[Count]:= $1818;
for Count:= 27 to 28 do
  cursor[Count]:= $1FF8;
for Count:= 29 to 31 do
  cursor[Count]:= $0000;


MakeGraphCursor(Cursor, HotX, HotY);
SetXLimits(24,XScreenMaxGlb);       {Set Min and Max Horizontal Position}
SetYLimits(0,YMaxGlb-30);           {Set Min and Max Vertical Position}
CursorOn;                           { Turn on Mouse cursor }
DefineWindow(1,0,0,XMaxGlb,YMaxGlb);
DefineWorld(1,0,70,5000,0);         {screen scaled for new coordinates}
SelectWorld(1);
SelectWindow(1);
SetLineStyle(0);                    {solid lines}
Range := GraphMax-GraphMin;


repeat
    GetPosition(ButtonPush,M3,M4);     {returns mouse button pushed}
    if ButtonPush = 1 then begin;      {paint lines if first button}
      RightLine := (Trunc((M4-1)/Scale))*Scale+4;
      Voltage := GraphMin + round(((M3-24)/(XScreenMaxGlb-24))*Range);
                {scale cursor position to voltage}
      CellNum := Trunc((RightLine-4)/Scale+1);  {determine electrode}
      InputTable[CellNum] := Voltage; {update tables}


      (*
      {code commented out for CGA use}
      str(InputTable[CellNum],Text);  {update text}
      *)
      CursorOff;                      {must draw with cursor off}
      SetColorBlack;                  {to write over old line   }


      (*
      {code commented out for CGA use}
      DrawTextW(0,CellNum,2,Chr(27)+'4'+Chr(27)+'4'+Chr(27)+'4'+Chr(27)+'4');
                        {wipe out old text on left side of screen}
      *)
      DrawStraight(24,XScreenMaxGlb,RightLine);       {wipe out old line}
      SetColorWhite;
      (*
      {code commented out for CGA use}
      DrawTextW(0,CellNum,1,Text);           {update new text}
      *)


      DrawStraight(24,M3,RightLine);         {draw new line}
      CursorOn;                              {turn cursor on}
    end;
  until ButtonPush = 2;                      {exit graphic if 2nd button}
leavegraphic;
end;
```

(Figure 1 continued next page)

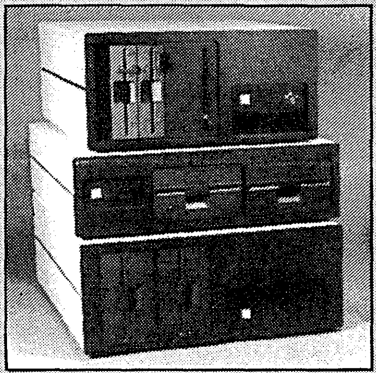
ENCLOSURES & POWER SUPPLIES



for FLOPPY WINCHESTER TAPE DRIVES from $100



for SINGLE BOARD COMPUTERS from $125



for IBM & S-100 SYSTEMS from $200

*1 piece; prices lower in quantity.
(Disk drives not included)

Call or write for free catalog

INTEGRAND RESEARCH CORP.

8620 Roosevelt Ave. • Visalia, CA 93291
209/651-1203
TELEX 5106012830 (INTEGRAND UD)

IBM TM International Business Machines

(continued from page 10)

pointer chasing the head pointer, but on the other hand, maybe the head chases its tail around the circular queue. You can figure out your own mnemonic. The present queue position may be obtained by adding 400H to the value in 0000:041C.

You can write your own routines to corrupt the keyboard queue with characters other than a carriage return and linefeed. I haven't had any problems with this method, but as with any technique that messes directly with memory locations reserved by the BIOS or DOS for housekeeping, it's difficult to ensure that it will always work as expected.

Wrap Up

Well that's it. Little white lab mice (mooses or whatever?) rarely bite. Thanks to INT 33H and the Turbo "intr" procedure, there really isn't any problem writing to a mouse. In fact, you can reduce the time required to program a mouse in your application by purchasing one of the optional mouse programmer's packages. Routines written in several languages are available from Logitech.

■■■

Figure 1 - Continued

```
procedure EditTable;
     {allows mouse editing of table of 64 electrode voltages}

var
   M2,M3,M4          :integer;
   XCoord,YCoord,
   CellX, CellY      :byte;


begin
IntSet($0004,Ofs(CramBuffer));  {sets interrupt for left button push}
TextCursor(2,NumLines-2);
  repeat
     SetXLimits(136,632);         {Set Min and Max Horizontal Position}
     SetYLimits(8,192);           {Set Min and Max Vertical Position}
     GetPosition(M2,M3,M4);             {get mouse status}
     CellX := Trunc((M3/8-18)/8);       {get coordinates of electrode}
     CellY := Trunc((M4/8-1)/3);
     XCoord := CellX * 8 + 21;
     YCoord := CellY * 3 + 2;
     GotoXY(XCoord, YCoord);            {move cursor to proper position}
     if KeyPressed then begin           {get new value if keypressed}
        CellNum := (CellX + 8 * CellY) + 1;
        Window(XCoord, YCoord+1, XCoord+3, YCoord+2);
        GetInput(CellNum, InputTable);
     end;
  until (M2 = 2);                       {exit this mode for right button push}
MouseReset(Enable);  {Reinitialize Mouse Driver}
TextCursor(NumLines,1);
end;



begin {main body of program LabMouse}
  MouseReset(Enable);  {Initialize Mouse Driver}
  if (Enable = 0)  then begin
     writeln('Please install mouse driver');     {exit program if no driver}
     exit;
  end;
  ClrScr;
  FillChar(InputTable,SizeOf(InputTable),0);
  SetScreen;
  NormVideo;


  repeat
    repeat
      read(kbd,selection);
    if NOT (selection IN ['E','e','G','g','Q','q'])
       then Beep;
    until (selection IN ['E','e','G','g','Q','q']);

    case selection of
      'E','e':  EditTable; {two modes of input available here}
      'G','g':  begin
                  GraphMode(InputTable);
                  SetScreen;
                  NormVideo;
                end;
    end;
  until (selection='q') or (selection='Q');    {to quit}


  TextCursor(NumLines-1, NumLines);          {restore cursor}
  ClrScr;
  end.
```

End of Listing

■■■

BUILD YOUR OWN SYSTEM FOR A FANTASTIC LOW PRICE!

VIDEO CARDS

Color/Graphics	63.00
Color/Graphics/Parallel	76.00
EGA Graphics	259.00
Mono	49.00
Mono/Graphics	66.00
Mono/Graphics/Parallel	76.00
Mono, Hercules Compatible, CGA, Color Emulation	250.00

EXPANSION CARDS

Clock Card	25.00
Floppy Disk Controller —2 drives	33.00
Game Port	20.00
Multi-Function, 1 ser/par/clk/game/ 2 floppy	89.00
Parallel (printer)	21.00
Parallel (printer)/serial	61.00
Serial Port (RS232) 1 port + 1 optional	32.00
Serial Port (RS232) 2 port	40.00
640K RAM (ØK installed)	39.00
XT/AT RS232 (4 port/2 installed)	69.00
AT Hard Drive & 2 floppy controller (WD)	225.00
AT 2MB Multifunction	153.00

CASES POWER SUPPLY

150 Watt Power Supply (XT)	62.00
200 Watt Power Supply (AT)	105.00
XT Fliptop Case	36.00
XT Slide Case	36.00
XT/AT with Lock & LED	59.00
AT with Lock & LED	81.00

MONITORS

Amdek Amber 310A	175.00
Packard Bell EGA/CGA (Auto Switch)	507.00
NEC Multisync	740.00
Quadram Amberchrome	195.00
Quimex CGA (Green switch)	450.00
Samsung Amber 12" TTL	96.00
Samsung Green 12" TTL	89.00
Samsung Amber w/Swivel	113.00

Prices are subject to change without notice. Exact shipping CHARGES will be added.

MOTHERBOARDS

XT/Turbo 4.77/10mhz	145.00
AT 6/10 mhz (5 layer)	499.00
XT 640 (2 layer)	109.00
XT/Turbo 4.77/8 mhz (2 layer)	125.00
XT/Turbo 4.77/8 mhz (4 layer)	172.00
For XT: need 18 pcs. each of 64K & 256K for 640K memory	75.00
For AT: 640K	75.00
1MB	125.00
***w/Adaptec	480.00
AT 30 MB Seagate #4038	625.00

NO WARRANTY ON BOARD WITHOUT MINIMUM MEMORY ON IT

KEYBOARDS

5339 Professional XT-AT w/12 function key	80.00
5060 Keyboard AT Style	63.00

FLOPPY DISK DRIVES

Fujitsu 360K	97.00
Toshiba 360K	109.00
Teac 1.2 MB	160.00
Toshiba 1.2 MB	150.00

HARD DRIVE & CONTROLLER

20 MB Miniscribe or NEC HD	367.00
30 MB High Density Drive w/Adaptec	480.00
AT 30 MB Seagate #4038	625.00
Western Digital Controller w/Cable (XT)	113.00
Adaptec RLL Controller (XT)	180.00

20 MB Miniscribe or NEC Hard Drive with Controller Card......... 399.00

ACCESSORIES

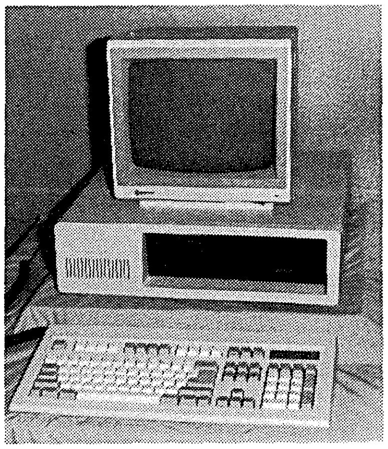
1200 Baud Modem (Leading Edge Model L) Hayes compatible	150.00
MS-DOS 3.2 with GW Basic	85.00
MS-DOS 3.1	35.00
Joystick (IBM Style)	25.00
V20-8mhz	17.00
Memory Chips	(call for prices)

BUILDING YOUR OWN CLONE

****FREE BOOKLET****
Contains a piece by piece explanation of how various IBM PC compatible cards interface. Includes check list for custom system design. Call or write to order.

SPECIAL XT KIT

Includes:
4.77/10 mhz XT with 640K, Mono/Graphics, Slide Cabinet, 150 Watt Power Supply, 12 Function Key Keyboard, Amber Monitor, 2 DSDD Disk Drives, Serial, Parallel, Clock & Game Ports, DOS 3.1.
Sale Price $695.00
With 20MB Hard Drive & 1 DSDD Floppy Drive ...**Sale Price $995.00**
Shipping Extra



**90-day warranty/30-day money-back (Subject to restock fee).*

Free instructions with each system.

MicroSphere, Inc.
P.O. Box 1221
Bend, Oregon 97709
(503) 388-1194

Hours: Monday-Friday 8:30-5:30 Pacific Time

Build A MIDI Interface For Your PC

By Jay Kubicky
934 North Orange St.
Media, PA 19063

A Project For Cloned Musicians

RS-232 isn't the only serial for your computer. In fact, RS-232 isn't even the best choice for some serial applications because MIDI is the interface of note for digital key bangers and their cousins who dance to a (very) different drummer. Join Jay as he looks at MIDI's bit part.

The MIDI Interface is the hottest thing to hit the music industry since the introduction of the electronic synthesizer. MIDI, an acronym for Musical Instrument Digital Interface, is a relatively simple hardware interface/software protocol that allows the interconnection and control of almost any electronic musical instrument. It also proves that things can be simple and still work well, even when computers are involved.

I've divided this article into three parts. First, I'll go over some of the uses of MIDI in today's music world. Then, I'll cover the entire interface in depth, from hardware to protocol. Finally, I'll outline some of my ideas on sequencing and MIDI software.

The Bottom Line

All sorts of devices use MIDI to pass all sorts of data. The most frequent users are keyboard players. Almost all the keyboards built in the last four years or so (which is most keyboards) are equipped with MIDI.

When you connect two MIDI keyboards together, all the notes you play on one (keyboard A) are transmitted to the other (keyboard B). Keyboard B then plays this data as if it had created it itself.

But MIDI ties together more than just keyboards. Rhythm machines (boxes that play back digitally recorded drum sounds) can transmit timing and note data over MIDI. Most of today's signal processors also exchange parameter and control data over MIDI.

Even guitarists can get into the MIDI scene. By connecting to a controller and playing on a specially retrofitted guitar, tone data can be decoded from the strings and sent over MIDI to drive other devices such as synthesizers and samplers (a sampler is a keyboard that plays digital samples of other instruments). Keep this in mind the next time you're listening to your favorite "keyboard" solo.

The Hardware

At the most basic hardware level, MIDI is an asynchronous serial interface. Data is sent and received at 31.25K baud (31,250 bits/second) in 8-bit bytes (one stop bit, no parity). I know the speed's nothing to write home about, but the Gods of MIDI wanted to make sure they weren't too quick for Commodore 64s or Apple IIs. They also wanted to minimize RF interference.

Data is transmitted over 5-line DIN-type cables. A given cable will never carry data in more than one direction at a time. Data originates at the MIDI OUT connector of the transmitting device and is received at the MIDI IN of the destination device.

For purposes of electrical isolation, each "receive circuit" is built around an opto-coupler or opto-isolator. The input portion of the isolator is driven by a 5mA current loop from the transmitting device.

MIDI THRU is simply a reproduction of the signal occurring at the MIDI IN jack. It's useful for chaining many devices to a single MIDI OUT.

See the transmit/receive circuits in Figure 1.

(Note: Figure 1 presents a complete MIDI interface circuit for IBM PC type machines. The 8253 Programmable Interval Timer is utilized by my sequencing software and isn't necessary to implement a barebones MIDI port. However, I advise you to include it. For more information on the interface, see "MIDI Project", Byte, June 1986.)

MIDI Protocol/Channel Messages

The MIDI protocol is a general set of commands used for sending both specific and non-specific information. Although all MIDI devices do not support all commands (in fact, most don't), the commands that are recognized are the same for all devices (a Note On is a Note On is a Note On). Where else in the computer world could you possibly find such standardization?

Commands are called messages. Each message is made up of one or more bytes. The first byte of every message (known as the Status byte) has its high bit set; all subsequent bytes (known as Data bytes) have their high bits reset.

Although all data is sent over a single electrical path, the MIDI protocol lets you send messages to any one of 16 receivers. The other receivers on the bus ignore the data. Thus, a single transmitter can send messages to (control) as many as 16 receivers. These messages are called Channel Voice and Channel Mode Messages.

I've illustrated the implementation of these virtual Channels in Figure 2.

Channels are specified by the bottom four bits (0-3). All other bytes in the message are unaffected by the channel number.

Channel messages are the most frequently used messages on the bus. They include messages concerning note status, note volume, pitch bending and program changes. All Channel Messages are made up of at least two

(continued on page 16)

Figure 1 - MIDI Interface Circuit

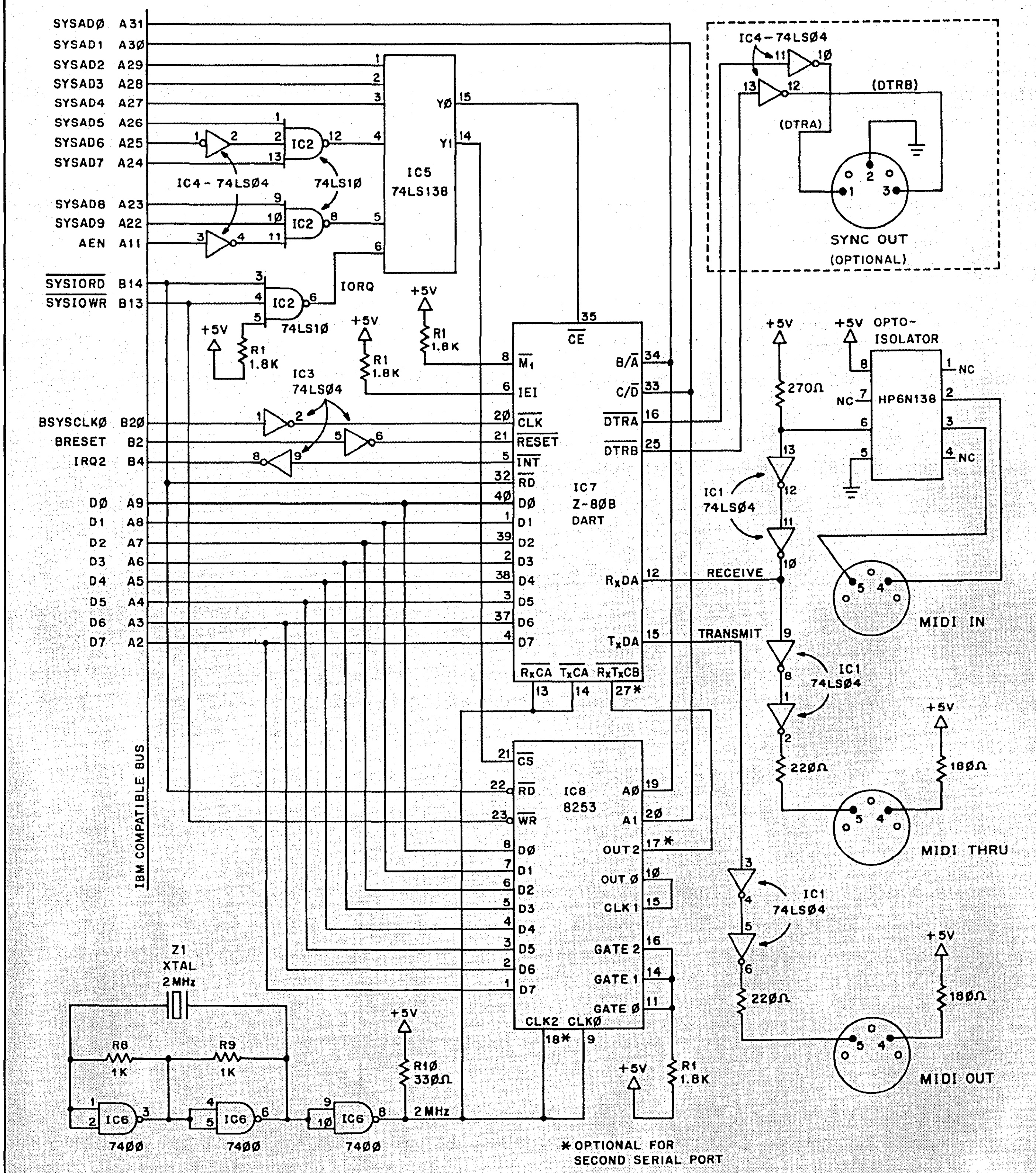


Figure 2 - Channel Message Status Byte Bit-Map

```
    Bit:7 6 5 4 3 2 1 0
---------------------
        1 N N N C C C C
```

where:
N N N is a 3-bit value between 0 and 6
defining the message type (function)

- and -

C C C C is a 4-bit value between 0 and 15
representing the channel.

Figure 3 - System Common/Real-Time Message Bit-Map

```
    Bit:7 6 5 4 3 2 1 0
---------------------
        1 1 1 1 F F F F
```

where:
F F F F is a 4-bit value
between 0 and 15 encoding the
function to be executed.

Figure 4 - MIDI Modes

OMNI ON/POLY: (Mode 1)

Transmitter:
All voice messages are sent over channel N.

Receiver:
Messages from all channels are played over all voices, polyphonically.

OMNI OFF/POLY: (Mode 3)

Transmitter:
Same as OMNI ON/POLY, above.

Receiver:
Voice messages from channel N only are played over all voices, polyphonically.

OMNI ON/MONO: (Mode 2)

Transmitter:
Voice messages for a single voice M are sent over channel N.

Receiver:
Voice messages for all channels control only one voice, monophonically (no more than one note will sound at a time).

OMNI OFF/MONO: (Mode 4)

Transmitter:
Voice messages for voices 1 through M are sent in channels N thru N+M-1, respectively. (Single voice per channel.)

Receiver:
Voice messages are received on channels N thru N+M-1 and assigned monophonically to voices 1 through M.

(continued from page 14)

bytes: one Status and one Data.

I've summarized the entire MIDI spec (well, at least MIDI 1.0) in Table 1. As far as I know, this is the first time the entire spec has been published and explained in a magazine anywhere (of course, you could always pay the IMA $35 for a copy).

The first part of the table, the Channel Messages, is what you've just spent the last few minutes reading about. But what about the System Messages? Read on...

MIDI Protocol/System Messages

Although Channel Messages make up the bulk of transmitted MIDI information, there are two other groups of MIDI messages: System Common and System Real-Time Messages.

System Messages (as a whole) are, as the name would imply, messages that apply to all receivers in the system. I've illustrated the general bit-map of a System Status byte in Figure 3.

I mentioned two subclasses of System Messages: System Common and System Real-Time. System Common Messages support functions such as tuning requests and system reset, while Real-Time messages deal with timing.

Since System Real-Time Messages have the task of keeping the bus synchronized, they may be sent at any time, even in the MIDDLE of another message. Thus, during normal operation, a Channel Status byte may be received, followed by a System Real-Time byte, followed by the Data bytes accompanying the initial Status byte. This only applies to System REAL-TIME Messages.

In order to save bus time (which can be at quite a premium at 31K baud), MIDI implements what's known as running status. Running status is simply this: if you're sending two messages of the same type (same Status byte and channel) in a row, the Status byte for the second message can be omitted. So if you were sending two Note On messages over the same channel, you could send them as follows:

```
Status
1001cccc 01100000 01000000
01100100 01001000
```

This would transmit two Note On events for notes 96 and 100 with velocities of 64 and 72. By not having to send the second status byte (it's assumed), the net bus time is reduced by around 16 percent. The more like messages sent in a row, the more time is saved.

MIDI Modes

I guess the most confusing thing of all about MIDI are the modes on the bus. Instead of trying to introduce the modes in some crafty way, why don't we just plow through 'em one at a time.

But first (seems there's always something), let's cover a little vocabulary.

To best understand anything about MIDI, you should always bear in mind that it was originally designed by keyboard players for keyboard players.

In keyboards, tones are generated by voices (also called oscillators). Most keyboards have between four and eight voices, so they can sound AT MOST four to eight different notes at a time.

There are four MIDI modes: OMNI ON/POLY, OMNI ON/MONO, OMNI OFF/POLY, OMNI OFF/MONO. Although I have been referring to "modes on the bus," MIDI modes are assigned independently to each device in the system. See Figure 4.

I should emphasize that Modes 1 and 3 are by far the most commonly used of the four modes. The Mono modes are essentially just carryovers from the early days of MIDI when buying a synthesizer meant refinancing your home and voices were at a premium. Where a single VOICE would be assigned to a given channel (through Mode 4) to achieve polyphony five years ago, a single SYNTHESIZER (through Mode 3) would be assigned today.

Shopping?

Congratulations. You've just trudged through the worst of MIDI. If you could follow most of what I've been saying, you probably have a pretty good idea of how it all works. Or do you? Well, just to be sure, we'll go over everything one more time.

But instead of a run-of-the-mill, textbook style review, we'll approach things from a more practical standpoint. In other words, get your checkbooks ready, we're going shopping.

Mondo MIDI

For a review, we're going to set up and analyze a Mondo MIDI system. Of course, we can't really set up the gear right here in the magazine (there's not enough room), but we can do the next best thing. Let me present Figure 5.

Figure 5 is our mondo MIDI setup. It is, in fact, similar to what you might find in an actual studio setup. I've divided our layout into three major sections: Input, Output, and Sync. Let's start with the Input.

The Mondo MIDI, INPUT:

As I've said, MIDI data can come from many sources. Among these are keyboards, guitars and rhythm machines. I've chosen the most common source: keyboards (yes, I see the Sync Converter, too, but we're saving that for later).

When played, these keyboards will produce MIDI messages and send them on a preassigned channel. Although there can be only one transmitter per cable, a couple of seconds on the HP-15C shows us that normal performance utilizes only a very small portion of the bus bandwith(time). How can we use some of this time? Enter the MIDI merge.

MIDI mergers are neat little black boxes that take two or more MIDI INputs and merge them into a single MIDI OUTput. They work much like conventional data multiplexers, and, like data multiplexers, can become temporarily overloaded. This will cause messages to pile up in a buffer (hopefully) until adequate bus time is free to catch up. Hence, the tradeoff: data merging for possible late notes.

It's worth mentioning that some of the newer keyboards implement MIDI merge by themselves. They do this by taking their MIDI IN, mixing it with data generated internally, and retransmitting it over MIDI OUT or MIDI THRU.

Mergers, in general, are somewhat exotic, and hardly necessary for normal operation. A single keyboard works just fine, giving us what we want: a single stream of Input data.

The Input stream is the basis of the whole system. In our system, we're recording the data into a computer for later editing and playback (more on that later). For now we'll worry about getting the data back out onto the bus. So, turn to part two of our diagram: the Output.

The Mondo MIDI, OUTPUT:

All transmitted data originates from a MIDI OUT somewhere. In many cases, you may want to drive several devices from the output of a single source.

There are two ways of splitting up MIDI data. The first is known as the MIDI THRU box, and this little guy's

(continued next page)

Figure 5 - Mondo MIDI Setup

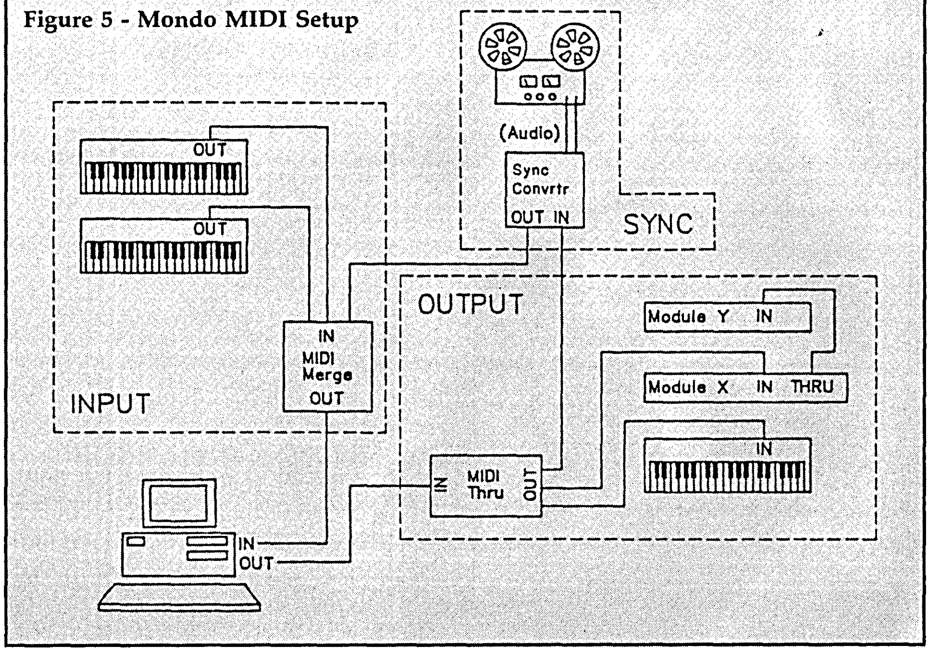


Figure 6 - Proposed MIDI data storage format:

```
Comment: ==> means 'Is stored as.'

Note On (80h, note, vel) ==>
(t) 00vvvvvv 1nnnnnnn

Note Off (90h, note, vel) ==>
(t) 00vvvvvv 0nnnnnnn

After-touch (A0h, note, vel) ==>
(t) 01000010 0nnnnnnn 0vvvvvvv

Control Change (B0h, cntrlr, val) ==>
(t) 11000000 0NNNNNNN 0ccccccc

Program Change (C0h, program #)
==>
(t) 01000000 0ppppppp

Channel Pressure (D0h, pressure) ==>
(t) 01000001 0aaaaaaa

Pitch Bender (E0h, LSB, MSB of val)
==>
(t) 10wwwwww

vvvvvv(v) - Top 6 (7) bits of note
 velocity value.

nnnnnnn- Note number.

NNNNNNN- Controller number.

ccccccc- Controller value.

ppppppp- Program number.

aaaaaaa- Channel pressure

wwwwww- Top six of 14 bit bender

 val.

(t)- This is the position of

 the timing byte(s). See

 text.
```

(continued from page 17)

sole purpose in life is to make a lot of copies of a single MIDI INput stream. Inside, he's just a whole bunch of MIDI THRU circuits wired to a common IN. Hence the name: MIDI THRU box.

But there's a more direct way of carrying a single MIDI signal to multiple receivers: the MIDI THRU JACK.

By chaining devices together, as I've shown in Figure 5, a signal can be taken as far as you like. The only problem is that not all devices have a MIDI THRU jack. (Oh well, you can't win 'em all.)

Now For Some New Stuff

And so it seems, Input and Output aren't major problems in MIDIland. Or are they? There's something we haven't considered, and for that we need a little more background information.

Synchronization & Sequencing

One of the major uses of MIDI is the digital recording and editing of music. (Well, it's not really music, but musical events.) This process of recording is generally referred to as sequencing (derived from the process of playing back sequences of notes).

We'll talk more on the theory of sequencing later, but, for now, consider this: a sequencer is playing back a recorded song. At the same time, a rhythm machine is to play back a preprogrammed drum pattern. And it all has to be synchronized over MIDI. What's the deal?

If you guessed Timing Clock, you win the prize (an IBM 3270...Oops! This isn't April any more).

The Timing Clock message (see System Real-Time Messages) is sent by the transmitter at a rate of 24 clocks per quarter note. By starting things off with the Start message, maintaining a steady stream of Timing Clocks, and stopping with Stop, the bus master keeps everything in time (see the table for more on these messages).

In our setup, we're recording all synthesizers with a multi-track tape recorder. Whenever multi-track is used with MIDI, one of the tracks must be set aside for synchronization. This is done by recording an FSK (frequency-shift keyed) or similar audio tone on the tape at 24 pulses per quarter note.

Our sync converter (and many commercial "Sync Boxes") not only generates these tones from Timing Clocks, but also reverses the process to output timing information onto the bus. We've mixed this with data from the keyboards to allow tape-synced recording into the computer.

Software & Sequencing

As I've mentioned, one of MIDI's major uses is the recording, editing, and playback of music data (sequencing). Personal computers prove to be very well suited to this task. As a conclusion to the article, I'm going to go over some methods of implementing a software sequencer.

The basic task of a sequencer is to record MIDI messages (events), so the first problem to overcome is how best to store data.

In order to most efficiently store MIDI messages in memory, we're going to to have to revamp the protocol a bit. We know that we're not going to be storing any System Messages because they don't define any real music. We also don't have to worry about channels because each channel will go in a separate buffer. All we have to save is the Channel Voice Message. Piece of cake. See Figure 6.

This protocol supports all Channel Voice messages, and saves memory. But there is a price.

The problem with the protocol is a loss of resolution in certain areas. Specifically, the velocity and pitch bender codes. The velocity loss is no big thing (only one bit), but the pitch bender is a little more radical (eight bits).

The reason for this is that most synths send out LOADS of data for even the slightest nudge on the bender. With a generous protocol, this can equate to a LOT of memory eaten up in a hurry. And pitch benders aren't all that precise anyway.

So now we have the data. But wait! I almost forgot something! We have the data, but we don't know when we got it. We need to record the timing information, and we have two choices - the absolute and relative methods.

We have to start with a timebase of some sort. Let's assume that we've set up the PIT (Programmable Interval Timer) in the IBM PC to trigger an interrupt, oh, say, 5760 times per second (5760 = 120 x 48). This interrupt is our internal metronome (set to 120 beats per minute), and our best resolution for recorded data will be 48 divisions per quarter note (or 24 per eighth note, or 3 per sixty-fourth note).

Once we've established a metronome, we have to figure out how to use it. The most direct method is the absolute. Using this method, the interrupt increments a 16-bit counter variable. The counter is then stored in memory with our earlier data. The process is simply reversed for playback.

The only real drawback to this method is its use of memory. As we shall soon discover, data can be stored just as well by using only one byte for timing. However, the absolute method is much simpler to implement and understand (let's hear it for simplicity).

Relative timing is somewhat more complicated. Instead of storing things in relation to an absolute timebase, relative timing stores only the number of beats between each message. Since, under normal circumstances, not more than five beats will pass with no bus activity, we can deal with most messages by saving only one byte (256/48 = 5 + a little bit).

But what happens if more than 256 clock ticks pass between messages? Then we must store a dummy message to keep things in order. This means wasting two bytes for every five beat pause. However, the final saving in terms of memory is probably around 20 percent.

Yeah! We can cut memory usage by one-fifth! Well, it's not that easy. Relative timing is a whole lot more difficult to code than absolute timing, especially in the playback routines. And the data (just about) has to be converted into some kind of absolute format to do any editing on it anyway. Well, that's just too much for me to think about, so I give the thumbs up to absolutism.

Of course, the ideas I've presented here are really just that: ideas. Although I've spent a great deal of time thinking about how to attack this problem, I'm not perfect. There could easily be something wonderful that I've overlooked. If you see it, let me know.

Winding Down

We've covered a lot of ground in just one short article. However, there's no substitute for experience, and to really get a complete understanding of MIDI, you must program MIDI.

The entire MIDI industry is really just getting started. With such a vast gap to be filled, it's really anybody's guess where things are headed.

Table 1, MIDI specifications is available on the Micro Cornucopia RBBS. (503) 382-7643

■■■

ANNOUNCING A

CALCULATED BREAKTHROUGH IN COMPUTING

If you use a SCIENTIFIC CALCULATOR and a PERSONAL COMPUTER, you need the

HP-PC HYPERCALCULATOR.

YOU NEED: A pop-up scientific calculator for the IBM PC **integrated with every program you use.**

YOU NEED: A programmable calculator with 100 registers, 1000 program steps, and alphanumeric prompts.

YOU NEED: A faithful emulation of the Hewlett-Packard HP-11C that runs 20-40 times faster.

YOU NEED: Only $39.95 plus $1 for shipping (includes **free 8087 version and utility programs**).

Stop copying from calculator to computer **now!**

Call toll-free:

(800) 628-2828, ext. 502

Sunderland Software Associates
Post Office Box 7000-64
Redondo Beach, CA 90277

HP-11C and IBM PC are trademarks of Hewlett-Packard Co and International Business Machines Corp., respectively

NEW LOW PRICE!

$59.95

Eco-C C Compiler

"This is the only package we reviewed that we would be willing to call a professional tool."
Computer Language, Feb., 1985

When the review mentioned above was written, the Eco-C C Compiler was priced at $250.00. Now you can have the same compiler for a mere $59.95. And that price is complete, including a library of 120 functions, all operators (except bit fields), structures, unions, long, floats, doubles, plus user's manual. We've even included a special version of the SLR Systems assembler and linker.

Benchmarks*
(Seconds)

Benchmark	**Eco-C**	Aztec	Q/C
Seive	29	33	40
Fib	75	125	99
Deref	19	CNC	31
Matmult	42	115	N/A

**Times courtesy of Dr. David Clark*
CNC - Could Not Compile
N/A - Does not support floating point

Eco-C requires 56K of free memory, 240K disk space (one or two disk drives or hard disk), Z80 CPU and CP/M 2.2 or later. We also have an MSDOS version at the same low price. Call today!

1-800-952-0472 (orders only)
1-317-255-6476 (information)

ECOSOFT INC. (317) 255-6476
6413 N. College Ave. • Indianapolis, Indiana 46220

Intro To Database Programming, Part 2

By Sandy Brabandt
6424 Sunnyfield Way
Sacramento, CA 95823

Entity-Relationship Models

Here we are again folks, the second installment in the continuing saga of Dr. Dobbs and Dee Base. I hope you'll read on as these two face the age-old question: Is Dee really off base or has Dobbs just gone to the dogs?

In our last episode (see Micro C #35 pp 14-18) we met Dr. Dobbs, the intrepid veterinarian who was attempting to stitch his business into a microcomputer. During the first few weeks of his project, he spent a great deal of money, used up all of his evenings and weekends, alienated his office staff, and lost one of his best clients.

He finally hired a consultant, Ms. Dee Base, who told him that his problem lay in poorly designed data files. She suggested a simple way to reorganize the structures he had created, and went on to convince him that they should do a complete business analysis together. That lead to a much more durable database design and much more...

An Entity-Relationship

There are many methods for designing relational databases, but Ms. Base prefers the Entity-Relationship model. The first step in this design process is to identify all of the entities used by a business.

An entity is a person, place, thing, or process that the business wants to keep data about. Dobbs decides to keep track of his clients and their accounts, including the bills he sends them, the payments he receives from them, the animals he sees, the number of visits each makes - plus routine vaccinations, medications, and procedures.

He must then define the relationships among all of these entities. Specifically, he must decide which entities relate to each other, and whether these relationships are one-to-one, one-to-many, or many-to-many.

One To, One To, Many To

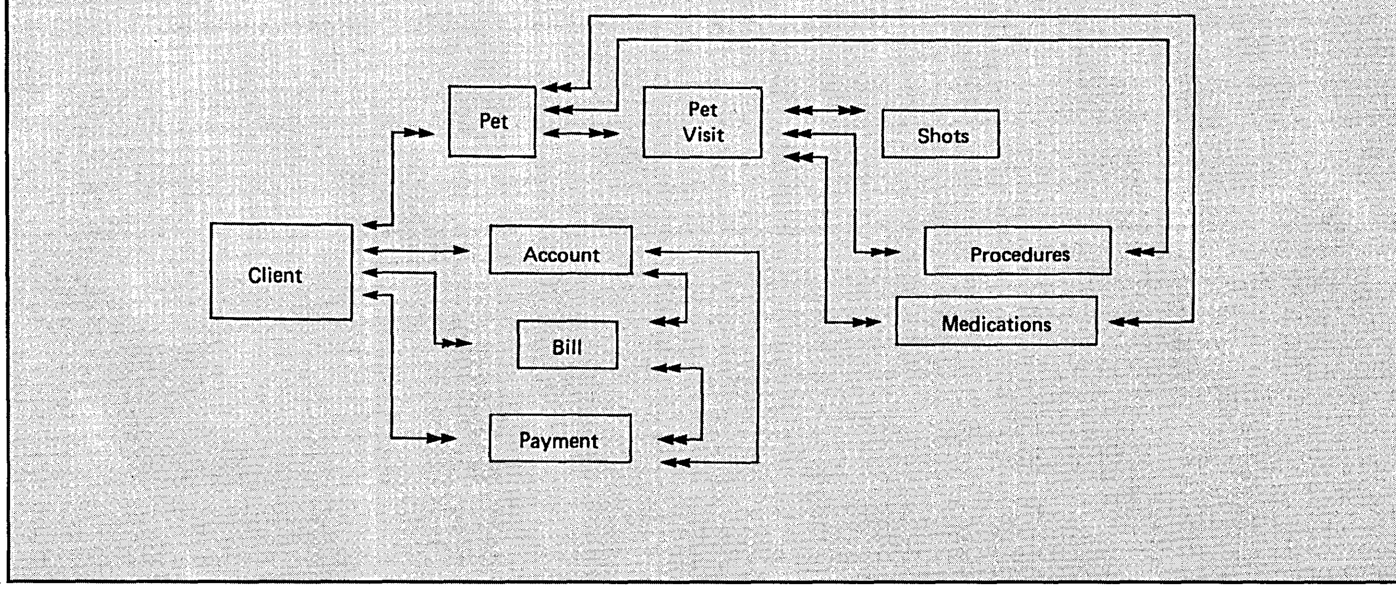
For example, each client has one account, and each account belongs to one client. The client-account relationship is one to one.

The client-pet relationship is one-to-many. One client may have many pets, but a pet may have only one owner.

The pet-procedure relationship is many-to-many, because a pet can have many procedures performed on it, while a particular procedure can be performed on many different pets.

Based on her discussion with Dobbs, Ms. Base now charts the entities and their relationships. In her chart, a single-headed arrow indicates the "one" end of a relationship, while a

Figure 1 - Entity Relationships



double-headed one indicates the "many" end of a relationship. The chart ends up looking like Figure 1.

Simplifying Relationships

The next step in the design process is to simplify the relationships as much as possible. First, eliminate the one-to-one relationship between client and account by combining them into a single entity ("client"), since a client will always have an account, and an account will always belong to a client.

Next, eliminate redundant relationships. The shots, procedures, and medications entities are all related to both the pet and the pet visit. Since shots, procedures, and medications can only be administered as part of a pet visit, and since the pet visit is related directly to the pet, the direct relationship of these entities to the pet is redundant, so we axe it.

Eliminate unnecessary relationships! For example, although payments are made against bills, Dr. Dobbs doesn't particularly care to know exactly which payments are made against which bills; he only cares about how bills and payments affect a client's account balance. So, we can remove the relationship between bills and payments.

Ms.Base also points out that "shots" are really a subset of "procedures", since giving shots is a kind of procedure. So, away with the "shots" entity. The updated chart looks like Figure 2, an improvement of two fewer entities.

Parent-Child Relationships

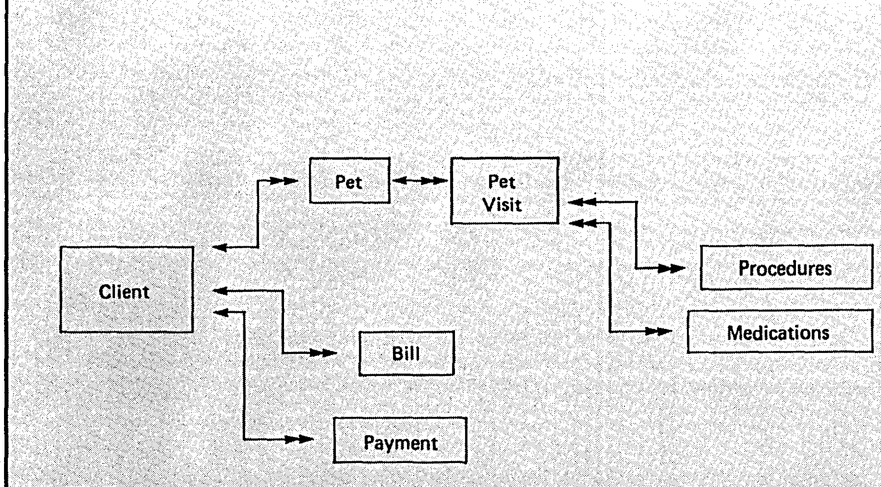
The one-to-many relationships are called "parent-child" relationships, where the "one" side of the relationship is called the parent, and the "many" is called the child. This is something of a biological misnomer, since in a relational database a child can have a theoretically unlimited number of parents, but it does enable us to speak more easily about relationships. Note that a table can be the parent of certain tables and the child of other tables at the same time.

The many-to-many relationships are a special case that we'll deal with later.

Once the parent-child relationships have been defined, the table keys can be determined. The key of a table is made up of the column or columns in the table that identify each record uniquely. In addition to identifying the individual rows in a table, the keys also provide the means to physically represent the relationships between the tables: every child table must contain the keys of all of its parent tables.

In an earlier example (Micro C #35 p 17), the Client table was the parent and the Pet table was the child, and the key of the Client table (the client ID) was stored in the Pet table, providing the link between the two tables.

Figure 2 - Updated Entity Relationships



The client ID, stored as PET_OWNER in the Pet table, is called a "foreign key." The primary key of the Pet table is the pet name. However, since there could be several pets of the same name, it takes both the PET_NAME and PET_OWNER columns to uniquely identify a row for a particular pet. From this you can see that the primary key of a table need not uniquely identify a row by itself, if a row can be uniquely identified by the combination of primary and foreign keys.

Defining Keys

To define the keys in the Entity-Relationship model, start at the top and work down. This means starting with the great-granddaddy table that has no parents, the Client table.

The key to this table, the Client ID, has already been established. It's already in the Pet table, so we add it to the Bill and Payment tables.

The Bill table has the billing date as its primary key, and the combination of the Client ID and the billing date serves to uniquely define the record. The Payment table similarly uses the payment date as its primary key.

The Pet Visit will contain the key of its parent record, the Pet table. As we discussed earlier, it takes both the primary and foreign keys of the Pet table, both the Client ID and the Pet Name, to identify a pet uniquely.

Therefore, both of these fields must be placed in the Pet Visit table as foreign keys in order to link a particular visit to a particular pet. The primary key of the Pet Visit table is the visit date, because it distinguishes one pet's visit from another.

So, continuing this procedure, we identify the primary and foreign keys of each of the tables. The final key assignments look like Figure 3.

Many-to-many Relationships

Now let's return to the many-to-many relationships in the chart.

These can't be represented directly in a relational database since there's no clear parent or child. In order to represent the many-to-many relationship between two tables, we create a third table to provide the link. To see how this works, consider these examples. In our design chart, we've indicated a

(continued next page)

(continued from page 21)

many-to-many relationship between the Pet Visit and the Procedure tables. A row in the Pet Visit table contains information about a particular pet visit, such as the date of the visit and the condition of the pet at the time of a visit.

A row in the Procedure table contains information about a particular procedure, such as how much Dobbs charges to perform that procedure. Dobbs considers this a many-to-many relationship because he wants to know what procedures he performed for a particular visit, and all of the different pets that he's performed a particular procedure on, over time. To represent these relationships, we add a Visit-Procedure table, which is the child of both the Visit and Procedure tables and, as such, contains the keys of both its parent tables.

This means it contains the Client ID + Pet Name + Visit Date (total unique key of the Visit table), plus the Procedure name (key of the Procedure table). Each row in the new table will represent a particular procedure performed at a particular visit. A few rows in the table might look like Figure 4.

Entity-Relationship Tables

Using this table, Dobbs can find out which procedures he's performed on a given pet, and also which pets he's performed a given procedure on. A table that links two other tables in this manner is called a "Relationship table" (the other tables we've worked with so far are "Entity tables").

The ability to navigate easily in both directions of a many-to-many relationship is a very important feature of Relational databases.

We also need to create a Relationship table to link the Medication and Pet Visit tables. The design chart now looks like Figure 5.

The key assignments for the new tables are shown in Figure 6.

Note: these tables don't need primary keys since the combined foreign keys are adequate to identify them uniquely.

Storage (Structures)

The Entity-Relationship model for Dobbs' vet practice is now largely finalized. The tables are defined, and the navigational structure is complete.

But Dobbs is still far from where he can begin writing programs. He needs to decide which data elements to use for storage, and which tables to store these elements in.

In order to determine his data element needs, he should now decide what information he wants to get out of the system.

With Dee's help he designs his system output - the printed and on-screen

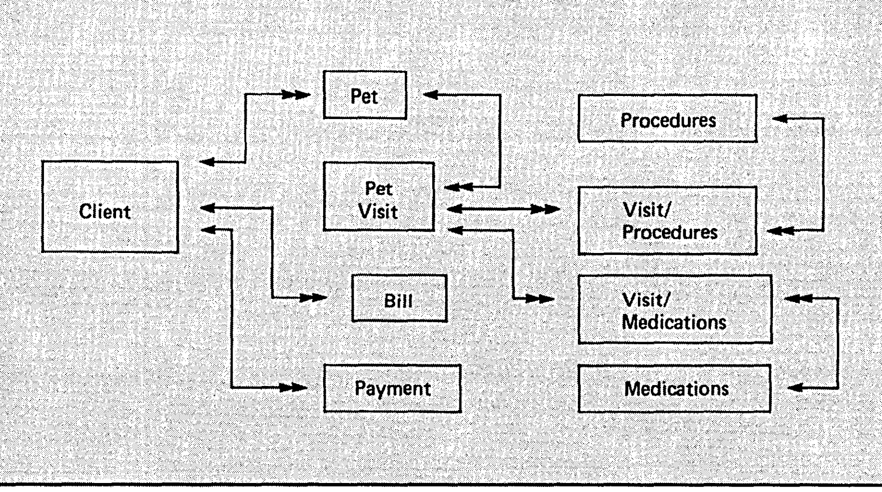
Figure 3 - Primary & Foreign Keys

CLIENT TABLE	
Primary key:	Client ID
Foreign key(s):	None
BILL TABLE	
Primary key:	Billing date
Foreign key(s):	Client ID
PAYMENT TABLE	
Primary key:	Payment date
Foreign key(s):	Client ID
PET TABLE	
Primary key:	Pet Name
Foreign key(s):	Client ID
PET VISIT TABLE	
Primary key:	Visit Date
Foreign key(s):	Pet Name + Client ID
PROCEDURE TABLE	
Primary key:	Procedure Name
Foreign key(s):	None
MEDICATION TABLE	
Primary key:	Medication Name
Foreign key(s):	None

Figure 4 - Procedure Table

ID	PET NAME	VISIT DATE	PROCEDURE NAME
13	Phydeaux	12/13/86	Rabies shot
13	Phydeaux	12/13/86	Heartworm test
13	Phydeaux	12/18/86	Foxtail removal
15	Towser	12/18/86	Rabies shot
16	Spot	12/20/86	Neuter

Figure 5 - Linking Medications & Pet Visits



reports that he wants the system to produce. From this, he can decide which data elements need to be stored in order to produce the output.

This output might include things like mailing lists, billing statements, daily production reports, pet medical history reports, and account status screens. In general, each output field on these reports and screens must either be stored directly in the database files or derived from data in the files.

As an example, he might want to show the items listed in Figure 7 on the billing statements.

After he makes a complete list of required data elements, he must assign these elements to the appropriate tables. Dee Base explains to Dobbs that these assignments will be made in three steps.

First, they'll place the data in tables using the ever-popular "best guess" method. Next, they'll normalize the tables. And finally, they'll tune the tables for performance and data integrity.

Dobbs is already a veteran at the "best guess" method and immediately starts deciding which tables the data fields should be stored in. It's fairly obvious for most of the fields. The client name and address information, for instance, obviously belongs in the Client table. The account previous balance is another item that relates directly to the client and should be stored in that table.

The procedure name is already being stored in both the Procedure and Visit/Procedure tables as part of the keys of those tables. However, its use is different in the two tables.

The Procedure table is a master file listing all of the procedures Dobbs performs, along with the standard fee he charges for each. The Visit/Procedure table is a transaction file that keeps track of which procedures are performed on which pets, where each procedure is attached directly to one pet and one date. Therefore, the procedure name in the Procedure table isn't really the same data element as the procedure name in the Visit/Procedure table.

The procedure date, another field needed by the billing process, is also stored in the Visit/Procedure table. This same logic applies to the medication names (stored in both the Medication and Visit/Medication tables), dates (stored in the Visit/Medication table), and fees (stored in the Medication table). The total of current charges need not be stored since it can be derived by adding the Procedure and Medication fees as the bill is being printed.

The dates and amounts of the client payments would be kept in the Payments table, and the total of these payments would once again be derived during the bill print. The new account balance can then be calculated by adding the charges to and subtracting the payments from the previous balance.

Dobbs goes on to assign all of the data elements he's identified to tables. Since this is just a first pass, he doesn't spend too much time on it. Once this is completed, he and Ms. Base will go on to normalize and tune the tables before any programs are written.

Wraps, Please

Dobbs likes the idea of working with Ms. Base. He's been consistently impressed by her professional competence, but his admiration for her goes far beyond that.

In fact, he can't help but notice that her voice sounds like the mellow purr of a bluepoint Siamese, that her agile hands fly over the computer keyboard like a frisky budgie, and that her hair (under fluorescent office lights) is the color of an Irish Setter.

In his daydreams he's having visions of an Entity-Relationship chart where a table labelled "Dobbs" and a table labelled "Ms. Base" are joined one-to-one.

In our next and final episode, we'll cover data normalization, tuning, and many other exciting subjects. And, we'll try to answer these ongoing questions:

On what data does Dobbs base his attraction to Dee? Is the relationship reciprocal, or will Dee turn the tables on him?

Will Dobbs' business ever get with the program?

And, for heaven's sake, WHAT ABOUT NAOMI?

■■■

Figure 6 - Key Assignments

VISIT/PROCEDURE TABLE

Primary key:	None
Foreign key(s):	Visit Date + Pet Name + Client ID + Procedure Name

VISIT/MEDICATION TABLE

Primary key:	None
Foreign key(s):	Visit Date + Pet Name + Client ID + Medication Name

Figure 7 - Data Required On Billing Statement

Bill heading:
  Client name, address, city, state, zip
Account previous balance
Current charges:
  Procedures performed on and medications dispensed to client's pets since last bill, including the name of each procedure or medication, the date it was administered, and the fee charged.
Total of current charges
Current payments:
  Date and amount of payments made by client since last bill.
Total of current payments
New account balance

Magic In The Real World:

By Bruce Eckel
EISYS Consulting
1009 N 36th St
Seattle, WA 98103

Digital-to-Analog Conversion

Real-world computing, take 4. In Micro C, #35, Bruce tackled A To D (Analog to Digital for the uninitiated). And it seems only natural now for him to go the other way. If D To A seems a bit too technical, you might want to review Bruce's lead-up pieces in #33, #34, and #35.

Throughout my real-world series, I've tried to give you the tools you'll need to build a control system. In short, we measure one set of values, and a control system uses them to induce changes in another set of values. All of this, of course, happens in the real world.

In recent issues of Micro C, we've examined ways to make binary changes for stepper motors, and measured analog data (with analog-to-digital converters). This time, I'll show you how to hook up a digital-to-analog converter (DAC), which will allow you to make analog changes.

In subsequent articles, I'll talk about:

- Acquiring binary data when we only care if it's above or below a threshold,
- Designing stable control algorithms, and
- Implementing a control system.

Numbers To Currents

A DAC turns a number into a current. In an 8-bit DAC (they also come in 10-, 12-, 14-, and 16-bit versions), the number 255 (eight bits of all ones) causes the DAC to output its highest value; the number zero (eight bits all zeros) causes it to output its lowest.

To use a DAC you'll need to:

(1) Establish what the highest output current value will be,

(2) Turn the current into a voltage and, when necessary,

(3) Give the voltage enough punch to drive the device you're interfacing to.

Connecting The Pins

Together, Figures 1A and 1B give a mythical view of how the DAC-0800 (a common DAC) works. *(Editor's note: Do not confuse the common DAC with a Yellow-Breasted Ruffled DAC.)* Anyway, this should give you enough of a feel for the chip to connect it properly.

The two sections are the reference amplifier (Figure 1A), which operates the "magic current controller" (that isn't really its technical name; I just made it up), and the "ladder" (see Figure 1B) which takes the bits you put at the digital inputs to the chip and, aided by the magic current controller, creates an output current, Iout, which is proportional to your byte.

The Reference Amplifier

The operational amplifier (the triangle in Figure 1A; see "Real World," issue #35) has three important rules:

(1) Its output (pointy part on the right) changes to make the inputs (marked + and -) the same; i.e., no voltage difference between them,

(2) Its inputs don't draw any current, and

(3) If it has to deliver an output voltage too close to the supply rails (V+ and V- in Figure 1A), it gets uncomfortable and doesn't act right.

The "feedback path" of the op-amp (a path from the output back to one of the inputs, which enables the input to do a reality check on the output) has the magic current controller in it.

The op-amp will change its output at point A of the magic current controller until point B puts out the voltage which makes the two inputs the same. Since the "-" input (Vref- at pin 15) is held at zero volts, point B, which is connected to the "+" input, must change until it, too, is at zero volts.

The current which flows into pin 14 under these conditions is the reference current Iref: the current which will be sucked into Iout on Figure 1B when you give the DAC a byte of all ones (also the current sucked into Iout\ with a byte of all zeroes).

Since the "+" input of the op-amp will always be held at zero, we need only select Vref+ and Rref to get the desired Iref - 2 mA (National Semiconductor linear databook spec).

Choose Vref+ to be five volts, since we'll probably have that lying about. The voltage difference across Rref is then 5V - 0V = 5V. Using Ohm's Law, R = V/I = 5V/2mA = 2.5 kOhms.

We might be tempted to set V- to 0V and try for just one five-volt power supply, but the op-amp wouldn't have enough elbow room (rule 3). It has to be a negative supply.

There are two more mysteries on Figure 1A: the so-called "bypass" capacitors and the compensation capacitor.

Just Pass Me By

Anything which contains digital circuitry has two phases: (1) waiting quietly for a change, and (2) madly making the change.

In the quiet phase, everything is nice and smooth, and the power supply only has to deliver a small, steady trickle of current. When things are changing, however, the chip makes sudden demands for large gulps of current from the supply.

The little stream of current from the power supply is briefly dried up while the chip is changing. Downstream, other chips attached to the same power supply rail see waves in the stream from these brief drainages - these waves are noise spikes in the power supply.

To prevent these spikes, we put a capacitor on the power supply pin right next to the chip. The capacitor acts like a big tub which fills up from the stream.

When the chip demands a big gulp, it takes it from the tub, which is temporarily depleted but soon fills up again. The stream can just pass right by without being affected, so no noisy waves reach the other chips.

Bypass capacitor values should be between 0.01 uF and 0.1 uF. There's a bypass capacitor on both the V+ and V- pins.

Preventing Oscillation

Remember the old "Tom and Jerry" cartoons? In one memorable scene Tom, the cat, grabs an axe and chases Jerry, who stands on something very solid, like a stove or an anvil, and jumps out of the way at the last second. The axe goes from full swing to a standstill - a change so abrupt that the vibrations work their way out of the handle and make Tom shake all over. That's oscillation.

An op-amp, too, can change so quickly that it oscillates. To prevent this, we put a capacitor on the compensation pin; it's similar to the bypass capacitor since they both slow down the rate at which a voltage can change.

The bypass cap slows down the rate the power supply voltage can change, and the compensation cap slows down the rate the output of the op-amp can change.

You may recall from issue #34 that the common and easy-to-use LM324 op-amp doesn't have a pin for compensation. If they omitted it there, why couldn't they keep it easy and omit it here, too? Well, in a DAC application like this one, where you put a byte in and expect a known, constant voltage value to come out, the value of the compensation capacitor is 0.01 uF (from the data book) and they might as well have put it on the chip, like the LM324.

But the DAC can also be used in a "multiplying" configuration, where you put an analog signal into the Vref pin, and the Iref will be that signal amplified or attenuated according to the input byte. (Aha! A digital volume control for my stereo!)

(continued next page)

Figure 1A - Reference Amplifier

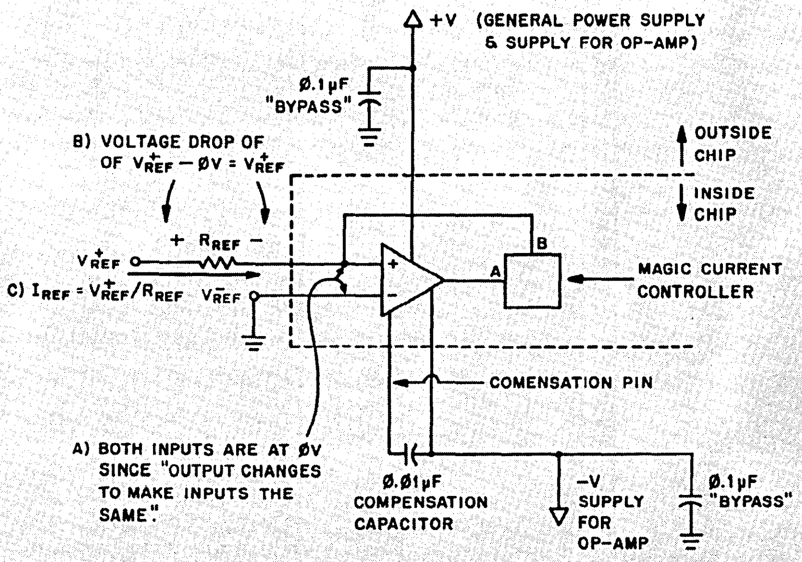
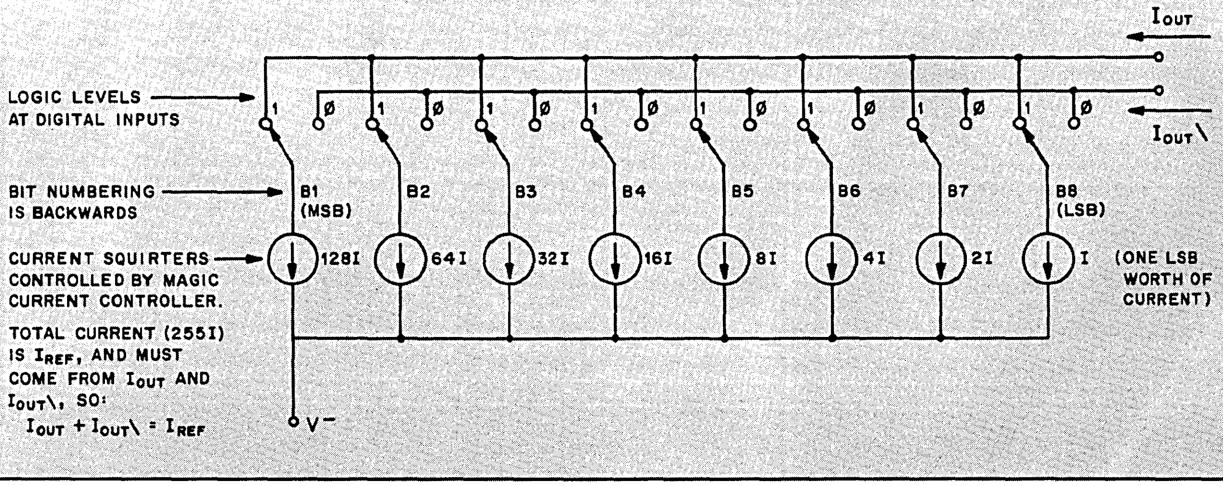

+V (GENERAL POWER SUPPLY & SUPPLY FOR OP-AMP)
Ø.1µF "BYPASS"
B) VOLTAGE DROP OF OF $V_{REF}^{+} - ØV = V_{REF}^{+}$
OUTSIDE CHIP
INSIDE CHIP
+ R_{REF} −
V_{REF}^{+}
C) $I_{REF} = V_{REF}^{+}/R_{REF}$
V_{REF}^{-}
B
A
MAGIC CURRENT CONTROLLER
COMENSATION PIN
A) BOTH INPUTS ARE AT ØV SINCE "OUTPUT CHANGES TO MAKE INPUTS THE SAME".
Ø.Ø1µF COMPENSATION CAPACITOR
−V SUPPLY FOR OP-AMP
Ø.1µF "BYPASS"

Figure 1B - Ladder Of Reference Amplifiers


I_{OUT}
LOGIC LEVELS AT DIGITAL INPUTS
1 Ø 1 Ø 1 Ø 1 Ø 1 Ø 1 Ø 1 Ø 1 Ø
$I_{OUT}\backslash$
BIT NUMBERING IS BACKWARDS
B1 (MSB)
B2
B3
B4
B5
B6
B7
B8 (LSB)
CURRENT SQUIRTERS CONTROLLED BY MAGIC CURRENT CONTROLLER.
128I
64I
32I
16I
8I
4I
2I
I (ONE LSB WORTH OF CURRENT)
TOTAL CURRENT (255I) IS I_{REF}, AND MUST COME FROM I_{OUT} AND $I_{OUT}\backslash$, SO:
$I_{OUT} + I_{OUT}\backslash = I_{REF}$
V−

(continued from page 25)

In this case, a 0.01 uF compensation cap will damp out signals we want going through the system, so we need to use a much smaller one (in the picoFarads - pF: 10^-12 Farads). Thus, the compensation pin needs to be brought out so we have the choice.

The Ladder

When I was a physics student, our professors used the phrase "waved their hands" when they didn't want to go into the gory details of something. It was supposed to be reminiscent of conjurers, who wrote down some equations, "waved their hands," and waited for the right answer to magically pop out.

We students felt this was an inadequate description of the hand motion which occurred (small breezes were often induced), so we called it "fanning." Well, I'm about to get briefly airborne.

The switches on the ladder (Figure 1B) are controlled by the binary inputs of the DAC. For some reason, they labeled them backwards: the most-significant bit (MSB - what we usually think of as bit 7) is B1 and the least-significant bit (LSB - bit 0) is B8. When we write 0x80 to the DAC, B1 is moved to the "1" position. If we write 0x01, B8 is moved to the "1" position.

The "current squirters" are the whole reason we had to go through all that stuff with the magic current controller. They're regulated by the MCC. Each squirter pulls a current proportional to its bit position.

The combined currents of all the squirters with "1" bit values are drawn through the Iout pin. The combined currents of all the squirters with "0" bit values are drawn through the complementary Iout\ pin (usually a bar overhead, but "\" is the best I can do). The sum of the currents from the Iout pin and Iout\ pin always adds to the reference current, Iref.

The DAC allows us to step through 256 current values from Iout = 0 to Iout = Iref (in our case, 0 to 2 mA).

This is really what the DAC does rather than how it does it, but it gives us enough to connect it correctly. There are only two more details: the input bias current of the op-amp, and the logic level selection pin Vlc (pin 1).

A Little White Lie

Op-amp rule two says that the inputs don't draw any current. In many instances this is a fine approximation, but sometimes (like here) we have to face reality.

There are tiny currents (called input bias currents) flowing into the inputs which can cause slight errors. To neutralize the voltage error caused by the bias current flowing through Rref into Vref+, we put an identical resistor at Vref- (see Figure 3).

The Price of Versatility...

...is that everything has to be configured. The DAC-0800 can talk to just about any kind of logic: TTL, CMOS, etc. Pin 1 decides which voltage level at inputs B1-B8 will mean a change from logic 0 to logic 1. Fortunately, hooking up TTL is very simple: just ground pin 1.

(continued on page 28)

Figure 2 - Translating Current Into A Voltage

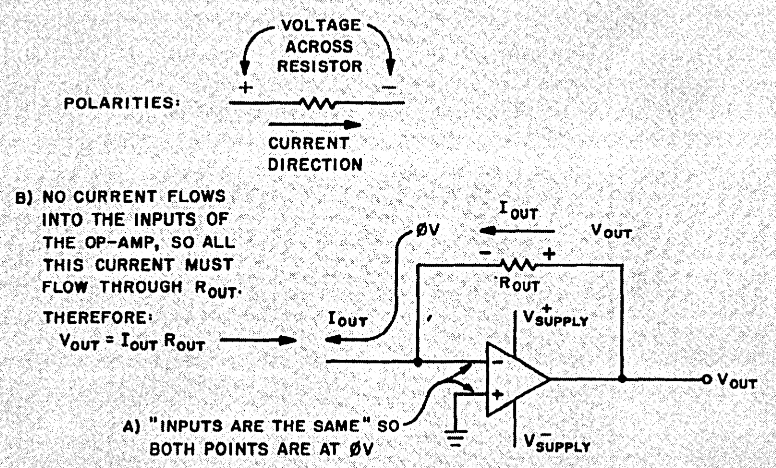

VOLTAGE ACROSS RESISTOR
POLARITIES:
+
−
CURRENT DIRECTION
B) NO CURRENT FLOWS INTO THE INPUTS OF THE OP-AMP, SO ALL THIS CURRENT MUST FLOW THROUGH R_{OUT}.
THEREFORE: $V_{OUT} = I_{OUT} R_{OUT}$
I_{OUT}
ØV
V_{OUT}
R_{OUT}
I_{OUT}
V^+_{SUPPLY}
V^-_{SUPPLY}
V_{OUT}
A) "INPUTS ARE THE SAME" SO BOTH POINTS ARE AT ØV

Figure 2A - Barefoot Voltage Translator

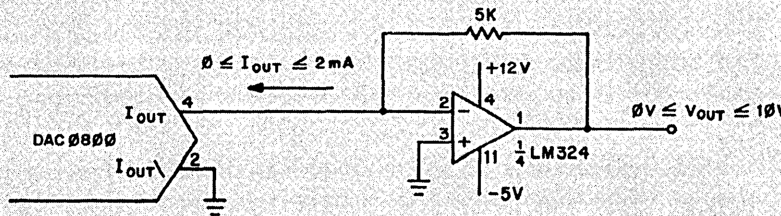


Figure 2B - Voltage Translator With Help

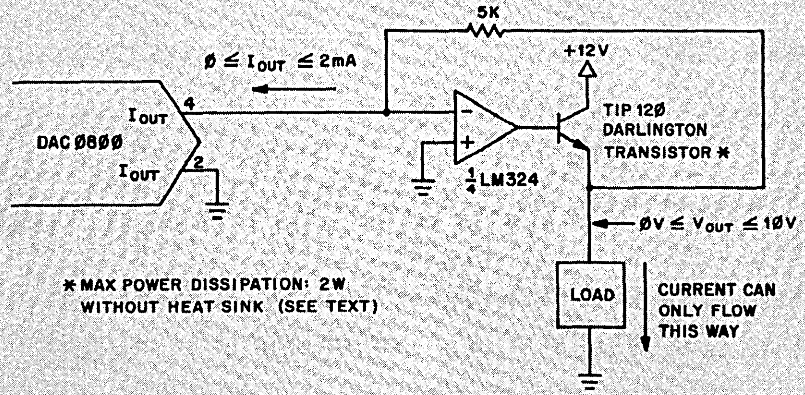

5K
$Ø \leq I_{OUT} \leq 2mA$
+12V
DAC Ø8ØØ
I_{OUT}
I_{OUT}
4
2
$\frac{1}{4}$ LM324
TIP 12Ø DARLINGTON TRANSISTOR *
$ØV \leq V_{OUT} \leq 1ØV$
LOAD
CURRENT CAN ONLY FLOW THIS WAY
* MAX POWER DISSIPATION: 2W WITHOUT HEAT SINK (SEE TEXT)

Figure 3A - Connecting A DAC To A Speaker

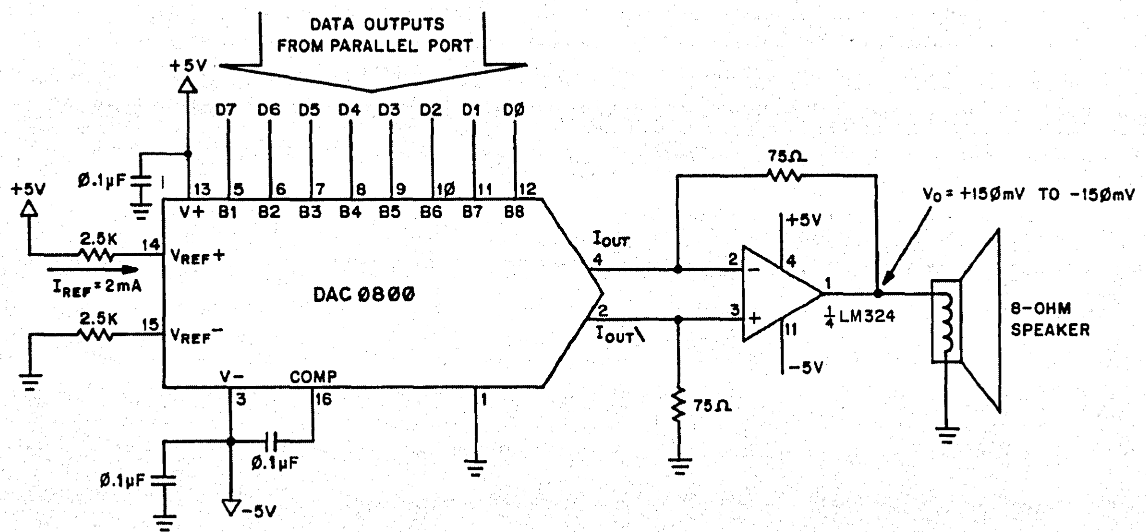
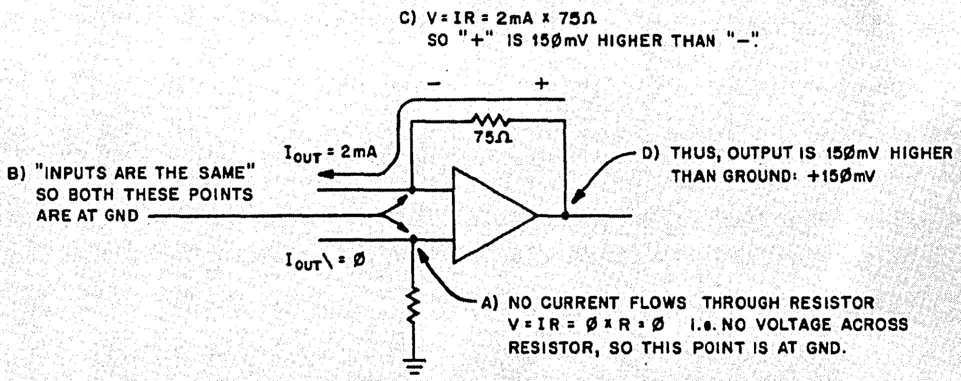
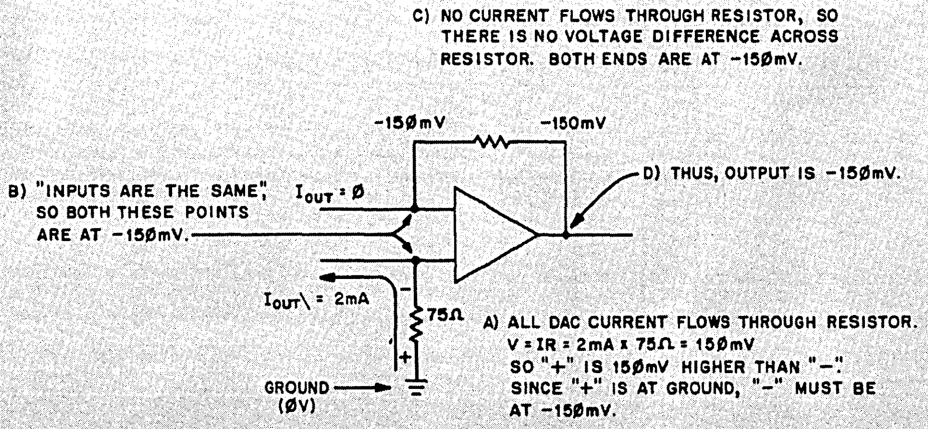


Figure 3B - Outputting A Byte Of "Ones"



C) V = IR = 2mA x 75Ω SO "+" IS 15ØmV HIGHER THAN "−".
− +
75Ω
IOUT = 2mA
B) "INPUTS ARE THE SAME" SO BOTH THESE POINTS ARE AT GND
D) THUS, OUTPUT IS 15ØmV HIGHER THAN GROUND: +15ØmV
IOUT\ = Ø
A) NO CURRENT FLOWS THROUGH RESISTOR V = IR = Ø x R = Ø i.e. NO VOLTAGE ACROSS RESISTOR, SO THIS POINT IS AT GND.

Figure 3C - Outputting A Byte Of "Zeros"



C) NO CURRENT FLOWS THROUGH RESISTOR, SO THERE IS NO VOLTAGE DIFFERENCE ACROSS RESISTOR. BOTH ENDS ARE AT −15ØmV.
−15ØmV −150mV
B) "INPUTS ARE THE SAME", SO BOTH THESE POINTS ARE AT −15ØmV.
IOUT = Ø
D) THUS, OUTPUT IS −15ØmV.
IOUT\ = 2mA
75Ω
A) ALL DAC CURRENT FLOWS THROUGH RESISTOR. V = IR = 2mA x 75Ω = 15ØmV SO "+" IS 15ØmV HIGHER THAN "−". SINCE "+" IS AT GROUND, "−" MUST BE AT −15ØmV.
GROUND (ØV)

(continued from page 27)

From Current To Voltage

The best way to turn our current into a usable voltage is with an op-amp. Figure 2 shows two methods for doing this. Figure 2A uses the output of an op-amp directly, and has the advantage of being able to both source and sink (exhale and inhale) current, but only in small amounts (source 40 mA, sink 20 mA).

For heftier applications, Figure 2B shows a TIP-120 Darlington transistor in the feedback loop, which passes much more current, but only in one direction - this is suitable for driving small DC motors. Both circuits are good examples of op-amp applications.

In Figure 2A, we have Iout from the DAC between 0 and 2 mA, and we want (as an example) Vout between 0 and 10 volts. The feedback resistor Rout lets one of the input terminals see what is going on at the output. All we need do is apply the three op-amp rules to decide which values to use.

Rule 2 says that no current flows into the inputs. That means all of Iout must come through the resistor Rout, since none will come out of the "-" terminal.

Rule 1 says "the output changes to make the inputs the same." Since the "+" input is tied to ground, the output will change so the "-" input will also stay at ground.

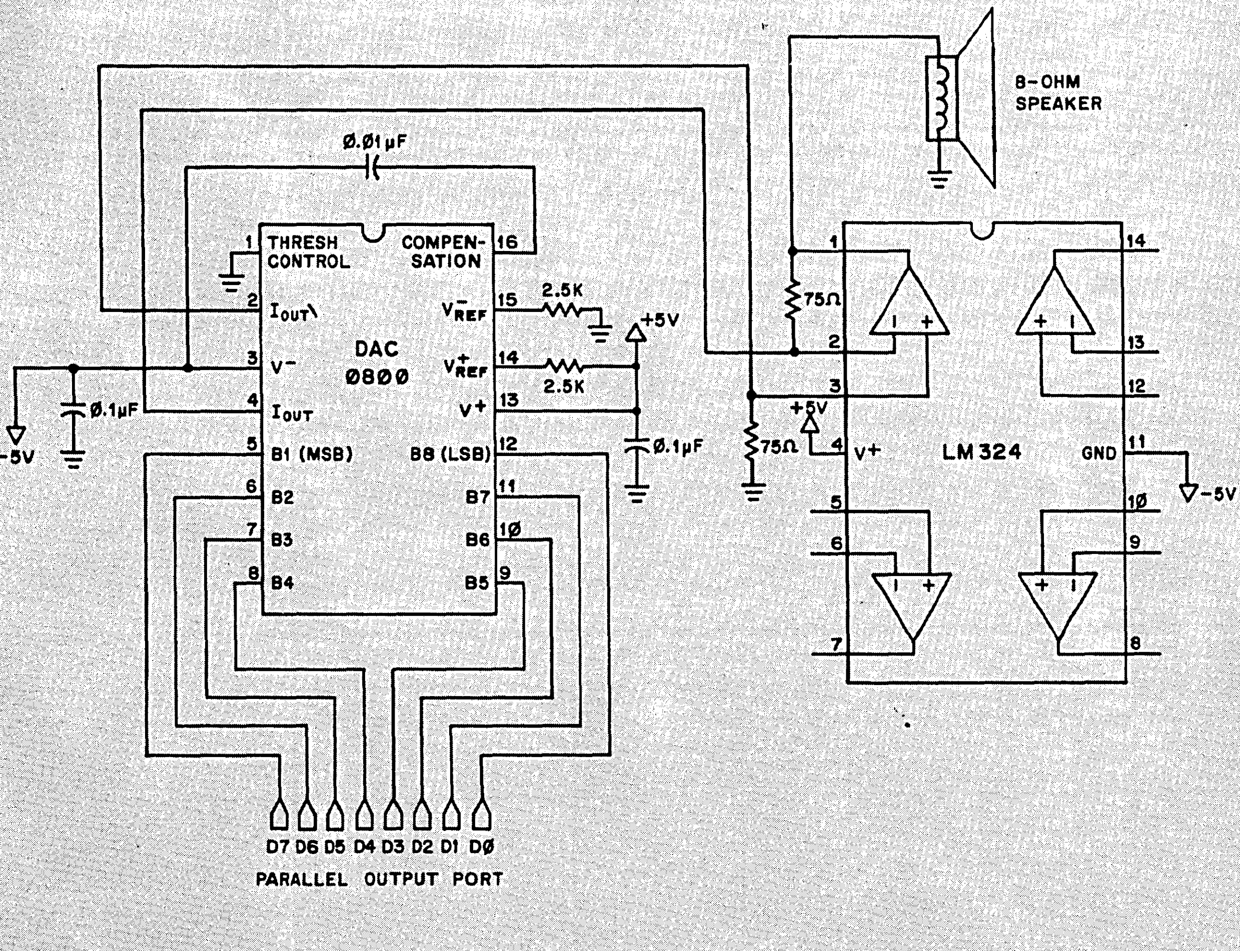
It's important to be aware of the labeling convention for voltages and currents. When current flows through a resistor, it goes from the plus end to the minus end. Alternately, if you know the direction of current flow, put the "-" at the arrow head and the "+" at the tail. Either way, "conventional" current flows from plus to minus.

In our example, the current Iout is flowing away from the op-amp and towards the DAC. All the current flowing into the DAC must flow through Rout.

Since the "-" end of Rout will always stay at 0 volts, the "+" end must be 0 volts + Iout x Rout, so Vout = Iout x Rout. Since Iout goes from 0 to 2 mA and we want Vout to go from 0 to 10 volts, Rout = Vout/Iout = 10V/2mA = 5 kOhms.

Rule 3 says the output of the op-amp must not be required to get too close to either of its power supply rails or it won't work the way we want it to (it won't put out enough current). The only limitation to the LM324 is it can't handle more than 30V across its rails

Figure 4 - Wiring Diagram Of Figure 3



(for example, +15V and -15V supplies).

Since we're already using -5V, we can use that for the negative supply (to get the output down to 0). To get the output to +10V, use a +12V positive supply. For $5 at Radio Shack, you can get a "wall wart" power supply which gives +5, -5 and +12 (Catalog #277-1022).

The only mystery left is whether to hook the feedback connection to the "+" or "-" input on the op-amp. Here's how you choose: if the feedback is connected to the "+" terminal, the op-amp will respond to an increase in voltage at the "+" (non-inverting) terminal by increasing the output voltage. If feedback goes to the "-" (inverting) terminal, an increase at the "-" terminal will cause a decrease in output voltage.

To make our example a little clearer, let's say the output voltage is 5V. This means Iout must be 1 mA. When Iout increases, we want Vout to increase. But when Iout increases, the voltage across Rout increases, which tries to force the input terminal *down.* This means we want the output voltage to go up when the input voltage goes down, so we use the inverting ("-") terminal.

Adding Some Punch

Figure 2B is the same as 2A except a high-current Darlington transistor has been placed inside the feedback loop (i.e., the feedback is taken from the output of the Darlington instead of directly from the output of the op-amp).

The beauty of this circuit is that the transistor is normally a non-linear device, which causes all kinds of headaches. But putting it inside the feedback loop linearizes it, since the op-amp doesn't care what kind of devices are in its feedback loop - it just changes its output until its inputs are the same! So when Iout is 2 mA, the op-amp changes its output until Vout = 10 volts - no questions asked.

The TIP-120 Darlington transistor is a cheap, common workhorse. It's useful for manipulating large amounts of current with either analog OR digital signals (for digital signals, see the "Real" articles in issues #32 & #33). It's a power device, and to keep from blowing it up, we must understand what power is, where it comes from, and where it goes.

Power

Power is the rate of movement of energy. Electrical power is voltage x current. Mechanical power is force x velocity. Thermal power is heat flow. Solar power is flow of sunlight. Any form of energy can be described in terms of power if its rate of flow is known.

We can change one form of energy into another. If we use our Darlington to drive a motor, we're changing electrical to mechanical energy.

But these changes always cost something - some power is lost in the conversion. That power shows up as energy in its lowest form: heat.

If this heat doesn't flow from the place it's generated, the temperature will rise until it does. If the temperature rises too much in an electronic device, that device may be damaged. We can increase the flow of heat away from a device by adding a metal "heat sink" (air is a relatively poor conductor).

In a circuit, power comes from a power supply. These are often assumed (in beginning electronics courses) to be magic machines which will put out an infinite amount of current. No place for that here. Real power supplies have limits, and when pushed to those limits, they react in different ways.

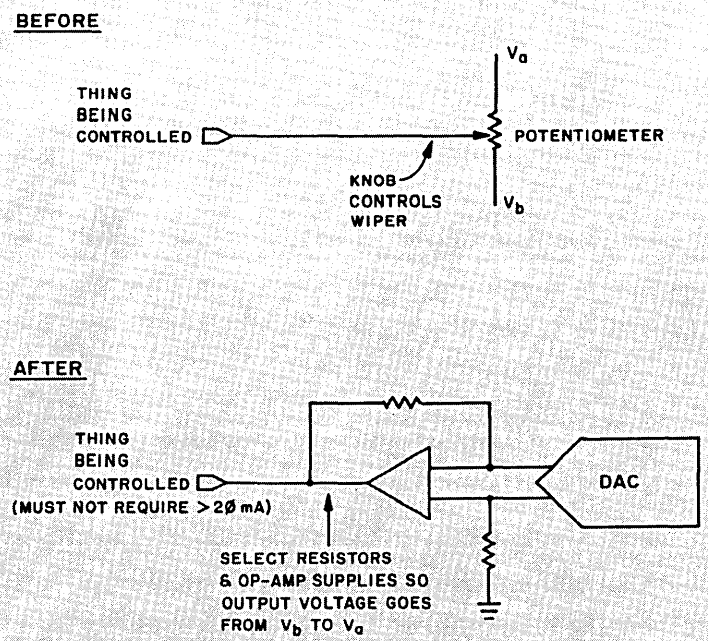
If, for instance, you short the outputs of a supply together (demanding the most current it can provide), some supplies will blow a fuse. Others may blow an expensive internal part. The kind I prefer will hum loudly, get hot, put out the most they can and keep on working.

Most "wall warts," like the one I mentioned from Radio Shack, are of this type (being thrifty, I haven't actually tried shorting the outputs of mine, but you're welcome to). Their transformers are wound with fine, high-resistance wire which just won't pass enough current to do damage. As I'm often the "idiot" referred to in "idiot proof," I find this very convenient.

The Radio Shack wall wart will put out 0 - 300 mA at +12V DC. If we ask it to put out all 300 mA, that's P = V x I = 12V x 0.3 A = 3.6 watts going into the circuit. All that power must either be changed into some other form of energy flow (such as torque x rpm from a motor) or dissipated as heat.

The circuit of Figure 2B shows the output voltage across the load going

Figure 5 - Replacing A Pot With A DAC


BEFORE
V_a
THING BEING CONTROLLED
POTENTIOMETER
KNOB CONTROLS WIPER
V_b
AFTER
THING BEING CONTROLLED
(MUST NOT REQUIRE >20 mA)
DAC
SELECT RESISTORS & OP-AMP SUPPLIES SO OUTPUT VOLTAGE GOES FROM V_b TO V_a

(continued next page)

(continued from page 29)

from 0V to 10V. Since the top of the Darlington is connected to 12V, this means there must always be at least a 2-volt drop across the Darlington (2 volts across the transistor + 10 volts across the load = 12 volts from the supply - that's Kirchoff's voltage law).

If we supply the load with 300 mA at 10 volts, that 300 mA must also pass through the transistor (which has two volts across it), so the transistor must dissipate 2 V x 0.3 A = 0.6 watts. With no heat sink, the TIP-120 will handle 2W, which is good enough.

However, if we use a motor for the load, the motor will draw current depending on how much it has to drive. In particular, if you "stall" it (grab the shaft), it will pull as much current as it can regardless of the voltage across its terminals. The worst case here would be almost 0V across the motor, almost 12V across the Darlington, and 300 mA: 12 V x 0.3 A = 3.6 watts. This would require a heat sink on the transistor.

A purely resistive load doesn't have these problems, since with a lower voltage across the load (and thus a higher one across the transistor), it will always draw less current.

Notice we are relying on the current limit of the power supply to prevent things from getting out of hand. If you have a supply with a much higher current limit, you'll have to use a bigger heat sink, and maybe a fan. Properly heat sunk (i.e., immersed in liquid refrigerant), the TIP-120 will handle 65 watts, but I find it easiest to design the circuit so you don't have to use any heat sink at all (if you're going to use a motor, don't stall it).

All this gives you an idea of the trade-offs: if you want a big one of these, you need a big one of those. If you change this, you need to change that. This is the juggling act called engineering.

For Example, Driving A Speaker

We can hear what the DAC is capable of by hooking it up to a speaker and driving it with some waveforms. Figure 3 shows the diagram and Figure 4 shows the physical connection. The common 8 ohm speaker is the type used in transistor radios.

Figure 6 - Musical Pascal Program

```
program dacsound;
{ Generates different tones through the DAC.
These are sine waves, but you can generate different types of
waves to see what they sound like. Turbo generates an array of
points which are passed to an assembly language program for quick,
consistent output.
  Passing byte values to the assembly routine worked fine, but I
had a lot of trouble passing anything else, so I declared "time"
and "wve_adr" at absolute addresses so I could just pluck the
values with assembly language. To determine the absolute addresses,
I looked at the Turbo compiler "free" message.

Note the length of the tone is controlled by the "time" variable
(which MUST be reloaded before each waveout call) AND the length
of the wave itself, so wave2 with a time of 8 has twice the duration
of wave1 with a time of 8.




Next time I'll be using a PC. }

const
        A_CONTROL: byte = $22;              {PIO address for KAYPRO 84s }
        A_DATA: byte = $20;                 {See issue #34 }
        MODE : byte = $0f;                  {0000 1111  mode 0 = output}
        INT : byte = $07;                   {0000 0111 ints disabled}

        count1 : byte = 50;                 { These are arbitrary; they don't }
        count2 : byte = 100;                { represent any particular tones. }
        count3 : byte = 200;                { (All those piano lessons wasted!)}
type
        wave = array[0..255] of byte;
var



        wave1,wave2,wave3 : wave;
        time : byte absolute $D000;
        wve_adr : integer absolute $D010;
        count : byte;

procedure make_waves(var wavex : wave; count: byte);
        {fills the array wavex with a sine wave which is count bytes long}
var
        radians : real;
        curve,i : integer;
begin
writeln('making waves ~~~~~~~~~~~');
radians := 0;
for i:= 0 to count do
        begin
                curve := 127 + round(127 * cos(radians));
                { 0 volts at the 8-bit DAC occurs with a byte value
                of 127, so the wave needs peak values of 127 with an
                offset of 127. }
                if curve  255 then curve := 255;
                if curve  0 then curve := 0;
                wavex[i] := byte(curve);
                radians := radians + (2 * 3.14159)/count;
        end;
end; {make_waves}
```

(Figure 6 continued next page)

Figure 6 - Continued

```
procedure waveout(COUNT: byte; PRT: byte);
        {Outputs bytes starting from memory location WVE_ADR to
        WVE_ADR + COUNT to port PRT. Repeats process TIME cycles.
        The value in DE is the number of cycles for each count in
        TIME. The OTIR instruction does the outputs. }

{$A+}
begin
inline(
$3A/PRT/                        { LD A,(PRT) }
$4F/                            { LD C,A }
$DD/$21/$00/$D0/                { LD IX,TIME }
$11/$FF/$00/                    { LOOP1:LD DE,00FFH }
$2A/$10/$D0/                    { LOOP2:LD HL,(WVE_ADR) }
$3A/COUNT/                      { LD A,(COUNT) }
$47/                            { LD      B,A }
$ED/$B3/                        { OTIR }
$1B/                            { DEC     DE }
$7A/                            { LD      A,D }
$FE/$00/                        { CP      0 }
$20/$F1/                        { JR      NZ,LOOP2 }
$7B/                            { LD      A,E }
$FE/$00/                        { CP      0 }
$20/$EC/                        { JR      NZ,LOOP2 }
$DD/$35/$00/                    { DEC     (IX) }
$20/$E4                         { JR      NZ,LOOP1 }
);
end; {Waveout}


procedure tone(var wavex : wave; _time : byte; count: byte; pause: integer);
begin
        { set global variables }
        wve_adr := addr(wavex[0]);
        time := _time;

        waveout(count,A_DATA);
        delay(pause);
end; { tone }

begin
        port[A_CONTROL]:= MODE;  {initialize the pio}
        port[A_CONTROL]:= INT;

        make_waves(wave1,count1);
        make_waves(wave2,count2);
        make_waves(wave3,count3);

while not keypressed do
        begin
                tone(wave1,8,count1,1000);
                tone(wave2,4,count2,1000);
                tone(wave3,2,count3,1000);
        end;
end. { "The Star-Spangled Banner" is left as an excercise. }
```

End of Listing

■■■

The parallel port is only used for output, so you can use a printer port (I'm using the port in my Kaypro 2X which I installed in issue #34).

Notice - we're using the complementary current output Iout\ in this circuit. This allows us to go above and below 0V, which is the kind of waveform the speaker wants to see. Figures 3B and 3C explain how this works with a byte of all ones and a byte of all zeroes.

To select the minimum and maximum voltage output levels, we have to work backwards from the fact the op-amp will only sink 20 mA (it will source more, but we have to design for the weakest link). Twenty milliamps sunk through 8 ohms (the speaker) is -160 mV.

If we tell the op-amp to lower the voltage any more, it won't be able to draw the required current, and the output will just stick there. So we design for + and - 150 mV (for a little safety).

The accompanying Turbo Pascal program (see Figure 6) creates the points for a wave and stores them in an array. This array is then sent a point at a time to the DAC.

Controlling Another Box

If you're tired of turning knobs by hand, a DAC can also be used to control a device which has a potentiometer on it (variable resistor - like a volume control). One of my early designs controlled a 15 horsepower AC motor this way. Someone else had done all the heavy-duty electronics, but they forced me to turn a dial to set the motor speed. All I had to do was overcome their pot with a DAC. Figure 5 shows how to do this.

If you're controlling something potentially dangerous, you'll want to stand back while you test your software. Then multiply your problem by a billion (at least), and (you guessed it) you've got Star Wars!

That's it. I'm out of here.

■■■

By David Thompson

Shareware Authors Talk About Their Experiences

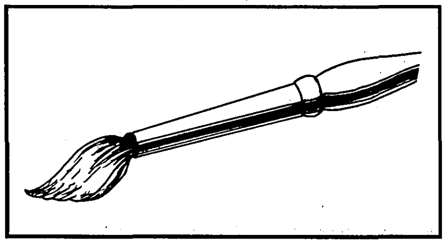
I must admit I had some misconceptions about shareware. I knew that some authors were doing well, but I wouldn't have guessed how well. Nor would I have guessed what it takes to be successful.

It was with some reservations (plane, hotel, & personal) that I attended the first annual PD/Shareware Convention in Houston, Texas. There were two reasons for the meeting. First, to get authors and distributors (SYSOPs and disk copy services) together. Second, to create a "shareware" organization.

"Sure," I thought as I looked over the single-page conference announcement, "half a dozen starving software writers looking for respect."

The event took place in a hotel located just 20 minutes ($34 cab fare) from Houston International Airport. Our meeting room held 100; it was only half full of people, but the energy level was tremendous.

Nelson Ford, librarian for the Houston Area League of PC Users (HAL-PC), started things off by having each attendee say something about himself. Then Nelson introduced the first speaker.



PC Key Draw

Ed Kadera began the day's program by discussing his experiences writing and marketing PC Key Draw.

Ed had wanted a way to create graphics on his own PC. There was nothing available at the time, so he wrote PC Key Draw. Once it was working he thought he'd try getting it out into the shareware marketplace.

"I sent copies of Key Draw to several public domain organizations. I figured that shareware meant just putting it out there and money would come back. But it didn't work that way.

"Then I decided to improve the program: add a clock on the screen, new graphics features, and so I released more versions. Once I got to Version 2.1, things really started rolling.

"The main complaint I heard was that the program was too hard to learn. This is probably because I write for myself. I like powerful programs, and powerful programs are always harder to learn than simple ones.

"I released version 3.0 last fall and started advertising. Since then there's been a big jump in registrations. Most people try it first and then register. Also, advertising gives me credibility and boosts orders. As of late February, I'd already exceeded sales for the first half of last year.

"There is a point where you sit back. I have to admit my old job designing motion compensators paid more, but I love what I'm doing now, and one of these days I'd like to do a newsletter. I work 40 hours a week on Key Draw, and I'll never go back to a 9 to 5 job."

About the future: Ed mentioned that Key Draw currently works with CGA cards only. He gets an average of three requests a day for a Hercules version, so he's considering that as a new project.

Automenu

Marshall McGee followed with a description of his automenu program.

In 1983 he found himself training people to use MS-DOS. The trainees didn't have the slightest idea how to move files around or change directories, so he wrote Automenu.

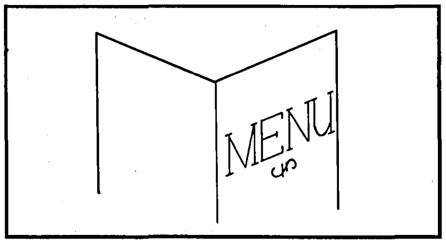
In July 1984 he put a copy of version 1.6 on Compuserve. He immediately received requests for new features. Soon after he released version 2.0, he left college to work on the program full time (instead of writing programs in BASIC for $3.35 per hour).

In June 1985 he was receiving a check ($40) a day. Then he released 3.01.

"3.01 had more features and more documentation (my sister-in-law corrected the spelling). My RBBS filled up, registrations came flooding in. The program finally had just about everything people wanted. I called every bulletin board I could find in the U.S. and uploaded it.

"Now I'm a member of 50 user groups. I get all their address changes, see all the reviews, and personally thank the reviewers when they write about Automenu. None of the newsletter authors had ever received a call from a shareware author.

"The program did really well in 1985. I made more money than anyone thought I could and I'm still two



quarters short of graduating.

"I have 15,000 registered owners, 500 to 700 registrations per day, ads for Automenu in lots of magazines, and I'm buying a quarter-page ad in 50 club newsletters for three months for $1500. Total.

"I've cleaned up my documentation, started using blue disks, and I've added fancy printed labels.

"I write out the word 'copyright.' If you have a shareware program, you should do everything you can to protect it. Send in the $10 to copyright it. Pay the $175 to register the name. I even added a custom PVC clamshell case because I wanted to make my product as impressive as possible. I also put a barcode on the outside of the case. The barcode doesn't mean anything, but it looks professional.

"Also, the package says 'Made in USA.' I've enclosed a prepaid registration card and a comment card. I've even put my picture in the manual. How many products do you see that have the author's name and picture on them?

"Tech support: I handle that, but I don't let on it's me. If I told them they were talking to Marshall McGee, they'd think we were a small company.

"As a small business you can be very aggressive because you have no costs. One company came to me and said they would be buying a large quantity of someone else's menu program because they'd already invested in 100 copies. They wanted everyone in the office to have the same program. But they said they liked mine better. So I gave them 100 copies of Automenu and then I came in with a bid for additional copies that was half my competitor's."

Marshall noted that he would sell anything, site licenses, limited site licenses, complete packages (disk, box, manual), economy packages (disk and manual only), disk only, manual only, label only, and so on.

The US Government couldn't accept its 100 copies on one disk so he sent them one disk and 100 labels. He also said that quantity purchases by corporations and agencies made up a significant portion of his sales.

PC Outline

John Friend described himself as a self-taught assembly language programmer who had spent every spare minute at a local computer store playing with software. One program that caught his attention was Thinktank. He liked the idea but didn't like the way they did it.

Though he'd never before written a commercial application, he decided to create his own memory-resident notepad editor/manager.

"I wrote the editor first, then created the outline structure. It was 16 hours per day, full time. By May 1985, I had a working outliner."

Chasing Distribution

Then he looked for a publisher, but he didn't know which companies to contact.

"I knew the big names and I'd found a book with a list of 200 software publishers. So I made up a free running demo, sent it to 50 outfits, and then sat back and waited for them to come to me.

"Two weeks later, I had received two rejections and five packages returned unopened, with notes saying they didn't accept unqualified submissions. It became obvious that I had to track down the correct person in each organization."

Three months later he was still looking for a distributor.

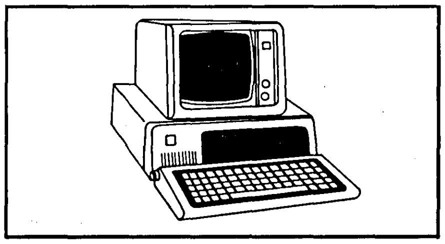
"Some were interested but not interested enough to write a contract. They kept looking for this feature or that feature. I just didn't know how to deal with them."

Then Living Videotext released a competing but less powerful product and spent lots of money promoting it.

Shareware

"I had this idea that good products would survive bad marketing. I thought about introducing it myself, but I had no experience and no money. It all added up to shareware. I wasn't being altruistic it was the only alternative."

So he cleaned up his product, finished the documentation (typesetting was a pain the first time, but now he



uses a laser printer), and then looked for ways to make his program known. "There's no question that editorial mention is by far more valuable than anything else you can do. I got hold of every computer publication I could find and made a list of the editorial people. I sent each person a press release that described what the program did, what it cost, and where to buy it. Those contacts proved to be a gold mine.

"Fortunately I wasn't doing this for my livelihood. I had $2,000 into it and the sales price was $49.95.

"My biggest mistake was not having a modem. For the first six months, people were purchasing it and then posting it on boards for me."

A Break

"PC-Magazine was finishing a review of outliners when our press release arrived. They called to say they'd include PC-Outline if I'd Federal Express them a copy."

The editor reviewed the product and called Friend to verify the facts (no mention of what the editorial comments would be).

PC-Outline was named co-editor's choice (along with Max Think).

"They were fairly forgiving in their review. The program wasn't perfect and the documentation wasn't complete, but then we were a shareware product. Magazines like telling people about software they can try for free."

He sold 150 copies in March.

Then he sold 200 during the 4-day West Coast Computer Faire. He had a sign on his booth that announced the "Editor's Choice" award and it really drew in the crowds. It also generated a lot more press contacts and, eventually, reviews in Info World and PC-Week.

"Sales grew to the point where I didn't have time to write code, and they continued to build through the summer. I had to choose between running a business and writing code. So I got in touch with Brown Bag Software and sold out to them."

Unfortunately, there were some problems during the transition. Registered owners were left unsupported while others were dunned for money they didn't owe. But John feels that the problems have been pretty well cleared up.

PC-Write

Unlike many other shareware authors, Bob Wallace had a strong programming background before begin-

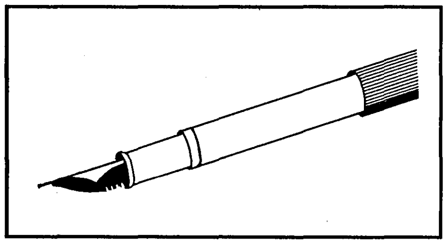
(continued next page)

(continued from page 33)

ning his first shareware program. He had a master's degree in computer science and had written a Pascal compiler for Microsoft. That was back in 1982, when Microsoft had a mere 300 employees.

After deciding he'd had enough of the corporate trip, he wrote a word processor for MS-DOS.

"I started coding in February, and



by August I released the first product. I'd spent about $5,000 to that point. December's sales were $17,000.

"The following summer I hired a support person, and by 1985 there were nine of us. That year total sales hit $750,000. Now we have 20 people and sales are running $40,000 to $50,000 per week.

"I work 5 days a week managing Quicksoft. I program on the weekends."

Marketing

"Marketing is getting your product to people. It consists of the five P's - product, place, price, promotion and positioning."

But he noted that the five P's don't mean much until you get down to specifics.

"Fifty-eight percent of our registrations come from people who got the editor from a friend. Of course we pay commissions ($25 to a registered owner for each of his friends who registers). We pay a commission on about one-sixth of the registrations.

"I also like card decks (those ubiquitous little packets of postcards). And believe it or not, you can actually make money selling your product at the West Coast Computer Faire. Even more effective than advertising is editorial mention. I work with both foreign and domestic publishers.

"We're starting to get into direct mail, group licensing, OEM licensing, and retail sales."

Support

Bob feels it's important not to get lost in program details.

"If you're really helping people do something, then you'll do well. Don't worry about the people who don't register.

"If unregistered owners call and ask, 'Will it do X?,' we always answer the question. If they ask, 'How do I do X?,' we ask them if they'd like to register. If they say fine, we get their address and answer their questions. Then we send them a letter saying, 'Hi, now it's time to register.' "

Business

"For some reason every August our receivables go way up, about $100,000 to $200,000 in receivables. We get very low on cash, so maybe this August we'll have to borrow money. It's OK to borrow money. You have to look at your ratio of liabilities to assets. You have to learn about that kind of stuff.

"Also, on the operations side, you have to make sure you're getting orders in and getting them out on time with the right stuff in them. And be sure you're checking people's credit card numbers.

"I'm president. My wife is VP of operations. Then we have our controller, a marketing manager, and a sales manager. They each have an assistant. Marketing people are expensive.

"We have four support people and one programmer. Tech support can't be done eight hours a day. Five hours is max for any one person. We have the tech people do other things to finish out their eight hours.

"Every time we hire a support person, we ask what his other specialty is going to be. That could be technical writing, beta testing, creating printer control files, etc. One does internal computer support, another does the newsletter.

"Four people answer the phone and three package the product. We run about 500-600 packages a week, about half are diskette orders and half are registrations."

Shareware

"One of the biggest problems is shareware's credibility. Some people say 'If it's free then it can't be worth anything.' Because it's so easy to start shipping stuff out as shareware, a lot of the shareware has never been tested by dealers and users. So if you've got something that's really good, it's necessary to have people evaluate it. That's why reviews are so important.

"On the other hand, the strength of shareware is the ease with which anyone can get the product out; and if the product's good, people will pay for it. I don't think we should exclude anyone from participating."

He also talked about tie-ins with other manufacturers. For instance, sellers of communications boards might include a shareware communications program. Ads for the boards would state that communications software was included free, but that the user would be asked to register if he liked it.

He also noted that software is very complex, so you really add a lot of value to the product if people can try it out before buying. The package with the most value will win.

New Versions Of PC-Write

"Version 3.0 should be released in June. It will have large-file editing, box editing, marking, multiple columns, and will lean toward the page layout sort of thing.

"We'll go on to outlining, numbering, etc. I could be programming seven days a week for years and still not get in all the things I want to do."

Selling By Stages

Bob sells PC-Write many different ways. If you purchase a manual, you get the right to use the software. Manuals are $35 paperback and $45 hardback. He says it makes sense to sell the rights to the program with the manual because the manual is much harder to reproduce.

Also, corporations love the idea of getting rights to use the program when they purchase manuals.

"When you sell to corporations, you have to sell them something, even if it's just labels."

Of course, you can purchase more than just the manual. If you want a disk and a hard-cover manual, the total is $59. Add basic tech support and it's $79. Or you can pay $89 for the whole nine-yards. (The nine-yards includes all the above, plus two free updates, a quarterly newsletter, and the $25 kickbacks if others register from your copy.)

His ads sell the $16 diskettes. The diskettes sell the registrations. It's a two-step process. He estimates that one user out of ten registers.

He's discovered there are several reasons why people register. First, they want the current version. The updates came in a close second, while the printed manual (which includes a quick reference card and a tutorial) was third. Supporting Quicksoft came in fourth.

Questions From The Crowd

He was asked if he were concerned with other companies publishing manuals for PC-Write.

"Oh, a little bit. Our defense against that would be to update the product enough so that the other manual wouldn't be any good. I hope we don't have to really deal with that. It's better that they work with us. McGraw-Hill has a shareware book and we've put a $20-off coupon in it."

Also, by arrangement, McGraw-Hill left out some significant things about the program.

Another participant asked how Bob would sell a shareware package that didn't need a manual.

"There are three important values added: the manual, phone support, and updates. The more of these there are, the more effective shareware will be. There are some kinds of software - game software, training software, and utility software where a lot of those features aren't available. I don't know how well they work without them, except that McGee has done very well with Automenu. It's just going to be more difficult to sell them.

"You might consider charging a very low price. You know, $20 and you own it. I think shareware works best when there's an evaluation cycle."

Working With Retailers

"We have something we call a 'Give it away kit.' It includes diskette labels and a little brochure that fits into the diskette. The first half of the brochure explains how to get started, the second half tells why you should register."

The dealer makes copies of the program, gets a registration number, and then gets $25 kickbacks from Quicksoft.

"As long as the copies get out there, I don't feel burned at all."

He noted that Brown Bag Software had handled PC-Write for a while (as the Brown Bag Editor), but he said he was no longer dealing with Brown Bag.

Limited Programs

The discussion then became a free-for-all as the topic moved to crippled software. Bob felt that limited or crippled versions might be an option for some types of packages.

"People can try the program in a limited sort of way, get a feel for it, and decide if they want to purchase it."

But the audience immediately jumped into the fray.

"We're Compuserve. We have 'Crippleware,' 'Hostageware,' and 'Beware.' And seriously, these are all viable marketing techniques. The author wants to get money back so he puts a time lock on it, or a gate lock. But a user doesn't want to pay long distance charges to get a 'pay-to-use' program. It creates lots of bad feelings."

Then an RBBS operator from Hawaii added, "Crippled software, it's about the only time we get negative flack from the user. The minute someone uploads a demo-style program, we just knock it off the board."

And, others brought up the point that many boards have limited space and SYSOPs are often donating their time and equipment. SYSOPs feel their boards are being abused by commercial operations, especially when the software they upload is of little value to the user.

Finally

As you can see, there is a lot to do about something in shareware. It's probably the only way that individuals can compete with the distribution and advertising might of the major software outfits.

However, the best guesstimates indicate that, at best, only one user out of ten pays for the software. But commercial firms are also guesstimating their user-to-purchaser ratio, and in some cases, they are not far from the same ratio.

Next issue we'll continue this discussion of shareware when we hear from Jim Button of ButtonWare and Sandy Schupper of Brown Bag Software.

■■■

Full Featured AT Motherboard

***fits XT or AT case!* $489**

Upgrade your XT to a real AT for about the price of an "accelerator" card

OR

build a space-saving AT from scratch.

Features:

Phoenix BIOS, 6/10 MHz operation, 1MB memory (ØK installed), VLSI technology, 8 expansion slots, Clock/ Calendar and Instructions in genuine English!

Add $75 for 1MB memory installed and tested.

Other selected components available include

• Toshiba 1.2MB floppy drive	$ 118
• Floppy/Hard drive controller	$ 194
• XT size case w/LEDS, lock, reset	$ 55
• ST4038 AT Hard drive (40ms)	$ 479
• V20-8 replacement for 8088	$12.95

Call Softside Systems day or evening at (503) 591-0870

Hardware Interrupts On The PC

By Larry Fogg
Micro C Staff

Or Who Was That Masked Bit?

While Bruce Eckel is thrashing about in hardware and John Jones is deeply immersed in Modula and Pascal, poor Larry can't decide. He showed us how to slow down our systems last issue; now he's trying to interrupt our programs.

Last issue I got around (in a roundabout way) to talking about the 8253 Programmable Interval Timer. Several other "smart," or programmable, chips live in the PC. I'll be covering each of these chips in upcoming issues, but right now it's on to Intel's Programmable Interrupt Controller (PIC), the 8259A.

Why Use Interrupts?

Microprocessors are busy critters. They need to execute code, talk to modems, disk drives, and printers, listen to the keyboard, and sometimes control processes in the real world.

These functions (and many more) must be coordinated with a minimum of mayhem to have a smooth-running system. Consider the keyboard. How can it tell the processor when a key has been pressed? One method would be to poll the keyboard periodically for input. Polling works well on large systems where terminals and printers clamor constantly for the processor's attention.

But a single-user PC keyboard needs very little hand-holding. A lot of processor time would be wasted in useless polling. It makes more sense to have the keyboard interrupt the processor when it needs some attention.

Loosely speaking, an interrupt sequence goes like this: The keyboard signals to the processor that it has a character ready. The processor stops what it's doing, pushes its flags, code segment (CS) and instruction pointer (IP) onto the stack, and executes any code necessary to process the keyboard interrupt. Finally, the flags and registers (showing where the processor was before the interrupt) get popped off the stack and the processor can take up where it left off.

Of course life isn't that simple. The 8088 only has one pin which can accept interrupts. But there are a number of devices which need its attention. And what if two interrupts occur at the same time? For these and other reasons, the PC makes use of the 8259A to control interrupts.

Inside The 8259A

Let's take a stroll through the PIC. The 8259A can control interrupts generated by eight different sources (see Figure 1). These interrupt requests enter the PIC through eight pins called IR0 - IR7 (see Figure 2). IR0 has the highest priority and IR7 the lowest.

As interrupt requests come in, the corresponding bits of a 1 byte internal register, called the Interrupt Request Register (IRR), are set. Pending interrupts then get processed in order of priority. This assumes three things - that the highest priority interrupt is not masked, that a higher priority interrupt isn't already being processed, and that the processor feels like being interrupted.

Figure 1 - Hardware Interrupts

Interrupt Number	Intr Req	Name
08	IR0	Time of Day
09	IR1	Keyboard
0A	IR2	Reserved
0B	IR3	Communications
0C	IR4	Communications
0D	IR5	Hard Disk
0E	IR6	Floppy Disk
0F	IR7	Printer

The Interrupt Mask Register (IMR) takes care of masking, or disabling, any combination of the eight interrupts. Each bit of the IMR corresponds to one of the interrupts. Masking certain interrupts proves to be very useful. For example, you can use the IMR to disable the keyboard during portions of a program's execution.

A third 1 byte register, the In Service Register (ISR), stores any interrupts currently being serviced. If there is an unmasked request which has a higher priority than any interrupts in the ISR, the PIC raises its INT pin. This pin connects directly to the 8088. It will be held high until the 8088 acknowledges the interrupt.

You can see that it's possible for an interrupt to interrupt an interrupt which has interrupted another interrupt ... No problem. All the information needed to trace the tortuous path back through all those interrupted interrupts is alive and well on the stack.

The interrupt runs into another check in the 8088. One of the 8088's flags is the Interrupt Flag (IF). The assembler instruction STI enables interrupts by setting IF to 1. There are situations (i.e., timing loops and disk sector reads) that must not be interrupted. In these situations, a CLI instruction resets IF to 0 and interrupts are disabled. With IF set, the interrupt passes its final test.

Before acknowledging the interrupt, the 8088 finishes its current task. This may be a single instruction or a pair of instructions. For example, if a MOV to stack segment (SS) or POP SS occurs, the 8088 waits until after the following instruction to recognize the interrupt.

There's a good reason for this. What if a program wants to create a new stack by changing SS and the stack pointer (SP)? Any interrupt acknowledged after the SS update, but before the SP update, will cause the flags CS and IP to be pushed into the wrong area of memory. Bye bye ...

Let's assume we've followed the good programming practice of always changing SS before SP (thus forcing the 8088 to wait for SP's value). When the current task finishes, the 8088 says, "Okay. Let's boogie!"

We Interrupt This Program ...

The 8088 boogies by pulling all of its Processor Status lines (S0 - S2) low. These active-low signals connect directly to the 8288 bus controller. The 8288, in turn, decodes S0 - S2 and pulses the Interrupt Acknowledge (INTA) line low. As soon as the PIC sees the INTA pulse, it freezes the values of its registers in preparation for resolving the current status of interrupts. It then sets the ISR bit corresponding to the highest priority non-masked interrupt in the IRR. Since the PIC is now dealing with the request, the IRR bit gets reset.

Now the 8088 initiates another INTA pulse which tells the PIC to put an 8-bit pointer onto the data bus through its D0 - D7 pins. Values for this pointer correspond to the eight hardware interrupts and range from 08h through 0Fh. The 8088 grabs the pointer off the bus, and from here on the hardware interrupt acts like a software interrupt.

The lowest 1K of PC memory is devoted to a table of 256 4-byte pointers or vectors. If interrupt 08h asks for service, the 8088 looks for the vector at memory location 0000:4*08h. This vector points to the interrupt handler code for interrupt 08h. The 8088 then jumps to and executes that code. When the code finishes, it resets the appropriate bit in the ISR with an End Of Interrupt (EOI) command. Now the 8088 can turn to either the next highest priority interrupt or whatever program it was executing before the interrupt process began.

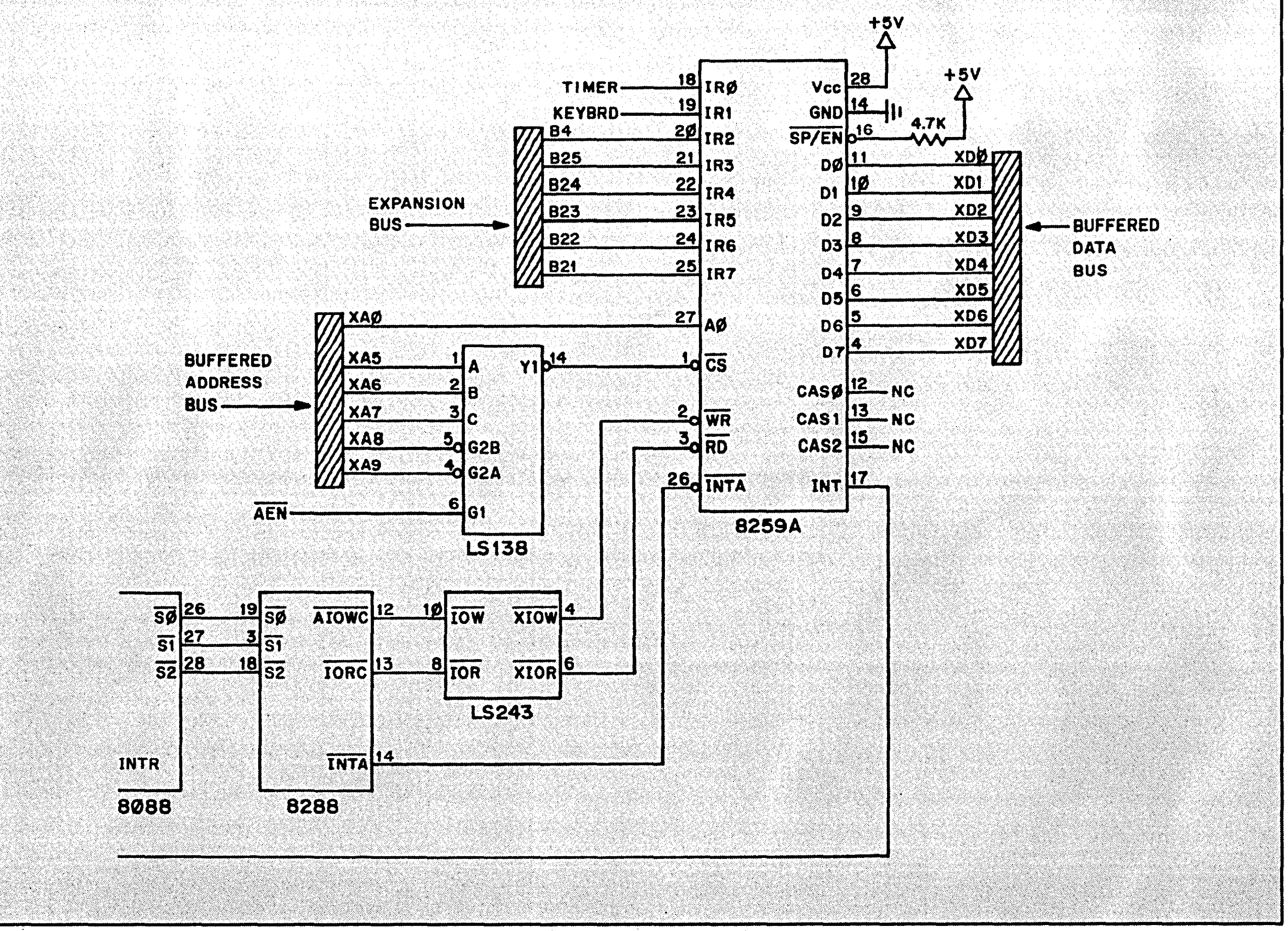
Programming The 8259A

On power up, the BIOS programs the PIC to handle interrupts in the manner discussed above. The PIC is actually a much more versatile chip. Let's go through its programming with an eye towards more of its capabilities. I'll restrict this discussion to 8086/8088 operation, although 8080/8085 mode also exists. See Intel's *Component Data Catalog* for more information.

We program the PIC with two types of commands: Initialization Command Words (ICWs) and Operation Com-

(continued next page)

Figure 2 - Interrupt Circuit



(continued from page 37)

mand Words (OCWs). Intel likes to say that "words" sent to its intelligent chips do the programming. Actually, they're bytes. Four ICWs take care of setting up the chip for operation. Any time after this initialization, three OCWs can set the various modes for handling interrupts.

The PIC uses its A0, Write (WR), and Read (RD) inputs to identify the various programming commands and access its registers. A0 ties directly to the buffered address line XA0. I'll explain its use below.

WR and RD originate in the bus controller just like INTA. The 8088 signals an I/O read or write by setting its status lines. The 8288 decodes the status lines and generates the appropriate I/O control signal according to the following table.

S0	S1	S2	Action
0	0	0	INTA
0	0	1	I/O port read
0	1	0	I/O port write

I/O control signals generated by the 8288 are driven out to the system by an LS243. Several chips listen in from the I/O address space. This space consists of a 64K area separate from RAM. The 8088 doesn't care what's there as long as it looks like RAM.

Somehow, only one of the chips that sees the control signals has to be selected. Chip Select (CS) signals come from an LS138 3-to-8 line decoder. The LS138's Y1 output supplies CS to the PIC. We want to drive CS low.

Take a look at TI's *TTL Data Book.* For a low on Y1, input A (XA5) must be high and B, C, G2A, and G2B (XA6 through XA9) must be low. This condition is met by any address whose least significant ten bits lie in the range 00 00100000b to 00 00111111b, or 20h through 3Fh. The six most significant bits of any I/O address are ignored. The PC gets away with ignoring them since it really doesn't need a full complement of 65536 innies and 65536 outies. So any I/O port read or write to an address between 20h and 3Fh will select the PIC.

Initialization

The initialization sequence begins with a write of ICW1 to port 20h. All other required ICWs must be written in sequence to port 21h.

In ICW1, only four bits have any meaning for 8086/8088 systems. A 1 in bit 4, along with a 0 on A0 (remember - A0 can be either 0 or 1 and still address the PIC), identifies the byte as ICW1. Bit 3 sets the interrupt triggering mode. Zero makes the PIC sensitive to rising edges on the interrupt request lines, while a 1 makes the request lines active high.

The PC operates with only one PIC, but PICs may also be used in a cascaded configuration. Up to eight slave PICs can attach to the interrupt request lines of a master PIC. A full complement of eight slaves gives the ability to service 64 unique interrupts. The highest priority interrupt of these 64 will be IR0 on the slave connected to the master's IR0 line.

With a little work, you could turn the PC into a powerful data acquisition or experiment/process control system. Anyway, bit 1 of ICW1 configures cascade mode (bit 1 = 0) or single PIC mode (bit 1 = 1). Finally, 1 in bit 0 means that ICW4 will be needed and 0 means it won't.

ICW2

Next the PIC expects to see ICW2 at port 21h. Bits 7 through 3 specify the location in the interrupt jump table of the highest priority interrupt vector. For example, on the PC, ICW2 = 8. So the interrupts are labeled 08h through 0Fh.

If you ever create the 64-interrupt monster made possible by cascading, each slave will have to be programmed separately. You'll also need to decode Chip Select signals for each of the slave PICs. Space in the interrupt jump table is easy to find if you don't mind bagging BASIC (right on!).

Label the existing interrupts 08h through 0Fh and the new interrupts 60h through 98h. The new interrupts will wipe out BASIC but leave the DOS interrupts alone. Do the labeling by sending an ICW2 of 08h to the first slave, 60h to the second, 68h to the third, etc. As an example, if a request comes in on IR3 of the second slave, the interrupt label becomes:

60h + 3 = 63h

ICW3

Multiple PIC systems require the use of ICW3. When programming the master PIC, each set bit in ICW3 means that the corresponding interrupt request line on the master connects to a slave rather than an interrupt source. For the slaves, bits 2 through 0 of ICW3 constitute an ID number of 0 through 7. During an interrupt process, the master places a slave ID on its CAS0 - CAS2 lines. These outputs form a cascade bus connected to all slaves. A slave is selected when it sees its ID on the bus. The PC doesn't use the CAS lines since it's a single PIC system.

ICW4

On to the last ICW. Bit 4 of ICW4 tells whether the Special Fully Nested Mode is active (bit 4 = 1) or not (bit 4 = 0). In a cascade system, the master uses this mode to allow recognition of multiple levels of interrupts within a single slave. Slaves and single PICs do not use this mode.

Bits 3 and 2 work in conjunction. If bit 3 = 0, the PIC operates in non-buffered mode and bit 2 has no meaning. In this mode the PIC's SP/EN pin becomes an input with a high designating the PIC as a master and a low making it a slave. In buffered mode (bit 3 = 1), the SP/EN pin outputs a signal which enables bus-driving buffers whenever the PIC wants to put a byte on the data bus. Also, in buffered mode bit 2 specifies master (bit 2 = 1) or slave (bit 2 = 0) status for the PIC.

Let's muddy the waters a bit. IBM's BIOS listing shows initialization of the PIC to buffered mode and slave status. Their schematic shows SP/EN tied high. Buffered mode makes sense - the PC's data bus is buffered by an LS245 octal bus transceiver. But a buffered system should use SP/EN to enable the LS245. Why is it tied high instead? And slave status in a single PIC system?

Either I'm confused or someone's lying. IBM's BIOS checks out. I disassembled the PIC initialization code in their ROM. Sure enough, the PIC is a buffered slave. I looked at SP/EN on two different clone boards. One tied directly to +5 volts. The other went to +5 volts through 7.8 or 8.6 K Ohms depending on the orientation of the VOM test leads. The difference in resistance suggests the presence of an IC between that PIC and +5 volts.

One other bit of strangeness: A high on SP/EN would make perfect sense if the PIC was programmed for non-buffered mode. In that case the high would designate the PIC as a master.

But the PIC's in buffered mode ... If anyone can shed some light on the situation, I'd appreciate a note. Until then I'll treat it as one of life's little mysteries.

Back to ICW4. The ISR bit for a given interrupt can be reset in one of two ways at the end of its interrupt service routine. Bit 1 determines which method must be used. In automatic EOI mode (bit 1 = 1), the in service bit gets reset when the PIC sees the trailing edge of the last INT pulse. The interrupt service routine must send an EOI command to the PIC when normal mode (bit 1 = 0) is used. Finally, bit 0 specifies the processor in use: a 1 for 8086/8088 systems or a 0 for 8080/8085s.

The PC BIOS initializes the PIC with the following values:

ICW1 = 13h
ICW2 = 08h
ICW3 = not used
ICW4 = 09h

Operation

After initialization, the PIC is ready for interrupts. From this point on, the three OCWs can program the PIC for various modes of operation. In normal operation, OCW1 gets the most use. However, an idea of what can be done with the other two command words might prove interesting to folks like Bruce who are into real world processes.

OCW1 controls masking in the IMR. Normally masking is done with writes to port 21h. But any write to the PIC with A0 high is interpreted as OCW1. You can prove this to yourself by altering the IMR using port 31h. It works. To examine the IMR, just read the PIC with A0 high - that's a read of either port 21h or 31h or 23h or ...

OCW2

With A0, D4, and D3 low, the PIC recognizes OCW2 (see Figure 3). Back in ICW4, we configured the EOI mode. When ICW4 sets automatic EOI mode, the ISR register automatically resets. In the normal EOI mode, two methods exist for clearing the ISR bit. The PC has a straightforward interrupt structure. Its PIC knows that the highest priority bit in the ISR is the one to be reset on EOI. Therefore a non-specific EOI command can be issued.

In some systems it's not clear from the status of the PIC which interrupt needs to be cleared during an EOI. A specific EOI must be issued by the interrupt handler. In this case, bits 2 - 0 of ICW2 show the ISR bit to be cleared. Note that when issuing EOIs in a cascaded system, both the slave's and the master's ISRs must be reset. Send an EOI to the slave. Then, if that slave's IRR is clear, send another EOI to the master.

The PC operates with its PIC in default priority configuration. But IR0 doesn't have to be the highest priority interrupt. The PIC can be made to rotate priorities. As an example, consider a system where several interrupts have equal priority. We can set the PIC to automatically rotate priorities at the end of each interrupt sequence. So after servicing IR5, the PIC sets it to the lowest priority. IR5 may have to wait until all other interrupts have been serviced once before it again has the highest priority. But it won't have to wait any longer than that.

The other possibility is specific rotation of priorities under software control. A routine may adjust priorities by setting the lowest one. Again, the three least significant bits of OCW2 tell the PIC which interrupt request will now be the lowest priority. If these bits read 110b, then IR6 moves to the end of the line and IR7 becomes the highest priority request. A specific rotation can be performed at any time, or a rotation can be combined with an EOI.

OCW3

The PIC sees any write with A0 and D4 low and D3 high as OCW3. Bits 6 and 5 set or reset the Special Mask Mode (11b = set and 10b = reset). In this mode, masking an interrupt using the IMR not only masks that interrupt but explicitly enables all other unmasked interrupts. An interrupt handler can rotate priorities and have the PIC treat incoming requests according to the new priorities. Interrupts which used to have lower priority than the currently executing interrupt can now be processed.

Bit 2 of OCW3 allows for a "polling" mode. When active (bit 2 = 1) the INT line is ignored. Instead, the PIC treats the next low on its RD line as an interrupt request. It then processes the highest priority interrupt, if any. So the 8088 polls the PIC rather than the individual interrupting devices.

Finally, bits 1 and 0 determine which register will be read during the next low on RD. By default the IRR will be read. To read the ISR, set bits 1 and 0 to 11b. To get the IRR again, use 10b. The PIC remembers the last register read. So if you're reading the same register repeatedly, just read port 20h. Don't issue an OCW3 for each read. Also, you don't need OCW3 to read the IMR. See the discussion of OCW1 above.

A Custom Hardware Interrupt

Hardware interrupt handlers usually live in the monitor ROM. But you can take over any interrupt and install your own memory resident code to handle it.

Figure 1 shows that IR2 is reserved. Reserved? For whom? Must be for us. An unused interrupt just begs to be connected with the outside world, so let's play around with it. We'll have it do something trivial (play some notes on the speaker). But you could make it do anything: answer the phone, gather

(continued next page)

Figure 3 - Hardware Interrupts

D7	D6	D5	
0	0	1	Non-specific EOI
0	1	1	Specific EOI
1	0	1	Automatic rotate on non-specific EOI
1	0	0	Set automatic rotate / automatic EOI mode
0	0	0	Clear automatic rotate / automatic EOI mode
1	1	1	Rotate on specific EOI
1	1	0	Rotate priority only
0	1	0	No effect

(continued from page 39)

temperature data, whatever. The shell of the code will remain the same. Just put in your own routine in place of the noisy one I've written.

You may recognize this interrupt handler. It's the same shell I used for the memory resident speed switch in issue #31's PC speedup article. This simple-minded shell fails to take into account other resident programs. When used to take over the keyboard interrupt (as in issue #31), it can run into trouble. The trouble stems from the portion of the "setup" procedure which checks to see if the program has already been installed.

There's really nothing wrong with installing a resident program twice, but it does waste memory and offend the programmer's sensibilities. So the shell checks to see where the interrupt code lives for the particular interrupt it wants to take over. If it finds the code in the ROM (i.e., segment address of F000h or more), it goes ahead with the installation. But if the interrupt vector points to RAM, then the interrupt handler thinks it's already been installed and aborts. The egotistical code thinks that no one else could be interested in the same interrupt.

So this shell must be installed before any other resident code which deals with the same interrupt. Big hassle for keyboard interrupts. But this time around, I'm using a reserved interrupt and I feel pretty safe.

The six lowest priority interrupts (IR2 - IR7) all originate in the PC's expansion bus. Only IR0 (timer) and IR1 (keyboard) come from the system board. All interrupt lines are normally held high. When a piece of hardware wants attention, it pulls its interrupt line low, then releases it. The BIOS has programmed the PIC to respond to rising edges, so as soon as the interrupt line goes high, the interrupt process begins.

I'm a great fan of the "quick and dirty" school of technology. Therefore, my interrupt request consists of a temporary short from expansion bus line B04 to ground through a 1K ohm resistor. Before installation of the interrupt handler, IR2 apparently does nothing. However, once an interrupt occurs it must be serviced. And location 0000:4*0Ah in memory does contain a pointer to the monitor ROM, so something's happening.

The interrupt vector points to the ROM's Temporary Interrupt Service Routine. This routine aids in power on diagnostics, and handles all subsequent unused interrupts by making sure that it won't have to deal with them again. Setting the corresponding bit in the IMR does the trick. Actually, the BIOS initially masks off all hardware interrupts except timer, keyboard, and diskette. But in case someone removes the mask without installing an interrupt handler, this ROM code comes to the rescue. A bit of protection for the careless programmer.

On To The Code

Figure 4 lists an assembler file which takes over interrupt 0Ah. Assemble and link normally to an .EXE file. Don't worry about the stack segment error from LINK. Run the result through EXE2BIN to create the final .COM file.

I won't say much about the setup procedure. Take a look at issue #31 for more information. The main difference here is that the mask on IR2 must be removed during installation of the interrupt handler.

As discussed above, the flags CS and IP get pushed onto the stack before the interrupt handler is entered. At the same time, the 8088 resets IF and TF (Trap Flag). We don't care about TF. It allows single stepping through code for debugging purposes. We do care about IF.

With IF reset, interrupts are disabled. One of the Great Laws of Programming states, "Thou shalt not disable interrupts (for very long)." Since the code in my interrupt handler doesn't mind being interrupted, its first instruction reenables interrupts (STI).

This brings up an interesting aside. There are no guarantees that some program hasn't disabled interrupts over a long period of time. What happens to the clock in this situation? Interrupt requests come in on IR0 18.2 times per second. Only the first request can be stored in the IRR. After that, the requests go to interrupt heaven. All those ticks are lost until interrupts are reenabled.

So if you need an exact value for the time of day, get it from your real time clock (RTC) - not from the BIOS time of day call. The same holds true for benchmarks. Your benchmark code probably won't cause the loss of any clock ticks. But just to be safe, time it with the RTC.

As always, initial values must be preserved for any registers used in the routine. So push those suckers onto the stack.

The body of my interrupt handler would have trouble impressing a 3-year old. Its main purpose in life is to test the hardware interrupt process. However, you can try a very instructive experiment with it. Assemble the code both with and without the initial STI instruction. Without the STI, the interrupt handler can't be interrupted and you'll hear a smooth transition of notes. But with the STI, you'll be able to hear the clock interrupting the sound 18.2 times each second.

Two housekeeping chores must complete the interrupt handler. Since this interrupt request has now been serviced, an EOI command to the PIC resets the appropriate ISR bit. Finally, an Interrupt Return (IRET) instruction pops the flags and registers off the stack and the 8088 resumes execution of the code which was interrupted in the first place.

Fin

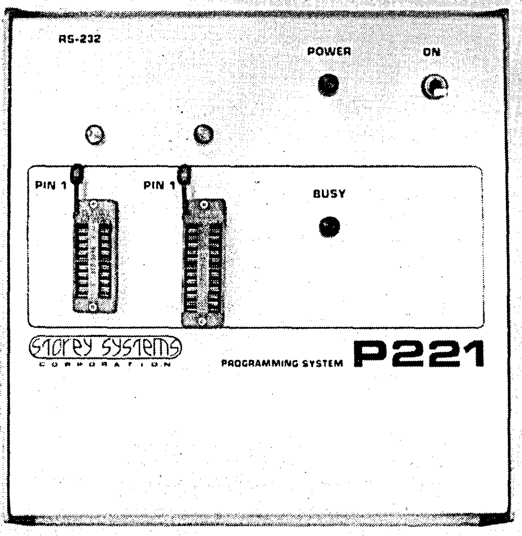
The PC's PIC obviously has a lot on its mind. Some day it would be fun to herd a few of them together. It could turn into the ultimate extension of the PC expansion bus (62 slots, folks - count 'em!) or an interface to some wild real world experiment. If anyone out there has done this kind of surgery to a PC, I'd love to hear from you.

That's all for now, folks. Next time we'll dive into the wonderful world of DMA.

■■■

Figure 4 - Interrupt Handler

```
title InterruptExample

code      segment                             ; everything in code seg
          org        100h
          assume     cs:code
DOS_entry            label      far
          jmp        setup
new_int              proc       far           ; beginning of int. handler
          sti                                 ; reenable interrupts
          push       ax                       ; save registers
          push       bx
          push       dx
          mov        al,0B6h                  ; set up timer for tones
          out        43h,al
          in         al,61h                   ; set 2 LS bits, enable spkr
          or         al,03h
          out        61h,al
          mov        bx,3000h                 ; initial tone divisor
top:      mov        ax,bx                    ; set up tone
          out        42h,al
          mov        al,ah
          out        42h,al
          mov        dx,30h
delay:    dec        dx                       ; sound tone
          cmp        dx,0
          jne        short      delay
          dec        bx
          cmp        bx,0
          jne        short      top           ; set up new tone
done:     in         al,61h                   ; disable speaker
          and        al,0FCh
          out        61h,al
          mov        al,20h                   ; signal end of interrupt
          out        20h, al
          pop        dx                       ; restore registers
          pop        bx
          pop        ax
          iret
new_int   endp                                ; end of our interrupt
end_res_code:

sign_on   db         'INT TEST NOW INSTALLED$'
err_msg   db         'INT TEST ALREADY INSTALLED$'
          assume     ds:code
setup     proc       near                     ; install our routine
                                              ; as resident code
          in         ax,21h                   ; get mask register
          and        ax,0FBh                  ; remove mask reset bit 2
          out        21h,ax                   ; send new mask to reg
          mov        ax,350Ah                 ; get address of int 0Ah
          int        21h
          mov        ax,es                    ; segment returned in es
          cmp        ax,0f000h                ; is this address in ROM?
          jae        short install            ; if so, install our code
          mov        dx,offset err_msg        ; if not, write msg
          mov        ah,9                     ; that our code is already
          int        21h                      ; installed
          int        20h                      ; exit to DOS
install:mov          dx,offset sign_on        ; write sign on msg
          mov        ah,9
          int        21h
          mov        dx,offset new_int        ; set up new
          mov        ax,250Ah                 ; interrupt vector
          int        21h
          mov        dx,offset end_res_code          ; make code
          int        27h                             ; resident
setup     endp
code      ends
          end        DOS_entry
```



PLD PROGRAMMING SYSTEM **$995**

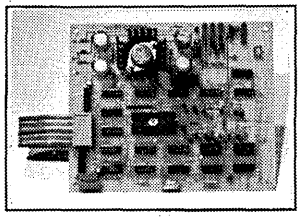
- LOWEST COST
- SUPPORTS MMI, NATIONAL, TI
- COMPLETE PLD DEVELOPMENT SYSTEM
- BUILT-IN COMPILER

CALL 1-800-852-2022

STOREY SYSTEMS
CORPORATION
3201 North Hwy. 67 Suite E Mesquite, Texas 75150 • (214) 270-4135

MS-DOS, CP/M E/EEPROM PROGRAMMING SYSTEM

2708
2758
2716
2516
2532*
2564*
68764*
2816A
2732
2732A



2764
2764A
27128
27128A
27256
27512
27CXXX
2864A
8751*
*SOCKET ADAPTER REQUIRED - DIAGAMS INCLUDED

A FULL FEATURED HARDWARE/SOFTWARE PACKAGE

- FAST PROGRAMMING ALGORITHM
- NO PERSONALITY MODULES REQUIRED
- INSTALL PROGRAM FOR SOFTWARE
- ALL SUPPLIES ON BOARD
- PROGRAMS 26, 25, 21 & 12.5V E/EEPROMS
- LARGE COMPREHENSIVE MANUAL
- USES NO SYSTEM POWER OR CHASSIS SLOT
- STAND-ALONE BOARD
- HIGH SPEED PARALLEL OPERATION
- FIVE LED STATUS/ACTIVITY INDICATORS
- HIGH QUALITY "TEXTOOL" ZIF SOCKET
- REQUIRES 24 OR 25 VOLT XFMR FOR POWER

PARALLEL PRINTER INTERFACE
CONNECTS TO ANY PARALLEL PRINTER INTERFACE
USES 8 OUTPUT DATA BITS AND THE PRINTER BUSY LINE FOR DATA INPUT

CONTROL PROGRAM COMMANDS

- PROGRAM EPROM(S) FROM DISK FILE
- READ DISK FILE INTO BUFFER
- READ EPROM(S) INTO BUFFER
- VERIFY EPROM IS ERASED
- CHANGE EPROM TYPE
- SAVE EPROM(S)/BUFFER TO DISK
- PROGRAM EPROM(S) FROM BUFFER
- COMPARE EPROM(S) WITH BUFFER
- COPY EPROM(S)
- BUFFER MONITOR MODE (SEE BELOW)

THE BUFFER MONITOR MODE HAS 17 SUB-COMMANDS FOR DETAILED OPERATIONS. THESE INCLUDE: FILL, DUMP, TRANSFER, PROGRAM, READ, VERIFY, EXAMINE, MODIFY, CHECKSUM, BIAS, INSPECT, SINGLE BYTE BURN, LOGICAL OPERATIONS(AND/OR/XOR), SET BUFFER BIAS, HEX ARITHMETIC, ETC.

ASSEMBLED AND TESTED UNIT WITH COMPLETE DOCUMENTATON AND SOFTWARE ON DISKETTE **$199**

PARTS KIT WITH SOFTWARE AND DOC.-$179 BARE BOARD, SOFTWARE & DOC.-$69
SOFTWARE AVAILABLE ON 5 1/4" OR 8" DISK FOR IBM, KAYPRO, & OTHER FORMATS

TO ORDER SEND CHECK, MONEY ORDER, WRITE OR CALL:

VISA

**ANDRATECH
P.O. BOX 222
MILFORD, OHIO 45150
(513) 752-7218**

MASTER CARD

CALL OR WRITE FOR MORE INFORMATION -- ADD $4.00 FOR SHIPPING - $3.00 COD

By Stephen M. Leon
200 Winston Drive
Cliffside Park, NJ 07010

New Games And New Business Software For The PC

Steve tests the inexpensive multifunction RAM cards and finds a winner. Then he covers the new MS-DOS disks from PC-Blue.

Compatibility. It's probably the most commonly used word in the computer field today. In fact, I was all set to write this column about compatibility problems I was having with my Everex Magic Card 16 AT multifunction board. Unfortunately, I had to rewrite the column when it turned out that the card was defective.

But concerning compatibility in general, I now insist that a vendor guarantee his product will work in my system, as it's set up, or I get my money back.

The Everex Magic Card

The multifunction card looked like a perfect way to add a RAM disk to the BBS system we use at work. It was reasonably priced and designed to work at 10 MHz. Instead, even with 120 ns prime chips, it gave us strange memory errors. It turned out that there was enough of a glitch in the Everex card to make it allergic to the PC Network card we were using. However, once the Everex was replaced, the whole system worked perfectly.

We had another problem helping a friend set up a new AT clone. All of us realize that the best way to solve an equipment problem is to switch parts between two similar computers. We did that and everything worked in the other computer, so we put the blame on the mother board. A new mother board produced the same problem. You guessed it - it was the brand new power supply, shirking its duty.

Here's another example - two spanking new XTs arrived at the office the other day. We set them up. One worked fine. The other gave a disk controller card error message. Since we buy directly from IBM, their service department came to the office and changed the mother board, the hard disk, and the disk controller! So much for quality control.

New PC/BLUE Releases

Last issue we made mention of PC/BLUE 277 containing this horrible example of shareware called AMTAX86. (This was the program that allowed you to calculate your income tax, but would not print a return until you sent them a contribution.)

I persuaded Hank Kee to pull back the release. He did that, but then re-issued it as PC/BLUE 281. However, this time he also included on the disk a program by Stephen F. Procko called TAX87. It does a projection of your 1987 taxes, so the disk is not a total waste.

Dave Alexander has revised his MR. BILL, a legal time and billing system, and it is on PC/BLUE 277 and 278. (The earlier version was on 207 and 208.) The program is more than adequate for a small office billing on a time basis.

I prepared the two MR. BILL volumes and 279 and 280 for PC/BLUE. Volume 279 has a good label program called Label Master and a not too bad outliner and text processor, the Classical Classifier. If you are big on disk manager systems, by all means take a look at 280. Commando and Master Key are both excellent disk managers if you need help in handling DOS.

There is, however, a real gem on 280. I never did figure out how they got the name, but MSPANTOC from the Nunnery Works, Ltd., is something that you should get if you do any document formatting. It will give you the style sheet capabilities of Microsoft Word plus many, many more features.

Games People Play

Volume 284 has Intercept, Flightmare and Monopoly 6.2 (all CGA) as well as the "Original" Adventure (text). I suspect that Parker Brothers (or whoever now owns Monopoly) will one day sue to stop the computer version of its game. However, a board game on a computer, as good as it is, is really nothing more than whetting the appetite of the user to go out and buy the real thing. Unless you have a spare computer to keep the game board on, it sure is easier to play Monopoly with the real game - and a lot more fun.

Speaking of fun games - while you will never see these in the PC/BLUE library, I've seen an increasing number of sexually-oriented games on the bulletin boards. From the download statistics, they are very popular. The best of the lot seems to be a program called COUPLES, written by a student at Union College.

More family oriented is PC/BLUE 284, which contains Bible-Q, a Bible quiz by Rev. and Mrs. Robert Smith. For Ivan Boesky fans, the same volume has a stock market simulation. I am not positive, but I think hidden in the code of that program is inside information. I-Ching, the book of changes, is on Volume 283. Genealogy on Display (version 5.0), another very popular program, is on volume 285.

Attention All SYSOPs

Last issue we mentioned a program I wrote for dBase3 and Clipper called BBDIR. It automates directory handling for BBS systems. It is now on PC/BLUE 288, as is Dan Doman's BBSUTILS. Dan has a fine collection of utilities to sort directories and to match up what is on the disks and in the directories. If you are a SYSOP, or hope to be one, you should have this volume.

Since I wrote BBDIR I have been helping a number of SYSOPs get their directory structure in order. The program seems to work well with all of the various directory structures we have run into. However, we have had to shift the spacing in one of the data bases to match the file structure of FIDO. Anyone with a little dBASE familiarity can modify the file structure of the transitional data base (called TEMP.DBF) to get it to work well.

Business, Business, Business

Several times a week I get calls and letters from people seeking the dBase3 version of my Property Management program. We finally decided to release it in PC/BLUE and it is on Volume 289. It handles all aspects of the management of tenanted real estate. The program is written in dBASE3 plus and comes with source code. It also comes with a Clipper compilation.

However, it was written before the Autumn '86 release of Clipper, and we created BROWSE with a special library. Therefore, if you decide to do a recompile, you will have to rewrite the BROWSE to the new version of Clipper. It is not that difficult to do. (We have done it for our office programs which contain BROWSE.) The reason we did not do it for the release was because the Harry Van Tassell version of BROWSE, which appears in our old Clipper library, is far superior to the August '86 version of Clipper.

Also on the business side is Volume 290 - AsEasyAs, an excellent Lotus 123 clone. (Another lawsuit?) It cannot handle a spread sheet beyond 1024 rows and 256 columns - but who needs such a big spread sheet? Volume 283 also has a spread sheet - Betterway Calc. In the quasi-business area, we have two desk management utilities on 291. DeskTeam and Right Hand Man are for those of you who do not want to go out and buy SideKick.

My Kind Of Disk

PC/BLUE Volume 287 is my kind of disk. It is a collection of 32 screen handling and subdirectory utilities. The screen handling utilities are:

DISPLAY.ARC - blink, highlight, etc., parts of screen

EGA4325.ARC - EGA switch 25-43 line mode

FONT3270.ARC - EGA change screen font to emulate 3270

FRULER.ARC - memory resident on-screen ruler

HERC1.ARC - control Hercules graphics

HERCBIOS.ARC - Hercules BIOS

HERCIBM.ARC - CGA emulation for Hercules and clones

HGC.ARC - clears screen within 5 minutes for Hercules

HGCIBM.ARC - converts CGA programs for Hercules

HVIEW.ARC - Hercules 43 lines by 180 columns

KBORDER.ARC - memory resident COLOR set

NEWFNT30.ARC - EGA fonts

PALCON.ARC - EGA palette utility for Lotus 123

SAVSCR.ARC - text screen to text file w/PrtSc

SCRNSV20.COM - blank screen after 20 min of non-use

SIMCGA.ARC - simulate CGA on Hercules monochrome

SPS13.ARC - select part of screen for PrtSc

SWH.ARC - Hercules ONLY arcade game

VIDEO.ARC - change screen color attributes

The subdirectory utilities are:

ALTER.ARC - hides/unhides files and subdirectories

CATT.COM - change file attributes

CDD.ARC - change drive and directory in one shot

CDSECRET.COM - change to a secret subdirectory

DELDIR.ARC - delete subdirectories and files

DIRUT3.COM - global drive directory

DSIZ.ARC - directory tree with size

MDIR2.ARC - RAM resident directory utility

PMOVE5.ARC - move files between subdir/devices

RENDIR.COM - rename subdirectories

SECDIR.ARC - hide/unhide subdirectories

TREEDIR.ARC - locate all files in all subdirectories.

WHEREIS4.ARC - locate file within subdirectories

No New SIG/M Disks

Since yours truly stepped down as SIG/M disk editor, there have been no new SIG/M releases. The last volume is still 294. It was released on November 21, 1986. I received considerable mail on the subject, including one from Professor Harold McIntosh (the author of REC). He comments:

"There was some comment ... about a huge volume of CP/M programs waiting to be released. This is a statement about which I, myself, am skeptical. It will be interesting to see if somebody actually manages to materialize these programs.

"Whatever may be said about the installed base of CP/M machines, their importance to their owners, and the remaining years of service which they have to offer, it just doesn't seem to me that there are going to be any major software efforts oriented toward them. This is a shame. Now that Digital Research cannot have much of any commercial interest in defending its proprietary interests, it would be nice to go back and do CP/M over again - right, this time."

That does not mean to say, however, that there is not a wealth of CP/M material still out there. With 294 volumes in the SIG/M library - most of it with source code - the majority of your requirements should be met. If you are still writing new CP/M software for your own use, or if you have programs that you developed under CP/M that you would be willing to share, why not make a contribution to the SIG/M library.

So Nice to Meet You

It really has been quite enjoyable to see the vast number of Micro C readers who call up the BBS system we set up. I have chatted with quite a number of you and find you to be a swell bunch of guys and gals. I now know why everyone has such a good time at SOG.

In any event, we have a multi-user BBS system running on two lines. It consists of two AT clones and two XT clones. One AT clone is the server. The two lines feed into two turbo XTs, and the second AT services the system. If need be, a third line will be added and the second AT will be on line most of the time.

(continued next page)

XT CLONE SYSTEMS

(One YEAR guarantee on system)

Turbo Mother Board 4.77 and 8 MHz.
640 K Ram installed on board
Serial, Parallel, Game Ports
Clock/Calendar   AT Style Keyboard
Color Video Board (CGA) Monochrome Opt
150 Watt Power Supply   Flip Top Case

ABOVE WITH 2 FLOPPY DISK DRIVES	**$ 699.00**
WITH 1 FLOPPY AND 20 MEG	**$ 950.00**
WITH 2 FLOPPY AND 20 MEG	**$1000.00**

Assembled and Tested for 24 Hours

20 MEG HARD DRIVE WITH CONTROLLER LAPINE LT200	**$310.00**

AT TURBO SYSTEM

AT COMPATIBLE MOTHER BOARD WITH BIOS
6 MEG AND 10 MEG SWITCHABLE SPEED
512K RAM INSTALLED UP TO 1024 ON BOARD
WA2 HARD DISK/FLOPPY DISK CONTROLLER
MONOCHROME GRAPHICS VIDEO WITH PRINTER
1.2 MEG OR 360 K FLOPPY
220 WATT POWER SUPPLY   AT CASE
AT KEYBOARD   SET UP DISK

ONE YEAR WARRANTEE ON SYSTEM	**$1095.00**
EGA UPGRADE FOR ABOVE	$ 150.00
512K UPGRADE (1024 INSTALLED)	$ 50.00

HARD DISK DRIVES

20 Meg Seagate ST4026	(for AT)	$ 495.00
30 Meg Seagate ST4038	(for AT)	$ 595.00
30 Meg CDC Wren 1	(for AT)	$ 595.00

Does NOT include controller

Color Monitor RGB (CGA)	$ 300.00
Color Monitor RGB (EGA)	$ 510.00
Monochrome TTL (Green)	$ 110.00
Monochrome TTL (Amber)	$ 120.00
EGA Color Video Card	$ 195.00
MS DOS 3.2 WITH GW BASIC	$ 85.00

CITIZEN PRINTERS

MODEL 120D	120 CPS	9"	$ 200.00
MODEL MSP-10	160 CPS	9"	$ 300.00
MODEL MSP-15	160 CPS	15"	$ 400.00
MODEL MSP-20	200 CPS	9"	$ 350.00
MODEL MSP-25	200 CPS	15"	$ 500.00
MODEL 35	35 CPS LETTER QUALITY		$ 500.00

ALL PRINTERS COME WITH CABLE

CASCADE ELECTRONICS, INC.
ROUTE 1 BOX 8
RANDOLPH, MN 55065
507-645-7997

Please ADD Shipping on all Orders
COD Add $3.00   Credit Cards ADD 5%
Limited to Stock on Hand Subject to change

(continued from page 43)

Both the SIG/M library and the PC/BLUE library are available on the system. At last count, approximately 90 full PC/BLUE volumes (each as a single .ARC file labeled PCxxx.ARC) were on line without prior request. Twenty-five SIG/M volumes (each as a single .LBR file labeled SIGxxx.LBR) were also on line. More than 2000 other separate files were also on line, including more than 200 AMIGA files. There are no preregistration requirements, no hassles - and no nonsense.

First time on you get full access to the system, and a full 92-minute time frame per day. There is no requirement that you upload to us. We actually prefer that you do not upload, unless it is software you have written. The phone number is (201) 886-8041. That line jumps to other available lines, so please don't call my voice line in the middle of the night with your computer. (Several issues back, I made a mistake and put in the voice number as the computer number. I still get calls on that one.)

Unless you have a friend at the telephone company, all SIG/M and PC/BLUE volumes are available through local SIG/M and PC/BLUE distribution points or may be ordered directly. In addition, any SIG/M or PC/BLUE volume (or any file from any volume) not already on the BBS will be put up on request.

SIG/M volumes are available on 8" SSSD disks for $6 each ($9 foreign) directly from SIG/M, Box 97, Iselin, NJ 08830. They are also available in most 5" formats. The charge for 5" disks is $7 per volume. However, for SSSD formats, or any format which requires more than one disk, please add another $2 per volume. Printed catalogs are $3 each ($4 foreign). PC/BLUE volumes are $7 each ($10 foreign). The printed catalog is $5.

Both are available from the New York Amateur Computer Club, Box 3443, Church Street Station, New York, NY 10008. (Note that it is a new box number for PC/BLUE.) Both groups have a disk catalog (Volume 0) available at the price of a standard disk volume. This catalog volume is usually more current and more readily available than the printed catalog.

■■■

Disk storage problems? We have the solutions.

Hard Disk Host boards with software for your Z80 by Emerald Microware and MICROCode Consulting

No other upgrade improves your computer's productivity like a hard disk. We have everything you need to install a hard drive on your Xerox 820, Kaypro, Zorba, or almost any Z80 system.

- Host plugs into the Z80 socket, no wiring required
- Interfaces to the WD1002 controller board
- Works with one or two hard disks - 5 to 64 meg
- Menu installation, no software to assemble
- Automatic swap, warm boot from hard drive
- BIOS drivers install above BIOS or below CP/M
- Selectable I/O port addressing
- Allows custom partitioning and mixed drives types
- Includes manual, format program and extensive utilities
- Host board comes assembled and fully tested

HDS Board with software	$ 89.00
HDS Board with software and WD1002-05 board	$250.00

WD1002-05 Hard Disk Controller Board by Western Digital

- Standard ST506 drive interface (20 & 34 pin conn.)
- Same size as standard 5 1/4" drive
- 40 pin host interface
- Can control up to three hard drives
- Direct replacement for Kaypro 10 controller

WD1002-05 Controller Board	$185.00
Rodime RO252 - 10 Meg - 3½" Hard Drive	$275.00
LaPine 20 Meg - 3½" Hard Drive	$345.00

Adaptec 2070 - RLL Hard Disk Controller Card for the PC/XT

The Adaptec 2070 controller board uses the new Run Length Limited technology to squeeze an additional 50% more capacity than a standard MFM encoding. For example, a 20 meg drive formatted in standard MFM mode can operate with a capacity of 30 Megabytes in RLL mode. Works in both RLL and standard MFM encoding modes. Please call or write for RLL compatibility list.

Adaptec 2070	$135.00

THE KayPLUS ROM PACKAGE by MICROCode Consulting

The most important element in the performance of your Kaypro is its monitor rom. With KayPLUS you get all of the advantages of a Kaypro 10, even on your Kaypro 2.

- Boots from floppy or hard disk
- Install up to four floppies and two hard drives
- No software assembly required
- Runs standard single and double sided formats on 96 TPI drives
- 32 character keyboard buffer
- Automatic screen blanking
- 12 disk formats built-in
- Full automatic disk relogging with QP/M
- Internal real-time clock support
- Includes manual, standard utilities, AND hard disk utilities

KayPLUS ROM Set	$ 69.95
KayPLUS ROM Set with QP/M . .** SPECIAL **	$115.00

Parts and accessories for the Kaypro and XEROX 820

Kaypro 2X Real-time Clock parts kit	$ 29.00
Kaypro 2X Hard disk interface parts kit	$ 16.00
Kaypro 10 or '84 series Hard Disk host board	$ 49.00
Kaypro four drive floppy decoder board	$ 35.00
Xerox 820-2 CPU Board - new	$ 75.00
Xerox 820-2 Floppy Controller board - new	$ 65.00
Xerox 820-2 CPU board w/ Floppy Controller	$125.00
Xerox 820 power supply	$ 30.00
Xerox 820-1 CPU board - new	$ 85.00
Xerox 820 complete high profile keyboard	$ 65.00
Xerox 820 bare high profile keyboard - new	$ 25.00
Xerox 820 5¼" drive cable	$ 9.00
Xerox internal video cable	$ 7.00
Xerox 820-2 cabinet (no CRT frame) - new	$ 65.00
Xerox CPU board mount power connector	$ 2.50
Dual Half Height 5¼" Disk Drives - DSDD, in cabinet with standard Xerox cable	$265.00

Call for other parts or repair services.

Prices subject to change without notice. Include $4.00 shipping and handling, $7.00 for COD, call for Blue Label charges. VISA and Mastercard accepted. 30 day money back guarantee on all products.

Keep better track of your files with QP/M by MICROCode Consulting

Full CP/M 2.2 compatability with outstanding performance and many more features. You've seen the replacements that eat up memory and need auxillary programs to run. Not QP/M. Fifteen internal commands, automatic disk relogging (no more control C), user area selection from colon, 31 user areas, drive search path, archive bit maintenance, and transparent time/date stamping; all in the same space as CP/M 2.2. Installs from a convenient customization menu, no software assembly required. Bootable systems available for the BBI, Kaypro, and Xerox 820.

QP/M Operating System, complete bootable copy	$ 80.00
QP/M without BIOS (installs on any Z80 system)	$ 60.00

Get the ultimate in versatility on your Xerox 820-1 with the PLUS2/X120 Double Density package by Emerald Microware and MICROCode Consulting

Run up to four floppy disk drives at once, both 8" and 5¼" at the same time. Software compatable with Kaypro and Xerox 820. Supports all standard serial and parallel printers, and most add ons like the Ferguson Ram Board. You get mini-monitor functions, autoboot capability, 19 built in disk formats, and bank mode operation for more space in your TPA. Lets you run 48 TPI disks on 96 track drives. Works with Uniform and QP/M.

PLUS2 ROM Set and X120 Board A&T	$135.00
PLUS2 ROM Set and X120 Bare Board	$ 62.00
PLUS2 ROM Set only	$ 49.95
120 Bare Board only . .** CLOSE-OUT SPECIAL **	$ 15.00

or two for $ 25.00, five for $ 50.00

Other kits, parts, and packages available

End diskette incompatibility on your PC with UniForm-PC by Micro Solutions

This program allows you to read, write, copy, and format diskettes for over a hundred CP/M, TRSDOS, and MSDOS computers on your PC, XT, or AT, including 8", 96 TPI, high density, and 3½" formats (with optional hardware). Once installed, UniForm is mem ory resident so you can use your standard DOS commands and other programs directly on your original diskettes.

UniForm-PC	$ 64.95
Uniform for Kaypro and other machines	$ 64.95

Run your CP/M programs on your PC/XT/AT with UniDOS by Micro Solutions

UniDOS uses the NEC V20 or V30 CPU chips to actually RUN your favorite 8080 CP/M programs on your PC. Z80 programs can be run in an emulation mode. Use UniDOS with UniForm-PC, and you can run them directly from your CP/M format diskettes. All standard CP/M system calls are supported.

UniDOS	$ 64.95
UniDOS w/ UniForm & V20 chip	$129.95

The CompatiCard universal floppy drive controller by Micro Solutions

Finally a full function floppy controller card that allows you to run up to four 8", 5¼" (standard, 96 TPI, or high density), and 3½" disk drives on your IBM or compatible. This board works with the UniForm-PC program to format, read, and write lite rally hundreds of CP/M, TRSDOS, and MSDOS disk formats on your PC/XT. The Compaticard can be addressed to use with up to three other floppy controller cards.

CompatiCard	$169.95
Compaticard with UniFORM-PC	$225.00

Use your Apple or North Star disks on your PC/XT with MatchPoint-PC by Micro Solutions

This half-card works with your controller card to let you read and write to NorthStar hard sector and Apple II diskettes on your PC. INCLUDES a copy of the UniForm-PC program, as well as utilities to format disks, copy, delete, and view files on Apple DO S, PRODOS, and Apple CP/M diskettes.

MatchPoint-PC	$169.95

VISA

EMERALD MICROWARE

MasterCard

P.O. Box 1726, Beaverton, OR 97075
(503) 641-0347

PC-DOS 3.00 And Beyond

By Siegmund F. Kluger
Definicon Systems
31324 Via Colinas #108/9
Westlake Village, CA 91362

The Search For Inside Information

If you've been trying to write reentrant code for MS-DOS, stay with us. This super piece is really a blow-by-blow heavyweight look at mucking with DOS. Note that you'll need MS or PC-DOS 3.XX to do any of the exciting things that follow.

After hacking up operating systems such as CP/M-80 and TurboDOS, I was pushed headfirst into the MS-DOS world a bit over a year ago. This has led to a seemingly endless search for inside information and undocumented functions.

When I started out in CP/M, many "useful utilities" were either in their infancy or had not yet been written. Under MS-DOS, the big stuff is, for the most part, already available, so I've limited myself to writing little stuff. Besides, two kids and caring for Definicon's 68020 product line doesn't leave much time to play.

I admit that most of my knowledge has come from the BYTE Information Exchange (BIX), in particular the "ms.dos/secrets" conference. Some information I have painstakingly extracted with a debugger, only to find it posted there. Or, as happened a few times, after having posted my own insights, I've had someone say, "Hey, don't you read XXX Magazine," (no I don't), "it was published there!"

Anyway, here are a few insights.

Reentrancy

While writing a background loader/interface for the DSI-020 to allow 68020 programs to do something useful, I had to overcome the same hurdles everyone has been writing about with respect to DOS reentrancy (or rather the lack thereof). I did it mostly by reinventing the wheel.

By now, you probably know that you can use the so-called INDOS flag and poll once every timer tick. Many people know that DOS makes an INT 28H call every time it polls for a character, saying, "despite INDOS, it's safe NOW to enter."

Thus, in order to steal CPU time for a background task, I trap both INT 1CH and INT 28H. I use INT 21H function 34H to get the pointer to INDOS and store it. Thus I avoid using an INT 21H to determine whether it's safe to make an INT 21H call... Mostly, it saves execution time.

If I see an INT 1CH, I do an EOI to tell the hardware I received it, then check INDOS and execute my routine if INDOS = 0, otherwise I simply do an IRET.

If I see an INT 28H, I execute my routine without any further DOS checking.

Of course, it's possible for the background to take longer than 1/18th of a second to execute, in which case the next INT 1CH will come along, finding INDOS = 0, and go on to wreak havoc. Here's where the "mutual exclusion semaphore" (one of the very few things I still remember about TurboDOS) comes in.

The background routine checks a flag before doing anything else. If it finds the flag set, it doesn't execute, but instead exits to the IRET above. Otherwise, it quickly sets the flag, then saves all registers, sets up a new stack pointer (IMPORTANT!), and executes the background task. When it's finished, after restoring all of the caller's registers and just before IRETing, the mutual exclusion flag is reset, allowing the next call to enter.

This isn't really a semaphore, since according to the MX-semaphore used in TurboDOS, the calling process waits until the semaphore is cleared. Here, the calling process skips if the MX-flag is set.

See Figure 1 for some example code.

This code should be largely self-explanatory to an experienced assembly language programmer.

Writing DOS Files In The Background

Many stories have been written on read/write file access in background mode. Most of them do indeed work. But note that in order to satisfy DOS requirements better, we should be using a technique that I haven't yet seen in print.

The first 256 bytes of the data segment of any DOS program is called the PSP (Program Segment Prefix). UNIX gurus are familiar with the term PID (Process ID). DOS also has a PID and a (documented) function to interrogate the current PID. It also has a function to SET the current PID, but the function isn't documented.

In DOS, the PID is the segment address of the current program's PSP! Everything DOS cares about is referenced by that PSP, including the Process File Table which contains the handles of currently open files.

Now, assume the following scenario:

1. A background process begins to run asynchronously, possibly a BBS program or an unattended XMODEM transfer.

2. A program begins to run in the foreground.

3. The background process opens a file and starts writing to it. DOS, in its ignorance, uses the current PSP to record the file handle. The current PSP is, naturally, that of the foreground program.

4. The foreground program terminates. DOS closes all the files the foreground program had left open. The current PSP is switched to COMMAND.COM's PSP.

5. The background program's write

request fails because DOS closed the file when killing the foreground process, and COMMAND.COM doesn't currently have a file open with a matching handle. (Imagine COMMAND.COM opening a file with the same handle as that of the background program, then the BG program writing to it!!!)

Clearly, only very careful coding can avoid disaster. If you don't wish to make the precautions necessary to prevent the above scenario, you must use an undocumented DOS function. See Figure 2.

In Figure 2, the first thing the "BGRTN" code should do is use FCN 51H to get the current PSP. It should then save it in memory, set up BX to point to the local PSP (always CS register in a .COM file), and use FCN 50H to tell DOS about the new PSP. When finished, and just before returning to the task dispatcher, get the saved PSP value and use FCN 50H to set it back to what it was before the call.

Crossing The 15-File Barrier

HELP! I need to work on 25 files all at once and DOS only lets me open 15 simultaneously!

Don't worry, help is on the way. DOS limits programs to a maximum of 20 files (15 available, 5 predefined). You can cheat by closing handles 2, 3 and 4 and get 18 files. But if you really need 25 (or more) files at the same time, here's how to do it.

At offset 18H of the PSP is a 20-byte table of file handles. This table should be duplicated elsewhere in memory, its size depending on the number of files you'll want open.

Let's assume we want to be able to have up to 50 files open at one time. So, the new PFT (Process File Table) has to be 50 bytes large. All 50 bytes should be initialized to 0FFH.

First, you must copy the old PFT at PSP:18H over the new one to keep track of any pre-opened files (usually only handles 0-4). At PSP:34H there's a 16-bit offset which points to the start of the PFT. Change that location to point to the offset of the new PFT.

The segment address of the PFT is located at PSP:36H. Stash the segment address of the new PFT there if it's different from the PSP segment. Finally, stash the maximum number of files

(continued next page)

Figure 1 - Redirected Entry Points

```
;          This is the redirected INT 1CH entry point
           TICK    1C:CLI                        ; no interrupts
           MOV     CS:Byte Ptr IS1C,1            ; set a flag to indicate 1C
           MOV     AL,20H                        ; EOI
           OUT     20H,AL
           PUSH    DS
           PUSH    SI
           LDS     SI,Dword Ptr INDOS            ; get INDOS flag
           CMP     Word Ptr 0FFFFH[SI],0         ; check
           POP     SI
           POP     DS
           JNZ     MISS
           JMPS    HITIT
;
;          This is the redirected INT 28H entry point
DI28:      CLI
           MOV     CS:Byte Ptr IS1C,0            ; indicate not 1CH
HITIT:     CMP     CS:Byte Ptr MXSPH,0           ; check MX flag
           JNZ     MISS
           MOV     CS:Byte Ptr MXSPH,0FFH        ; set MX flag
           STI
; SAVE ALL REGISTERS AND SET UP NEW STACK HERE
           CALL    BGRTN                         ; call our background code
           CLI                                   ; no interruptions please!
; RESTORE ALL REGISTERS HERE
           MOV     CS:Byte Ptr MXSPH,0           ; clear the MX flag
MISS:      CMP     CS:Byte Ptr IS1C,0            ; did we have a timer tick?
           JZ      MISS28                        ;  no
           STI
           JMPF    CS:Dword Ptr TICKSV
;
MISS28:    STI
           JMPF    CS:Dword Ptr I28SV
```

Figure 2 - BGRTN

```
GET PSP:
INT 21H function 51H or 62H (identical)

AH = 51H
On return,
BX = current PSP segment

SET PSP:
INT 21H function 50H (undocumented)

AH = 50H
BX = our PSP segment
```

Figure 3 - Set Process File Table

```
MOV     AX,ES:Word Ptr .36H       ; get old PFT segment
PUSH    DS                        ; move our DS
POP     ES                        ;  into ES
MOV     DS,AX                     ; point to old PFT segment
MOV     SI,18H                    ; start of old PFT
MOV     DI,Offset PFT             ; start of new PFT
MOV     Word Ptr .34H,DI          ; set pointer to new PFT
MOV     CX,20                     ; copy all 20 bytes
REP     MOVSW                     ; copy old process file table
MOV     AX,ES                     ; get new PFT segment
MOV     Word Ptr .36H,AX          ; stash into PSP
MOV     Byte Ptr .32H,MAXFLS      ; and store max number of files
```

(continued from page 47)

into PSP:32H as a 16-bit number. See Figure 3 for some sample code.

CP/M-86 programmers may recognize the assembler syntax. Yes, I use RASM for all my 8086 assembly work. There is a version of RASM86 (with a linker for use with PC-DOS) which was originally intended for CCP/M, and is now sold by Alexander & Lord in Carmel, California. I hate the brain-damaged convolutions one has to go through when programming in MASM.

Imagine UNIX(tm)...

Now you can have a lot of the look and feel of UNIX on your PC. I usually work with "#" as a prompt, JOIN drives to form one single pseudodrive, use slashes in pathnames, and ignore executables in the current directory. Let's go over this in detail.

The Bleeping Backslash

The single biggest objection I have to the DOS command syntax is inconsistency. (There must be an intense dislike of AT&T by the folks at IBM.) Internally, DOS allows forward slashes, so that in C, "fd = fopen("/foo/bar/ zot.dat", "w")" is perfectly legal, while on the command line, "copy /foo/bar/zot.dat ." stirs up an error message.

In older versions of DOS, the undocumented "SWITCHAR" command could be used to change the so-called SWITCH CHARACTER from "/" to something else like the UNIX-ish "-" that I prefer.

I guess IBM musta found out because that feature is no longer present in DOS 3.x. The function call, however, is still in there and can be used. It's just that many DOS 3.x applications require "/" for a switch character. With the switch character set to "-" you can, for example, use "CHKDSK -F", but if you use XCOPY you must use "/", as in "XCOPY *.* D: /S". Check out the code in Figure 4.

You can use the program in Figure 5 to change or report the current switch character. Note that I've used a different assembler - this time it's A86.COM which is distributed on BBS as beggarware.

The CWD Execution Dilemma...

Let's say you've written a fabulous TSR. You've just assembled and linked it. You should EXE2BIN it, and you

Figure 4 - Get Switch Character

```
INT 21H FCN 37H
AX = 3700H
get switch character
Returns:
DL = current switch character

AX = 3701H
DL = new switch character
set switch character
Returns:
nothing, the new switch character is set.
```

Figure 5 - Change Or Display Switch Character

```
;               CSC - Change SwitchChar
;
;               This program can be used to display or change the
;               MS/PC-DOS switch character.
;               Invoke with:
;                               CSC  E>
;               to display the current switch character, or
;                               CSC  /
;               to set the switch character to /.
;
;               The first nonblank character following the command name
;               is taken to be the new switch character.  Example:
;               To set the switch character to "-", enter:
;                               CSC -
;
                org             100h
;
start:          jmp             skip                    ; skip over the data area
;
message:
                db              0dh,0ah
                db              'Switch character is "'
sch             db              ' "',0dh,0ah,'$'
;
skip:           mov             ax,cs
                mov             ss,ax                   ; set stack
                mov             sp,start                ;  to below the program
                mov             bx,80h                  ; point to commandline
                xor             ch,ch                   ; get line length
                mov             cl,[bx]                 ;  into CX
                jcxz            tell                    ; display old swchar if no argument
fsc:            inc             bx                      ; else scan for first nonblank
                mov             dl,[bx]
                cmp             dl,' '
                jnz             found
                loop            fsc                     ; continue till exhausted
                jmp             tell                    ; only blanks in argument...
;
found:          mov             ax,3701h                ; set up for "set switch character"
                jmp             newsc
;
tell:           mov             ax,3700h                ; set up for "get switch character"
newsc:          int             21h
                mov             sch,dl                  ; stash it into display string
                mov             dx,message                          ; point to string
                mov             ah,9
                int             21h                     ; display the message
                mov             ax,4c00h                ; return with no error
                int             21h                     ; all done, DOS here we come!
;
                end
```

Figure 6 - TSR Routine Limits File Searches

```
;        This small TSR makes MS/PC-DOS file executable file
;        searches behave like UNIX where the local directory
;        is never searched unless specified in a PATH command
;        or typed on the command line.
;        Example:
;        Assume you have the files C:\BIN\FOO.COM and C:\WORK\FOO.COM
;        and are logged into C:\WORK. PATH=C:\BIN
;        Cfoo                will execute C:\WORK\FOO.COM
;        After installing this TSR, things will change a bit:
;        Cfoo                will execute C:\BIN\FOO.COM, whereas
;        C.\foo              will execute C:\WORK\FOO.COM
;
         CSEG
         ORG     100H
;
SETUP:   MOV     AH,30H                  ; get DOS version
         INT     21H
         CMP     AL,3                    ; v3.xx?
         JZ      V3
         MOV     AH,9
         MOV     DX,Offset NOTV3         ; complain!
         INT     21H
         MOV     AX,4CFFH                ; set errorlevel on exit
         INT     21H
;
V3:      MOV     AX,3521H                ; get int21 vector
         INT     21H
         MOV     Word Ptr INT21,BX       ; and save it here
         MOV     Word Ptr INT21+2,ES
         MOV     AX,2521H                ; replace with our little detour
         MOV     DX,Offset ENTRY
         INT     21H
         MOV     AX,Word Ptr .2          ; get top of memory
         MOV     Word Ptr EOM,AX         ; stash into program
         MOV     DX,Offset BANNER
         MOV     AH,9
         INT     21H
         MOV     AX,3100H                ; terminate wasting 1k of RAM
         MOV     DX,64
         INT     21H
;
ENTRY:   CMP     AH,4EH                  ; have a search first call?
         JZ      IS4E                    ; yes!
CONTI:   JMPF    Dword Ptr INT21         ; continue on to DOS
;
;        NOTE:
;        The constant 800H below is an empirical value assuming that
;        the CS of the transient part of COMMAND.COM can't be more
;        than 32k below the top of memory...
;
IS4E:    ADD     SP,2                    ; point to caller's CS
         POP     Word Ptr SEGM           ; get caller's code segment
         SUB     SP,4                    ; adjust stack pointer
         PUSH    AX
         DB      0B8H                    ; MOV AX,
EOM      DW      0                       ;     adjusted at install time
         SUB     AX,800H                 ; this seems a safe assumption
         CMP     AX,CS:Word Ptr SEGM     ; if call came from below...
         JNC     ONWARD                  ; ... then go on to DOS
         PUSH    BX                      ; save bx
         MOV     BX,DX                   ; get pointer to pathname
;
;        first time through, check for a "/", ":" or "\", exit
;        if either one found, meaning a pathname or drive was
;        specified.
;
LOOP1:   MOV     AL,[BX]                 ; get a byte
         INC     BX
```

(Figure 6 - continued next page)

know you should, but forget and enter its name expecting it to run. DOS will obediently execute the .EXE file you just created not knowing that it'll lead to instant disaster.

Next thing you'll certainly do is reach for the BRS (Big Red Switch). Depending on whether you remember the mistake you made, you'll either panic or run EXE2BIN.

Unlike UNIX, DOS always searches the current working directory (CWD) for executable files before picking up the PATH you set (you did, didn't you?) and going there to find your commands.

In any well-kept DOS system, especially with large hard disks, you will rarely have a command file in the current directory that you actually wish to execute. Thus, DOS is wasting your precious time trying to find a file that *you know* is not there.

UNIX allows you to search the current directory at any time in the search path, or not at all. After you've run my little TSR (assuming you can assemble it into a COM file after converting it to [ugh] MASM), DOS will behave like UNIX.

The program, which I call NOLOC (NO LOCals!) intercepts all "search first" requests and checks to see whether there are any leading drive or path names in the program name. If there are, it reports "no file". The program is intelligent enough to know whether the call came from COMMAND.COM.

Of course, it's still possible to execute command files in the CWD, but it must be done explicitly by prefixing the name with "./" (oops, that's ".\" if you didn't use CSC).

You may also alter your PATH to explicitly scan the current directory, as in "PATH C:\BIN;C:\DOS;." (note the ";." at the end meaning, "search the local directory after all else has failed").

This is also an excellent means of avoiding long delays in large directories. I wrote it mostly to speed up compilation time.

NOLOC.A86 is in Figure 6.

That's Not All, But

That's it for now. If I write much more, I won't have anything else to say next time. Happy computing, everyone!

■■■

Figure 6 - Continued

```
MP      AL,'/'                          ; unixish path separator
        JZ      QUIT
        CMP     AL,'\'                  ; MSDOSish path separator
        JZ      QUIT
        CMP     AL,':'                  ; drive separator
        JZ      QUIT
        OR      AL,AL                   ; end of string
        JNZ     LOOP1                   ; go check next character
        MOV     AX,33FH                 ; loop counter, three "?" to check
        DEC     BX
LOOP2:  DEC     BX
        CMP     AL,[BX]
        JNZ     QUIT                    ; exit if not a "?"
        DEC     AH
        JNZ     LOOP2
        POP     BX                      ; restore registers
        POP     AX
        STC                             ; set error flag
        MOV     AX,18                   ; error #18 - no match
        RETF    2                       ; return to caller
;
QUIT:   POP     BX
ONWARD:         POP     AX
        JMPS    CONTI                   ; go on to DOS
;
INT21   DW      0,0
SEGM    DW      0
;
NOTV3   DB      0DH,0AH
        DB      'Requires DOS 3.xx',0DH,0AH,'$'
BANNER  DB      0DH,0AH
        DB      'NOLOC v1.00 by ESKAY is now active.'
        DB      0DH,0AH,'$'
        END
```

■■■

C UTILITIES

COMPLETE SOURCE, OF COURSE!

DOS-PACK: A disk full of useful MS-DOS programs, including a fancy C listing utility, disk sector editor and many others! ($19)

TELED Plus: Inter-system communications program, with Hayes / Zoom modem support, text capture w/ editing, MODEM protocol wildcard file transfers. Also available for MS-DOS, CP/M and ISIS-II. ($89)

VIEW: The ultimate disk utility for CP/M systems! Recovers erased files, even if your directory is crashed. Displays or modifies every sector. ($59)

ACCELER 8/16: Best of the CP/M emulators for MS-DOS. Enables PC's to run most CP/M programs, even Z-80 code! Also includes the Media Master disk conversion program. (no source code, V-20 chip included) ($89)

Request a catalog of our products!

CP/M TM Digital Research
MSDOS TM Microsoft
ISIS TM Intel Corp

Western Wares
303-327-4898
Box C • Norwood, CO 81423

C CODE FOR THE PC

source code, of course

Panache C Program Generator .	$150
QC88 C Compiler	$90
EMACS-like Editor	$75
TELE Kernel & Windows	$60
Make	$50
PC/MPX Multitasking Executive .	$45
Coder's Prolog	$45
Biggerstaff's System Tools	$40
Translate Rules to C	$30
ICON String Processing Language	$25
Bison (YACC clone) & PREP . .	$25
LEX	$25
XLT Text Translator	$20

The Austin Code Works
11100 Leafwood Lane
Austin, Texas 78750-3409
(512) 258-0785

Free shipping on prepaid orders — MasterCard/VISA

ICs PROMPT DELIVERY!!!
SAME DAY SHIPPING (USUALLY)
QUANTITY ONE PRICES SHOWN for APRIL 26, 1987

OUTSIDE OKLAHOMA: NO SALES TAX

DYNAMIC RAM			
1Mbit	1000Kx1	100 ns	$28.50
51258	*256Kx1	100 ns	6.95
4464	64Kx4	150 ns	3.50
41256	256Kx1	100 ns	4.35
41256	256Kx1	120 ns	3.50
41256	256Kx1	150 ns	3.25
4164	64Kx1	150 ns	1.30
EPROM			
27512	64Kx8	200 ns	$10.50
27C256	32Kx8	250 ns	5.15
27256	32Kx8	250 ns	4.85
27128	16Kx8	250 ns	4.10
27C64	8Kx8	150 ns	4.85
2764	8Kx8	250 ns	3.75
STATIC RAM			
62256	32Kx8	120 ns	$12.95
6264LP-15	8Kx8	150 ns	2.95

640K MOTHERBD UPGRADE: Zenith 150, IBM PC/XT, Compaq Portable & Plus; hp Vectra

* STATIC COLUMN DRAM FOR 386 COMPAQ

8087 5 Mhz $115.00
80287-8 8 Mhz $250.00

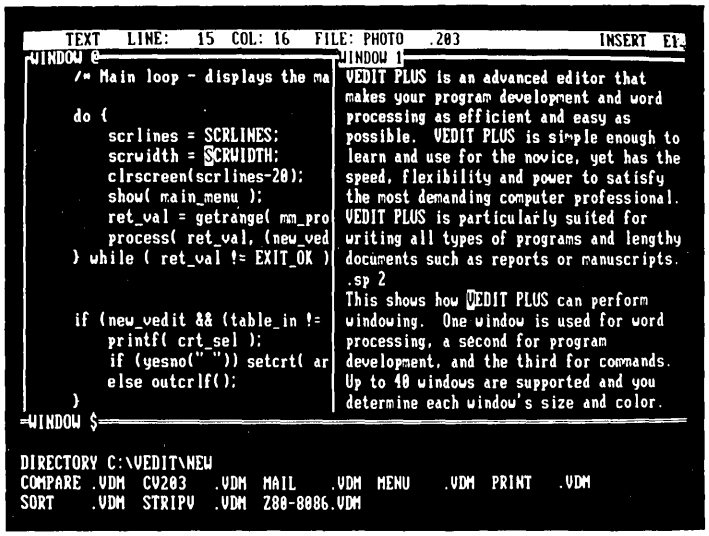
OPEN 6½ DAYS, 7:30 AM-10 PM: SHIP VIA FED-EX ON SAT.

SUNDAYS & HOLIDAYS: SHIPMENT OR DELIVERY, VIA U.S. EXPRESS MAIL

SAT DELIVERY INCLUDED ON FED-EX ORDERS RECEIVED BY:
Th: Std Air $6/4 lbs
Fr: P-One $13/2 lbs

MasterCard/VISA or UPS CASH COD
Factory New, Prime Parts μP∞
MICROPROCESSORS UNLIMITED, INC.
24,000 S. Peoria Ave.,
BEGGS, OK. 74421
(918) 267-4961

No minimum order. Please note that prices are subject to change. Shipping & insurance extra, & up to $1 for packing materials. **Orders received by 9 PM CST can usually be delivered the next morning, via Federal Express Standard Air @ $6.00, or guaranteed next day Priority One @ $13.00!** All parts guaranteed.



```
    TEXT   LINE:   15  COL: 16   FILE: PHOTO   .203                    INSERT  E1
WINDOW @                                  WINDOW 1
    /* Main loop - displays the ma  VEDIT PLUS is an advanced editor that
                                    makes your program development and word
    do {                            processing as efficient and easy as
        scrlines = SCRLINES;        possible.  VEDIT PLUS is simple enough to
        scrwidth = SCRWIDTH;        learn and use for the novice, yet has the
        clrscreen(scrlines-20);     speed, flexibility and power to satisfy
        show( main_menu );          the most demanding computer professional.
        ret_val = getrange( mm_pro  VEDIT PLUS is particularly suited for
        process( ret_val, (new_ved  writing all types of programs and lengthy
    } while ( ret_val != EXIT_OK )  documents such as reports or manuscripts.
                                    .sp 2
                                    This shows how VEDIT PLUS can perform
    if (new_vedit && (table_in !=   windowing.  One window is used for word
        printf( crt_sel );          processing, a second for program
        if (yesno(" ")) setcrt( ar  development, and the third for commands.
        else outcrlf();             Up to 40 windows are supported and you
    }                               determine each window's size and color.
WINDOW $

DIRECTORY C:\VEDIT\NEW
COMPARE .VDM  CV203   .VDM  MAIL    .VDM  MENU    .VDM  PRINT   .VDM
SORT    .VDM  STRIPV  .VDM  Z80-8086.VDM
```

VEDIT PLUS

#1 PROGRAMMABLE EDITOR

(Call for FREE DEMO disk)

Stunning speed. Unmatched performance. Total flexibility. Simple and intuitive operation. The newest VEDIT PLUS defies comparison.

Try A Dazzling Demo Yourself.

The free demo disk is fully functional - you can try all features yourself. Best, the demo includes a dazzling menu-driven tutorial - you experiment in one window while another gives instructions.

The powerful 'macro programming language helps you eliminate repetitive editing tasks. The impressive demo/tutorial is written entirely as a 'macro' - it shows that no other editor's 'macro' language even comes close.

Go ahead. Call for your free demo today. You'll see why VEDIT PLUS has been the #1 choice of programmers, writers and engineers since 1980.

Available for IBM PC, TI Professional, Tandy 2000, DEC Rainbow, Wang PC, MS-DOS, CP/M-86 and CP/M-80. (Yes! We support windows on most CRT terminals, including CRT's connected to an IBM PC.) Order direct or from your dealer. $185.

Compare features and speed	BRIEF	Norton Editor	PMATE	VEDIT PLUS
'Off the cuff' macros	No	No	Yes	Yes
Built-in macros	Yes	No	Yes	Yes
Keystroke macros	Only 1	No	No	100 +
Multiple file editing	20 +	2	No	20 +
Windows	20 +	2	No	20 +
Macro execution window	No	No	No	Yes
Trace & Breakpoint macros	No	No	Yes	Yes
Execute DOS commands	Yes	Yes	Yes	Yes
Configurable keyboard Layout	Hard	No	Hard	Easy
'Cut and paste' buffers	1	1	1	36
Undo line changes	Yes	No	No	Yes
Paragraph justification	No	No	No	Yes
On-line calculator	No	No	No	Yes
Manual size / index	250/No	42/No	469/Yes	380/Yes
Benchmarks in 120K File:				
2000 replacements	1:15 min.	34 sec.	1:07 min.	6 sec.
Pattern matching search	20 sec.	Cannot	Cannot	2 sec.
Pattern matching replace	2:40 min.	Cannot	Cannot	11 sec.

VEDIT PLUS FEATURES

- Simultaneousy edit up to 37 files of unlimited size.
- Split the screen into variable sized windows.
- 'Virtual' disk buffering simplifies editing of large files.
- Memory management supports up to 640K.
- Execute DOS commands or other programs.
- MS-DOS pathname support.
- Horizontal scrolling - edit long lines.
- Flexible 'cut and paste' with 36 'scratch-pad' buffers.
- Customization - determine your own keyboard layout, create your own editing functions, support any screen size.
- Optimized for IBM PC/XT/AT. Color windows. 43 line EGA.

EASY TO USE

- Interactive on-line help is user changeable and expandable.
- On-line integer calculator (also algebraic expressions).
- Single key search and global or selective replace.
- Pop-up menus for easy access to many editing functions.
- Keystroke macros speed editing, 'hot keys' for menu functions.

FOR PROGRAMMERS

- Automatic Indent/Undent for 'C', PL/I, PASCAL, etc.
- Match/check nested parentheses, i.e. '{' and '}' for 'C'.
- Automatic conversion to upper case for assembly language labels, opcodes, operands with comments unchanged.
- Optional 8080 to 8086 source code translator.

FOR WRITERS

- Word Wrap and paragraph formatting at adjustable margins.
- Right margin justification.
- Support foreign, graphic and special characters.
- Convert to/from WordStar and mainframe files.
- Print any portion of file; selectable printer margins.

MACRO PROGRAMMING LANGUAGE

- 'If-then-else', looping, testing, branching, user prompts, keyboard input, 17 bit algebraic expressions, variables.
- Flexible windowing - forms entry, select size, color, etc.
- Simplifies complex text processing, formatting, conversions and translations.
- Complete TECO capability.
- Free macros: • Full screen file compare/merge • Sort mailing lists • Print Formatter • Menu-driven tutorial

VEDIT and CompuView are registered trademarks of CompuView Products, Inc. BRIEF is a trademark of UnderWare, Inc. PMATE is a trademark of Phoenix Technologies Ltd. Norton Editor is a trademark of Peter Norton Computing Inc.

CompuView

1955 Pauline Blvd., Ann Arbor, MI 48103 (313) 996-1299, TELEX 701821

Laine Stump
Development Foundation of Turkey
Tunali Hilmi Cad.22
Ankara, Turkey

Trapping DOS's Fatal Errors

DOS errors aren't too bad, they're certainly clearer than those generated by most other operating systems. However, there are times when you just can't have them lighting up their own little corner of the CRT. Laine shows you how to trap the little beasties.

What would you think if you were running a nice, friendly little program on your computer and it suddenly came up on the screen with:

Ziss frizsen heimel diskzen flop!
Reizen Zein Flimmel? _

After you've thought about that for awhile, maybe you'll understand why it's very important for our Turkish software to trap out all those "Abort, Retry, Ignore?" errors that DOS throws out at you when you forget to plug in the disk or turn on the printer.

In the U.S. it may not be as much of a problem, but it's still kind of unsightly to have the messages scrolling off the bottom of the screen and ruining all your lovely window borders.

Fortunately, MS-DOS allows the programmer to specify his own "Fatal Error Handler" to replace the default handler supplied by DOS (actually supplied by COMMAND.COM, but let's not get into needless details unless they can be made to hopelessly confuse). This is done by pointing to your handler with the interrupt 24h vector. (You know, I really get tired of putting that "h" after all the numbers I write. Surely by now you guys have figured out that I only speak in Hex anyway, right?)

Many languages have built-in features to support this. For instance, in Microsoft BASCOM (no, I *don't* use it!) the "ON ERROR GOTO" command causes BASCOM's internal INT 24h handler to call your BASIC error handling code after it has converted the error codes into something easily accessible in BASIC.

Unfortunately, Turbo Pascal provides no such facility. And not only does Turbo rely completely on the DOS error handler, the only way to mix assembly (required to retrieve the error codes) and Pascal in the same procedure is to use those horrible "inline" statements, which I detest (sure would be nice if they would put in a #asm directive like Aztec C).

To top it all off, Turbo's I/O library isn't reentrant. So if you encounter an error while doing I/O (which is the only time this kind of error occurs) and happen to do something stupid (like writing an error message to the screen or asking for a reply), the I/O library will freak out and do something wonderful, like send the rest of your printer output to the screen (or send your mother-in-law to Tahiti).

It's possible, however, to write a fatal error handler for Turbo. (Though that may not retrieve a high-flying relative.)

After several lost evenings, I finally managed to work out all the bloody details, all the way from the register pushes and data segment restores right down to the bypassing of the I/O library without sacrificing portability (almost). Since it was such a bother for me to figure out, I decided I'd better put it in the magazine to save the rest of you the trouble.

Initialization

The first step of any hunk of code is initialization. In our case, this means changing the INT 24h vector and saving the contents of the DS register to a variable in the code segment. (We need access to the program's DS during the error routine, but DS will likely be pointing somewhere into DOS when the routine is entered.)

This is all simple enough, just use DOS's "Set Vector" call to point INT 24h to CS:ofs(FatalError), then store the value of "Dseg" (Turbo's name for DS) into a typed CONST. (Turbo's typed CONSTs are actually just initialized CS variables.) From now on, whenever DOS encounters some kind of hardware error, it will call our little routine instead of its own.

If we wanted to be really thorough, we could first use the "Get Vector" call to save the original, then write a RestoreFatalError at the end of the program, but I'm lazy. DOS automatically restores the vector when your program exits anyway. So even though restoring it is nice and symmetrical and clean, and all those other words you like to use when you talk about your programming style, it's just a waste of code segment space. And there's little enough of that in a Turbo program, with its 11K library and 64K total code limit. (Haven't they ever heard of far CALLs and .EXE files???)

Handler

The error handler itself (FatalError) is a bit more complicated. What needs to be accomplished is fairly straightforward. It's just that using Pascal (Turbo especially) to implement an interface that was intended to be written in assembly brings up a few problems.

At that, the explanation of what I did is all in the DOS technical manual, and you can find most everything you need to know about "what?" and "where?" either there or in my code. It's the "why?" and "how come?" that need explaining. Therefore, I'll give a short explanation of entry conditions, tasks to accomplish, and exit conditions, then get right down to the realities of torn hair and empty beer bottles.

Entry Conditions

Upon entry to the fatal error routine, the following information is available in the following places:

AX - IF bit 7 of AH = 0 THEN error is on disk and drive is in AL

IF bit 7 of AH = 1 THEN error is on device and AL is meaningless

DI - Error code of 0-12 corresponding to error messages in the array

BP:SI - points to first byte device header of erring device

Because I didn't have the time to go looking around for the original value of BP (one of the first things a Turbo procedure does is to push the value of BP onto the stack and then clobber it), I didn't use the pointer to the device header. I leave that as an exercise for those of you with just a hint of masochism in your psyche.

Tasks to Accomplish

Once we have an error, we must save all the registers, examine AX and DI, and output an appropriate message. Then, it's usually a good idea to display a message telling the operator how to correct things (turn on the printer, put the damn disk in the drive...). Finally, we wait for a reply from the user. Simple as that.

If we wanted to get real fancy, we could use DOS function 59 (hex, remember?) to get more details about what caused the error and what DOS thinks would be an appropriate action to take.

Exit Conditions

Only a single parameter is passed back from the error handler on exit. Register AL must contain a "response code" telling DOS what to do about the error. The codes are: 0 = ignore, 1 = retry, 2 = abort (and 3 = fail for DOS 3.1 and above).

Other than that, all registers must be unchanged from entry to the handler (that's why we saved them).

Problems

Okay, now to the real painful and alcohol-binge-inducing parts.

The most common problem (although the easiest to solve) was installing the inline code. Since Turbo doesn't guarantee that any registers will be saved, we will have to put some pushes at the start of the procedure to save the registers. Then, because the internal variables of all Turbo's library routines are in the program's DS, we will need some inline code to restore DS from the value we saved in DSSave during initialization.

Since registers are not directly addressable from Pascal, we must also use inline code to move AX and DI into memory variables for inspection. After the message printing and keypress waiting has finished, we will again have to use inline code to restore the registers and move the response code into register AL. Finally, because the procedure will be called through an INT instruction, it must be terminated with a RETI (Turbo procedures are terminated with a normal near RET). In addition to all of this, we will have to do a couple of seemingly stupid things with BP to counteract Turbo's use of BP as a "Frame Pointer" register.

But that's not all. For some reason you can't use local variables if you have monkeyed around with the stack (like, maybe because the local variables are allocated on the stack, huh?) so all local variables need to be declared as typed CONSTs (like ErrorType, ErrorCode, and ch). Keep that in mind if you want to expand the functionality of my little jewel.

"The only thing I want to do is put message at the bottom of the screen and wait for a character to be typed, and I have to go through all this???" Yep. That's right, big fella.

With all this trouble, is it really worth the effort? Well, that's debatable, but it doesn't matter because I've done it already, anyway.

Reentrancy

And I'm not finished complaining yet! I mentioned earlier that Turbo's I/O library isn't reentrant. Basically this is because the Read and Write procedures apparently keep a "handle" of some type stored in a static variable (instead of allocating it on the stack like a good boy should).

I discovered this problem after I finally made the first working version of FatalError. If I had a printer error, the first two characters of the string I was outputting to the printer would eventually get there, but the remainder of the string was cheerfully displayed at the console. Thanks, guys.

Since I like to write all my software so that it can run on ANY MS-DOS system (not just compatibles), I didn't really want to get by the problem by coding in BIOS calls. On the other hand, if I'm running on a compatible, I would like to be clean and consistent (and all those other good "C" words) and use the BIOS for outputting, since that's the way the rest of the program is outputting.

Fortunately, the built-in I/O routines pointed to by ConOutPtr and ConInPtr are reentrant. Unfortunately, since they are pointers to routines (not routines), they are not directly callable without using inline code. And that was the mother that necessitated the invention of WriteString and ReadChar. If you are using the IBM version of Turbo, WriteString and ReadChar will use BIOS calls; if you are using the MS-DOS version, they will use DOS function 6. Even if you create your own custom windowing routines and plug them in via ConOutPtr, WriteString and ReadChar will still work correctly.

Notice that I didn't have to do anything special for GotoXY and ClearEOL. Since they are specific to console I/O, they don't go through the file system, so we wouldn't have to worry about reentrancy even if they had problems with it.

The Finished Product

See Figure 1 for the four procedures needed for operation of FatalError - WriteString, ReadChar, FatalError, and SetFatalError. As always, feel free to modify them any way you like and use them any way that suits your fancy.

The example version does pretty much just what the default error handler does, except that it always prints on lines 24 and 25. I made it this way so that you could see how the original relates to real life. Of course, I expect you to replace the cryptic error messages (straight from the DOS manual) with plain English (or Swahili, or Portuguese, or whatever).

To use your own special little error

(continued next page)

(continued from page 53)

handler, just include all this code somewhere in your program (I keep it in an include file called MSDOS.INC with loads of other neat little MS-DOS specific routines), then call SetFatalError once right at the beginning of the program.

Reflections

Writing the inline code was really quite straightforward, with a bit of help from SYMDEB, once I figured out exactly what code was compiled by a "begin" statement. After I found a DOS manual that had "the rest of the story" on error routines, it was quite simple to decide what message to put with which code.

The one part that should have been completely unnecessary (and that I was offended at being forced to do) was the writing of the WriteString and WaitChar procedures to overcome the non-reentrancy problem of Turbo's I/O library. There is just no excuse for that. Frankly, my dear, I'm appalled.

A Hint

A technique that I used quite often while I was figuring out all the details behind this (and have used quite a bit in the past as well) was to run Turbo under SYMDEB (or DEBUG) and put an "inline($CC)" in the code at some stategic place to cause a breakpoint out to SYMDEB. By doing this, I could get a look at exactly what code Turbo is generating (and my, oh my, what sloppy code it is...).

If you try this, you should remember that putting a breakpoint right after "begin" will not set a breakpoint on the very first instruction of the chosen procedure - "begins" generate code, too. If you really want to look at everything involved with a certain procedure or function, set a breakpoint immediately before a call to the function. Then you can also see the code which sets up the arguments.

Also, if you don't replace the CC (INT 3) with a 90 (NOP), you may stay on the breakpoint forever. If you want to keep the breakpoint there, but don't want to keep saying "rip=ip+1;g" all the time, you can just replace it with a NOP and then set a SYMDEB breakpoint at the same address with the BP

Figure 1 - Fatal Error Handler

```
TYPE
      string128 = string[128];

      Registers = RECORD
      CASE INTEGER OF
            1:(AX,BX,CX,DX,BP,SI,DI,DS,ES,Flags : INTEGER);
            2:(AL,AH,BL,BH,CL,CH,DL,DH : BYTE);
            end;            { Registers }

{---------------------- WriteString -----------------------}
{     output a string to the console w/o going through standard  }
{     WRITE procedure. This procedure is to be used during        }
{     interrupt service routines to avoid reentrancy problems in }
{     WRITE.                                                      }
{---------------------------------------------------------}
PROCEDURE WriteString (Str : String128);

VAR ct : INTEGER;

begin
IF Length(Str)  0 THEN
            FOR ct := 1 TO Length(Str) DO
            inline($8B/$76/<Ct/                 { MOVSI,CX }
                  $FF/$B2/Str/                  { PUSH [BP+SI+Str] }
                  $FF/$16/ConOutPtr);           { CALL word ptr [ConOutPtr] }
end;        { WriteString }


{----------------------- ReadChar ------------------------}
{     reads a char from console w/o going through standard READ      }
{     procedure. This procedure is to be used during interrupt  }
{     service routines to avoid reentrancy problems in READ          }
{---------------------------------------------------------}
FUNCTION ReadChar : CHAR;

begin
inline($4C);                  { DEC SP }
inline($FF/$16/ConInPtr);                       { CALL word ptr [ConInPtr] }
inline($88/$46/$04);                            { MOV       [BP+ReadChar],AL }
end;        { WriteString }


{---------------------- FatalError -----------------------}
{           Interrupt 24H error-handling routine.   p. 1-21         }
{---------------------------------------------------------}
CONST DSSave : INTEGER = 0;

PROCEDURE FatalError;

CONST
      ErrorType : INTEGER = 0;                  { register AX }
      ErrorCode : INTEGER = 0;                  { register DI }
      ch          : CHAR          = ' ';

ErrorMessage : ARRAY [0..12] OF STRING[30] =
      ('Disk is Write Protected',
            'Unknown Unit',
            'Drive Not Ready',
            'Unknown Command',
            'Bad CRC',
            'Bad Request Structure Length',
            'Seek Error',
            'Unknown Media Type',
            'Sector Not Found',
            'Printer out of Paper',
            'Write Fault',
            'Read Fault',
            'General Failure');
```

Figure 1 - Continued

```
begin
{ PUSH all registers , enable interrupts }
inline ($50/$53/$51/$52/$56/$57/$1E/$06/$FB);

inline($2E/$A3/ErrorType);                    { MOV CS:[ErrorType], AX }
inline($2E/$89/$3E/ErrorCode);                { MOV CS:[ErrorCode], DI }
inline($2E/$8E/$1E/DSSave);                   { MOV DS,CS:[DSSave] }

GotoXY(1,24);
IF ((ErrorType and $8000) = 0) THEN
         WriteString('Disk Error - '+ErrorMessage[ErrorCode]
         +' on Drive '+chr((ErrorType and $FF)+ord('A')))
ELSE
         WriteString('Device Error - '+ErrorMessage[ErrorCode]);
GotoXY(1,25);
WriteString('<I>gnore, <R>etry, <A>bort ? ');
REPEAT
         ch := ReadChar;
         UNTIL (UpCase(ch) in ['I','R','A']);
CASE (UpCase(ch)) OF
         'I': ErrorType := 0;
         'R': ErrorType := 1;
         'A': ErrorType := 2;
         end;        { case ch }

GotoXY(1,24); ClrEol;
GotoXY(1,25); ClrEol;

{ restore all registers }
inline($07/$1F/$5F/$5E/$5A/$59/$5B/$58/$8B/$E5/$5D);

inline($2E/$A0/ErrorType);                    { MOV AL, CS:[ErrorType] }
inline($CF);                                  { IRET }
end;                                          { FatalError }


{-------------------- SetFatalError ----------------------}
{      Enable Int 24H error-handling routine called FatalError }
{-------------------------------------------------------}
Procedure SetFatalError;

var Reg : Registers;

begin
Reg.AH := $25;      Reg.AL := $24;
Reg.DS := CSEG; Reg.DX := Ofs(FatalError);
MsDos(Reg);
DSSave := Dseg;
end;                { SetFatalError }
```

End of Listing

■■■

command. Oh, yeah, I keep forgetting about all you lowlifes who only have DEBUG - you'll just have to suffer through without permanent breakpoints.

But more on SYMDEB and others another time.

Next Issue

I'm really getting tired of all these limitations and "almost bugs" of Turbo Pascal. It keeps presenting me with little obstacles which, although usually solvable, waste a lot of my time. Just to see if it's ever going to get any better, I think I'll try writing a Fatal Error Handler module for Logitech Modula for next time. I hope I'll be able to show a couple of other cute little tricks with Modula, too. On the other hand, maybe I'll send a documentary report from Mozambique on the migration of the Great Desert Whale. Ya' just can't never tell...

■■■

NEW FOR YOUR BIGBOARD

ONE MEGABYTE RAM DISK ON THE STD BUS Includes: STD adapter pcb & connector, 1 MB RAM pcb & connector and software. Price $125.00
INTEGRATED BIOS, reads and writes any 5" and/or 8" disk format.

INCLUDES:
- CONFIGuration program that lets you install any new floppy disk format INTERACTIVELY
- FORMATTER allows you to format almost any diskformat.
- PC-COPY reads and writes PC diskettes on your Bigboard II.
- MONITOR EPROM with serial keyboard and translate table.
- 300 page ZCPR2 manual.
- 60 page Bigboard II tech manual.
- BOOTABLE DISK contains free ZCPR2 and P2DOS system. With TIME and DATE stamping. BIOS also has provisions for 256K RAM disk, Centronics, System in EPROM AND 1 MBYTE RAM DISK.
- WINCHESTER FORMATTER and SYSGEN. Supports XEBEC, W-D and Adaptec type controllers. Subdivides into any specified number of drives. Price: $99.95 (specify disk)

TAKE BOTH 1 MB AND BIOS FOR
$199.95!!

ANDY BAKKERS
de Gervelink 12 • 7591 DT Denekamp
The Netherlands • Tel: 31-5413-2488
FIDO Net 500 Node 100. Please pay with US$ Money Order. MC or VISA welcome

By Ron Miller
1157 Ellison Dr.
Pensacola, FL 32503

Interrupting A PC From C

We interrupt this magazine to bring you this epistle on interrupts. Ron tackles the creation of generic interrupt routines in C along with C standards and other C things.

From the number of letters and calls I received after my recent column on graphics routines, I've concluded two things. First, lots of you are intrigued by the possibility of doing your own graphics programming on clones. Second, even more of you are trying to adapt patches of code in self-defeating ways. You're not alone.

When even Pascal Peter Norton proclaims C to be the wave of the future, and when Borland's Turbo C is about to appear out of the misty midregions of Vaporland into the mail order houses (at $70 a throw), C can no longer be written off as the sole domain of software engineers and hackers who count the days to Christmas in hexadecimal. (The other morning I opened my refrigerator and counted the eggs on hand, starting with zero - it was then I knew I'd been playing with C too long.)

Joys of Standardization

Like every other high level programming language, C was designed to free the programmer from thinking about the mechanics of the computer. Anyone who has read Kernighan and Ritchie will recognize the great pains taken to free the syntax from the system's architecture. Although integers can be 16-bits or 32-bits or 64-bits long on a specific machine, the programmer who writes:

```
int i,v[10];
for(i=0;i;i++) v[i]=i*i;
```

can be assured of getting 0,1,4,9,16 and so forth. There is, moreover, a set of standardized functions which effectively hide the hardware.

Though a terminal hooked to a Cray and a 40-column display wired to an Apple IIe are somewhat dissimilar, similar keyboard responses to a getchar() will produce similar squiggles on the screens.

This, of course, is what is meant when we talk of the "portability" of C programs. Fortunately, C compilers are getting better about providing a fairly large standard library, all functions accepting the same arguments, in the same order. And they do the same things.

The Sorrows of Individuality

Unfortunately, the ANSI standards have no control over machine-specific functions, but of course this problem isn't unique to C.

Consider the IBM BASIC "color" command. Try porting that one over to the BASIC on your old Kaypro II.

Obviously, folks with color need some control. Although we hacker types might prefer to get in there and attack the attribute bytes in video RAM, sane folk just want a way to get the light show going. So Microsoft decided that "color" is an easily-remembered word, and wrote the interpreter accordingly.

Somebody getting up another version of BASIC might decide, as did Borland with their Pascal, that "textcolor" and "textbackground" would be more useful and/or descriptive. Thus incompatibilities are born.

Those who construct C compilers usually solve the "color" problem by ignoring it. Therefore, the programmer who wants to put on a show must pull out his IBM Tech Manual, slog through the pseudo-English to figure out what BIOS calls to make, and roll his own function - which he can christen color(), or text_tint(), or elvira(), or whatever().

This is a delight for the hacker, but death to the casual programmer. (The world's too full of casual programmers anyway.) The latter has come to C from the warm shores of Pascal or BASIC. He buys a humongous $450 compiler from Microsoft, and then searches in vain through four volumes of abstruse documentation for a reference to a simple routine to turn his screen display green. I'll bet that two-thirds of the C compilers purchased by readers of Micro Cornucopia have been put away, never to be used again, for that simple hangup.

(Editor's note: Here is where the Kaypro II folks have a tremendous advantage. Their screens are already green.)

Interrupts

One unprescribed feature provided by every MS-DOS C compiler is a call-to-the-operating system or "interrupt" function. Strictly speaking, it's not really necessary, if you can link in assembly language. But austerity has its limits, even for those fleeing the junkiness of Turbo Pascal and its "moveturtle" procedures.

Since C doesn't acknowledge so parochial a routine (it acknowledges the peculiar structure of the 86 family), the software houses dream up their own names and formats. I've seen dosint(), interrupt(), int86(), and msdos(). I've also seen one integer, one integer and a pointer, and one integer and two pointers as arguments for these functions.

Behind each implementation is a defensible rationale and a bunch of assembly language. Not to be outdone, I plan to offer another at the end of this column. Let a hundred flowers bloom.

As several of you have discovered, this unweeded garden of function design can be very frustrating if you want to recompile Old Joe's serial port code.

So you sit down with the listing nestled under your CRT, type it all in, letter for letter, and press the button. With luck the compiler says "Huhh??" or "Unresolved Global." Worse than that, the compiler may think it understands you and freeze up tighter than January.

Pity the poor techno-journalist. What can I do when I'm writing - as I usually am - about operating system manipulations in C? I've got to tell you what to do. Yet, if you're unfamiliar in these matters, you'll be bewildered when you look through your documentation and find terms like "regs.h" and "interrupt()." The names I give my calls to the operating system will mean nothing to you. Nothing at all. The time I saved by not explaining all this up front has cost you a lot of frustration.

So let's look at operating system calls and how to design our own. The exercise should help those of you new to C to understand what is going on in the ready-made functions you have inherited from your compiler.

Underlying Assumptions

Long ago, when I was playing chemistry student, I learned that only by understanding the assumptions behind the Second Law or the Boltzmann Distribution could I apply them to practical problems.

Also, this project is a fine example of the roll-your-own approach of C and the trade-offs in function design.

If you're not a do-it-yourselfer, find another language - one that shields you from the nuts and bolts. As Chaucer puts it, "Turne over the leef and chese another tale." Those of you remaining may understand better why C lies somewhere between boiled shrimp and dogwood blossoms in my garden of earthly delights.

Software Interrupts

Except for the most metallic of to-the-bare-metal programming, MS-DOS programs communicate with the operating system and the hardware through interrupts. These are indirect calls to routines located in the MS-DOS core or the transient-but-stay resident heap or the BIOS ROM. The addresses of these routines are stored in the interrupt table, an array of 256 32-bit numbers (offset plus segment) located at the base of the memory.

Whenever you read or write to a file, enter characters from the keyboard, or send text to the printer, your code uses interrupts to instruct the operating system.

So common is this activity that these indirect calls are hard-wired into the Intel instruction set. Within the machine code, a hex character CD, followed by a one-byte integer (00 to FF), will cause the CPU to execute a special "long" call to the routine whose address is stored at the corresponding address in the table. Thus a call to interrupt 21h (CD21) causes the computer first to trot out 132 bytes from the base of the memory (132 = 21h X 4). It then plucks the 4-byte address, pushes the flags, the code register, and the instruction pointer onto the stack, and finally jumps to the new location.

In assembly language it's trivial:

```
int 21h
```

In practice, the assembly language programmer must set a few registers to get the right results. For example, if the letter "G" is to be sent to the screen, the AH register is set to 2 - the function number desired - and DL is loaded with the number equivalent to the letter "G," like so -

```
mov ah,2
mov al,'G'
int 21h
```

Return values, if any, are also stored in the registers.

Doing It In C

Suppose you want to do something not available in the meat and potatoes section of the standard library. If you don't wish to practice assembly language linking, and don't own a version of C that allows in-line assembler, here's what you do.

You use a generic C function that can load any register, call any interrupt, and recover the returned register values (as C variables). See Figure 1.

Of course, the generic routine will be overkill because it'll load and save everything as well as preserving all the register values.If you are going to roll-your-own function, you need to read about "assembly language interfaces" in your compiler's documentation. Or, you might have your compiler generate assembler output for a nice, simple C program.Then pull out your editor to see which registers the compiler pushes and pops at the beginning and end of each function. Most compilers I've examined would be very surprised if DS and BP were creamed by a function. Another compiler doesn't give a hoot about BP but wants ES, DI, and SI preserved.

I never said it would be simple. Just fun.

Perhaps these gritty details will also explain why a C program is always slower and larger than an equivalent assembly language version. If I were doing this in assembler, I'd only bother with the registers that were being used for the specific interrupt. However, that means I have to redesign every assembly language interrupt call.

Designing The Function

Now let's get down to coding. The obvious way to handle register values in a C program is to create a structure. Make integer spaces for AX, BX, CX, DX, DI, SI, and so forth.

We could use one structure or two. We might use one for the registers-in set and one for the registers-out. Having two can be a convenience when subsequent code must know what went into the function call. However, two structures take twice as much memory, and I find that I can always declare local variables to preserve "in" values if I need them. Let's go with one.

The structure(s) must be declared so that you can cram the right value of AX into the AX slot, BX into the BX slot, and so forth. Getting this quite arbitrary order the same every time is best accomplished with an included file. Thus the "#include <regs.h>" in my examples. For the interrupt function given in the listing below, the contents of "regs.h" would be -

```
struct
regs{intax,bx,cx,dx,di,si,bp,es,ds,flags;
   };
```

(continued next page)

(continued from page 57)

Pulling together the themes from the paragraphs above, we can see that the projected interrupt function must be constructed like so -

1. Push the registers that must be saved onto the stack
2. Load the registers with values from the structure
3. Make the interrupt call
4. Reload the structure with the new values from the registers
5. Pop the register values that must be preserved back off the stack

You get one more choice about your structure. You can declare your own structure and reference it with a pointer when calling the interrupt function or you can allocate space for a global structure.

The second choice means that there would be one less argument for the function - at times, a convenience. You pay a price with this approach, however. First, your .EXE file will probably be larger. Second, you can get into trouble if the function is used in complex resident programs where interrupts can interrupt interrupts. If two interrupt calls use the same register bin, the intruding interrupt could change the register values of its predecessor.

It can happen. DeSmet's C employs the global-structure technique, and a week or so ago I spent a couple of miserable hours staring at a terminate-but-stay-resident program that looked correct but was locking up in weird ways. I finally realized that my keyboard I/O interrupt handler was having its registers altered 18.2 times a second by my clock interrupt handler. (Try making an interrupt 16h, function 233h call some day.)

So let's use an explicit pointer and require that the user allocate a suitable structure. See Figure 2 for the assembly language listing for a version of C that -

1. Uses long or 32-bit pointers
2. Insists upon preserving DS and BP across the function
3. Treats the stack this way at the first of a function call:

- Push the function arguments
- Push CS and IP for a "long" call
- Push BP
- Move SP into BP as a base pointer for the stack

Under these circumstances the function arguments will begin at [BP+6] on the stack.

Figure 1 - Generic Interrupt Function In Assembly Language

```
interrupt(nn,ptr)
   int nn
   struct regs *ptr;    /* In this case a long pointer: 32 bytes */


/**
In a short C the pointers would be words, not double words, and you'd
use a ``mov bx,[bp+??]'' rather than the ``lds bx,[bp+???].'' In all
cases remember to recover DS before using BX to point to the base of
the structure being reloaded with register values.


Also note the rather sneaky way of generating the machine code
``CDnn'' patch that invokes nnth interrupt. Through an irritating
omission in the Intel instruction set, ``int'' command can only take an
``immediate'' or numerical argument; and therefore the assembly
language programmer must modify the code <i>in situ<d>. Not very
nice, and certainly not ROM-able, but necessary. What I do below is
load CD into the AL register, nn into the AH, and then poke the word
into the code at the proper location. Since the Intel chips use
byte-reverse layout, the code becomes CDnn.  Blush . . . but it works.


ASSUME PROPER SEGMENT DECLARATIONS

   push  bp                 ;Save bp
   mov   bp,sp              ;Establish a pointer to the stack
   push  ds                 ;Push whatever other registers need to be saved
   mov   al,0cdh            ;0CDh into low byte, int # into high
   mov   ah,[bp+6]          ;This displacement will vary with the compiler
   mov   cs:intcall[0],ax   ;Stuff CDNN into the instruction site
   lds   bx,[bp+8]          ;Load seg:ofs of pointer to structure into ds:bx
   mov   ax,[bx+0]          ;Move first slot of structure into ax
   push  [bx+2]             ;Push final bx onto stack, since bx needed as
                            ;   pointer right now.
   mov   cx,[bx+4]          ;Load the rest of the registers, skipping bp
   mov   dx,[bx+6]          ;   since bp is never used by interrupts and is
   mov   di,[bx+8]          ;   needed to point to stack.
   mov   si,[bx+10]         ;We can ignore flags on the way IN, not OUT
   mov   es,[bx+14]
   mov   ds,[bx+16]         ;There goes the pointer
   pop   bx                 ;Pop bx off stack
   push  bp                 ;Save bp across interrupt


 intcall:
   dw    00                 ;Scene of the crime of self-modifying code
   pop   bp                 ;Recover bp
   push  bx                 ;Save ds & bx so we can use a pointer again
   push  ds
   lds   bx,[bp+8]          ;Reload pointer
   mov   [bx+0],ax          ;Inverse of above: fill struct with values
   mov   [bx+4],cx
   mov   [bx+4],cx
   mov   [bx+6],dx
   mov   [bx+8],di
   mov   [bx+10],si
   mov   [bx+14],es
   pushf                    ;Preserve flag value this time
   pop   [bx+18]            ;When you can't point, pop
   pop   [bx+16]
   pop   [bx+2]
   pop   ds                 ;Recover saved registers
   pop   bp

          ......................................................
```

Figure 2 - A Goodie For Terminate-But-Stay-Resident Hackers

```
Since this column has paid tribute to Back To The Basics, perhaps you
more advanced codeslingers might appreciate a trick I just discovered.
This ``pointer'' technique, in tandem with the peculiar way that C
handles function call arguments, provides an extra benefit for C
programmers who are writing interrupt handlers.  Suppose you had
stolen an interrupt and put in its place in the interrupt table the
address of an assembly language function that does no more than this
--

1. Push the registers in the order flags,ds,es,bp,si,di,dx,cx,bx,ax
2. Call your interrupt handler WRITTEN IN C
3. Pop the registers in the order ax,bx,cx,dx,di,si,bp,es,ds,flags


Notice that the stack at the time the C program is operating will
therefore be laid out in precisely the order ordained by ``regs.h.''
Since pushing variables onto the stack in C (as opposed to almost
every other language) is the duty of the calling function, not the
called, a function compiled by a C compiler will treat whatever
happens to be on the stack as if it contained the variables defined
by its own arguments. Therefore if you give your interrupt handler a
fake argument, i.e. --

      handler(dummy)
         int dummy;


-- the function will treat the stack space occupied by the pushed AX
register as a local variable. If we point our ``struct regs'' pointer
to the ADDRESS of this variable --

      struct regs *rr;
      rr = &xx;


-- we effectively treat the stack itself as though it were a
structure that can be used as a storage area by an interrupt calling
function within the interrupt handler. If you have a zippy new
version of C that allows passing whole structures on the stack,
things can be even more elegant.
The benefit is that you can use the pointer-to-the-stack technique to
pass register values back and forth to the calling function.
Suppose, for example, you had stolen interrupt 16h and decided to use
a whole passel of hotkeys to carry out a number of resident program
tricks. The interrupt handler might look something like this --


handler(dummy)
  int dummy;
{
 struct regs *rr;
 char fnct;
 rr = &dummy;    /* point structure pointer at base of register stack */
 fnct=rr->ax>>8;  /* store what's in AH so we can check the function call*/
 interrupt(new16h,rr);  /* call original interrupt 16h BIOS routine */
 if(fnct == 0) switch(rr->ax){  /* we're only interested in function 0 */

   case HOTKEY1: do_1();  /* check for hotkeys and act accordingly */
                 break;
   case HOTKEY2: do_2();
                 break;
   case HOTKEY3: do_3();
                 break;
     etc. etc. etc.
   }
 }        /* return after stack popping to the calling program */


For the non-heroes among us, doing the work in C rather than in
assembler saves hours of debugging.
```

Compilers definitely differ in this regard. The only way to find out about yours is to set up the compiler for assembler output and see how the compiler addresses the variables pushed onto the stack. Try it.Even if you view this exercise as an example of wretched excess, you will learn a lot about how C works.

You'll also see that the overhead will change as you select small, medium, or large compiler options.

Doesn't Apply To You

"What If I Promise Never To Do This Sort Of Thing, Ever?"

OK, now that you've seen it, you've sworn off assembly language modules. Fair enough. There is, however, a practical and (I dare say) moral aspect to this.

On the practical side, you should have at least a notion of how your compiler's interrupt function works. Moreover, the function we designed is representative of many of the (standard and nonstandard) functions that come with each compiler. No language, no matter how stratospheric, rewards zombie programming. And, the flexibility and power of C makes it singularly unforgiving, a fact lots of folks will discover as Borland's Turbo C reaches the market.

The moral dimension is even more important. Without understanding, we can't appreciate the work that has gone into these routines. It's an unthinking person who thinks that milk magically appears in plastic-coated cartons at the supermarket - while others awaken at 3:30 a.m. to herd grumpy cows into milking machines, and shovel manure.

Software piracy would be much less common if we could see the human face hiding behind those fancy functions we invoke so effortlessly. Also, how many of us, I wonder, ever contemplate the sheer organizational effort that it took to coordinate all those cycles and epicycles within Lotus?

■■■

Trygve Lode, President
Lode Data Corporation
6450 E. Hampden Avenue
Denver, CO 80222

In-House Experimental Verification Of Nonconservation Of Parity And Quantum Mechanical Tunneling Of Macroparticles

Abstract
Experiments verifying macroparticulate parity nonconservation and macroparticulate quantum mechanical tunneling are discussed.

Introduction
The nonconservation of parity has long been observed in weak interactions, and quantum mechanical tunneling is a frequent event in radioactive decay; however, no significant research has been conducted to determine whether similar processes occur involving macroparticles.

Even general macroparticulate quantum effects have heretofore been ignored by the scientific community. This, in all probability, is due to the uncanny and disturbing resemblance macroparticles bear to actual physical objects, a drawback which frightens off all but the bravest of theoreticians. To assist in the amelioration of the relative dearth of knowledge in this field, it was decided to conduct two experiments to determine if quantum mechanical processes occurred in macroparticles: the first would determine if parity was conserved; the second would attempt to discover tunneling effects.

Because macroparticles do behave so much like actual objects, it was necessary to conduct all experiments as far from physics laboratories (1) as possible. Certainly the most convenient location satisfying this requirement was my house. So all experiments were conducted in-house.

Experiment 1: Nonconservation Of Parity
To demonstrate the nonconservation of parity in macroparticles, it was first necessary to have a group of macroparticles on which to experiment. As the macroparticles best suited to parity experiments, I chose socks (2). The socks chosen were size thirteen, black, over-the-calf men's dress socks purchased from a local clothing emporium. Several pairs were purchased at one time; their average mass was 38g per sock. They went through a two-stage (3) purification process and were removed from the washing machine two at a time to determine that they were indeed still in pairs.

Next, the macroparticles were loaded into the macroparticle dehydrator/storage-cylinder accelerator (4) which accelerated the macroparticles to 6.5 x 10E5 +/- 2.1 x 10E5 TeV and heated them to approximately 347 degrees Kelvin. They remained in the storage cylinder for 1561.1 +/- 0.4 seconds and then were removed en masse and placed in a drawer (5).

Each day, over a period of about two weeks, one pair of macroparticles was removed from the drawer, worn (6), and set aside for future recycling. At the end of the experiment, when all pairs had been removed, a single sock remained in the drawer - the group of macroparticles had changed parity from even to odd. This experiment was repeated a total of four times, and in three of the four trials, parity was not conserved.

Experiment 2: Macroparticle Tunneling
Discovery of the event that led up to this experiment came about entirely by accident: one morning several plates, bowls, and pieces of stainless flatware (these will be hereinafter referred to as Kitchen Macroparticles, or KMPs) appeared in the basement, clustered about the television set, which is directly beneath the kitchen where these KMPs would normally be found.

When questioned regarding this curious event, all proximate mini-persons (7) denied moving the KMPs or even being aware of their presence in the basement. I began a controlled experiment to determine if these KMPs were indeed tunneling through the relatively high potential barrier of the kitchen floor to the lower energy state of the basement.

First of all, all KMPs were removed from the basement, washed (8), and placed in cupboards. Proximate mini-persons were carefully instructed not to take any KMPs outside of the kitchen. The following evening (9 hours later), a thorough examination uncovered a total of fourteen KMPs in the basement distributed in a roughly Gaussian pattern around the television set (which, as you will recall, is directly below the kitchen).

There were two bowls, six plates, three spoons, two forks, and a knife. The thickness of the floor was measured to be 22.4 cm, which suggested that the

KMP wavelengths must be roughly on the same order of magnitude. The individual KMPs were measured and they ranged from 15.5 cm to 28.1 cm with a mean length of 19.3 cm, correlating remarkably well with the estimate based on floor thickness.

A closer examination revealed that every single KMP exhibited signs of recent contact with comestibles, although a relatively small quantity of actual edible material remained adhered. Perhaps most significantly, the material adhering to the KMPs was invariably food which apparently had been heated (soups, microwave quick-lunches, leftovers, ice cream soup, etc.); no unheated edible material (twinkies, cookies, etc.) adhered to the KMPs. We may therefore conclude that greater than ambient thermal energies are required for quantum mechanical tunneling.

Foods similar to those adhering to the KMPs were discovered spilled in the kitchen, strongly suggesting that the foods which heated the KMPs had been unable to tunnel through the floor themselves either because of shorter wavelengths or a lesser effect of gravity on food than on dishes.

Previous experiments had shown that, in fact, the force of gravity has a stronger effect on food than on dishes (9,10). So I suspected the former possibility, a suspicion which was confirmed when the spills were measured and all were found to be under 4 cm. Comestible fragments that remained adhered to the KMPs, on the other hand, were generally at least 10 cm in length, much more capable of tunneling through the floor.

Finally the distribution of food-heated plates (the most common KMP found) confirmed the tunneling hypothesis: 9 were found in the kitchen, 3 were found on a table near the television, and 1 was found under the table (11) - coinciding almost exactly with the exponential distribution expected for KMPs not tunneling, having tunneled through the floor, and having tunneled through both the floor and table.

I also discovered that a spoon was missing altogether which I assumed must have passed through the Earth entirely. Several calls to Hong Kong Universities failed to uncover the location of the wayward spoon, so this has not as yet been confirmed.

Conclusion

It has been conclusively demonstrated that the parity of macroparticles is not conserved. Therefore socks must come in right-left pairs rather than the single type invariant under reflection operations as was previously supposed.

Similarly, it has been shown that macroparticles of greater than ambient thermal energies are easily capable of tunneling through potential barriers such as a kitchen floor, and that the first-floor metastable state has an approximate half-life of 16.6 hours.

The discovery of quantum effects in macroparticles may be the single most important development in quantum mechanics since the Schrodinger equation, but research in this field is far from over. We still need to know the relative probabilities of appearance and disappearance of socks and whether the universal sock population remains constant.

We need to calculate macroparticle tunneling half-lives with a greater degree of accuracy, and we still need a clearer determination of the effects of temperature on macroparticle tunneling. For example, my cans of soft drinks are forever disappearing from the office refrigerator. The fact that they are cold suggests parity effects at work, but the fact that they always vanish and never appear suggests the effects of tunneling.

Perhaps most importantly, I still need a grant or a Nobel prize or something, which clearly indicates the need for further research.

Notes

1. Interactions of objects and researchers under laboratory conditions bear at best only the most superficial resemblance to their real counterparts (cf E.P.A. highway gas mileage estimates).

2. Socks come in pairs and are much cheaper than shoes.

3. The first stage involved removing labels, price tags, and those little plastic hooks. The second stage consisted of running the socks through the medium load cycle of a Speed Queen (R) washing machine with Tide (R) detergent.

4. Speed Queen (R) heavy-duty electric clothes dryer.

5. Approximate capacity 35,000 cc.

6. One macroparticle was placed on each foot. Feet were carefully counted each day to confirm that they had not also changed parity.

7. Juvenile Homo Sapiens, age 11-17 years.

8. Using a Kitchenaid (R) dishwasher and Cascade (R) dishwasher detergent.

9. R. C. Rutabeta, *Generalized Theory of the Buttered-Side Effect,* J. Recalcitrant Foods 4 (1981), 1630-1661

10. H. B. Rosie and N. Freap, *Experimental Verification of the Diner Effect With Particular Emphasis on the Comparative Analysis of Paper Towel Absorption Coefficients as Affected by Television Camera Proximity,* Murphy's Legal Journal 186282 (1992), 62431-62432

11. The other two plates were on the floor away from the table, and so were not included in this analysis as they might adversely affect the results.

■■■

TECHNICAL TIPS

Recovering Trashed Disks

In response to your very helpful article in Micro C #32 ("Recover A Directory By Reading & Writing Disk Sectors"), I would like to inform you of a relatively simple strategy for recovering files. It is based on your assembly language shell.

Problems frequently occur with initial read errors of diskettes. The cause may be unreadable signals written on the boot area, FAT area, or portions of the directory. Norton Utilities won't read these disks since DOS can't swallow the errors.

However, you can fool DOS by logging onto a good disk with Norton Utilities or PC Tools. Then use the "inspect/change sector" option to get into the data area and swap the bad disk into the drive. By this method, you can easily see what areas have been trashed and work out a recovery strategy. This might consist of using one of the FATs as a template to reconstruct the other or analyzing the FAT to put together a new directory.

The worst case is where both FATs and the directory have been trashed. Here is where your program really comes into its own. In my case, I was able to copy the boot record, FAT area, and directory from a good disk in B to my trashed disk in A. The good disk had one very large file taking up all but a trivial amount of the disk.

Using Norton Utilities, I substituted "$" for each end of file mark. The single large file was then copied to the hard disk and the original files were broken out with a word processor.

One hitch is that any bad sectors on the original disk are now allocated to the large file. But I found that DOS would skip over most of these with an "I" (Ignore) response to its error message.

This is a somewhat cumbersome procedure, but with it I was able to recover material that previously would have been lost. Thank you for your original article and keep up the good work.

J.R.E. Harger
UNESCO
Office For Science and Technology
Tromolpos 273/Jkt.
Jakarta, Indonesia

DOS 3.2 and 3.3

I've had a problem with flaky disk drives on my AT clone. The drives only live for five or ten minutes using MS-DOS 3.2 at 10 MHz. However, PC-DOS 3.2 works fine at 10 MHz. My controller is the Western Digital card with the cables coming off the side.

Speaking of drive problems, I talked to Dave over at Cascade Electronics recently. He said that the Phoenix BIOS couldn't format 1.2 meg under MS-DOS 3.2, but could with 3.1 or PC-DOS 3.2.

Dave sent me a copy of an IBM publication called *Exchange* which included a list of changes for PC-DOS 3.3 over 3.2:

1. PS/2 support.
2. Support for 1.44 meg 3 1/2 inch drives.
3. Up to four async ports at 19,200 baud.
4. Time and date function to set cmos clock.
5. Enhanced national language support.
6. Better batch file processor.
7. Append, a memory resident utility which is like PATH but for non-COM files. Also in 3.2.
8. Backup will format disks.
9. Call, a new batch file command which allows one file to call another.
10. FASTOPEN: TSR which will cache directory for non-removable drives.
11. MODE which will support the four comm devices.
12. RESTORE which will support new disk format and allow selective file recovery.

Alan Gomes
11751 Holly View Dr.
La Mirada, CA 90638

Western Digital Controllers

A simple modification (just a little trace cutting and jumpering) plus a new EPROM will allow the standard Western Digital hard disk controller card to work in a Tandy. The EPROM and instructions go for about $20. Contact:
Technoland
5830 E. Washington Blvd.
City of Commerce, CA 90040Z
(800) 222-3978

7 MHz Problem

I'm writing to you on my sped-up Kaypro (see Micro C issue #33 -- Kaypro Column). It works very well, but I'd like to tell you about one very thorny problem I had with this particular machine.

It concerns the IC at U2. Your article and diagram specify a 74LS04. With that chip in place, the computer refused to operate. Both drive lights came on and I had no video. When I replaced the 74LS04 with the original chip, a 74HCU04, it worked.

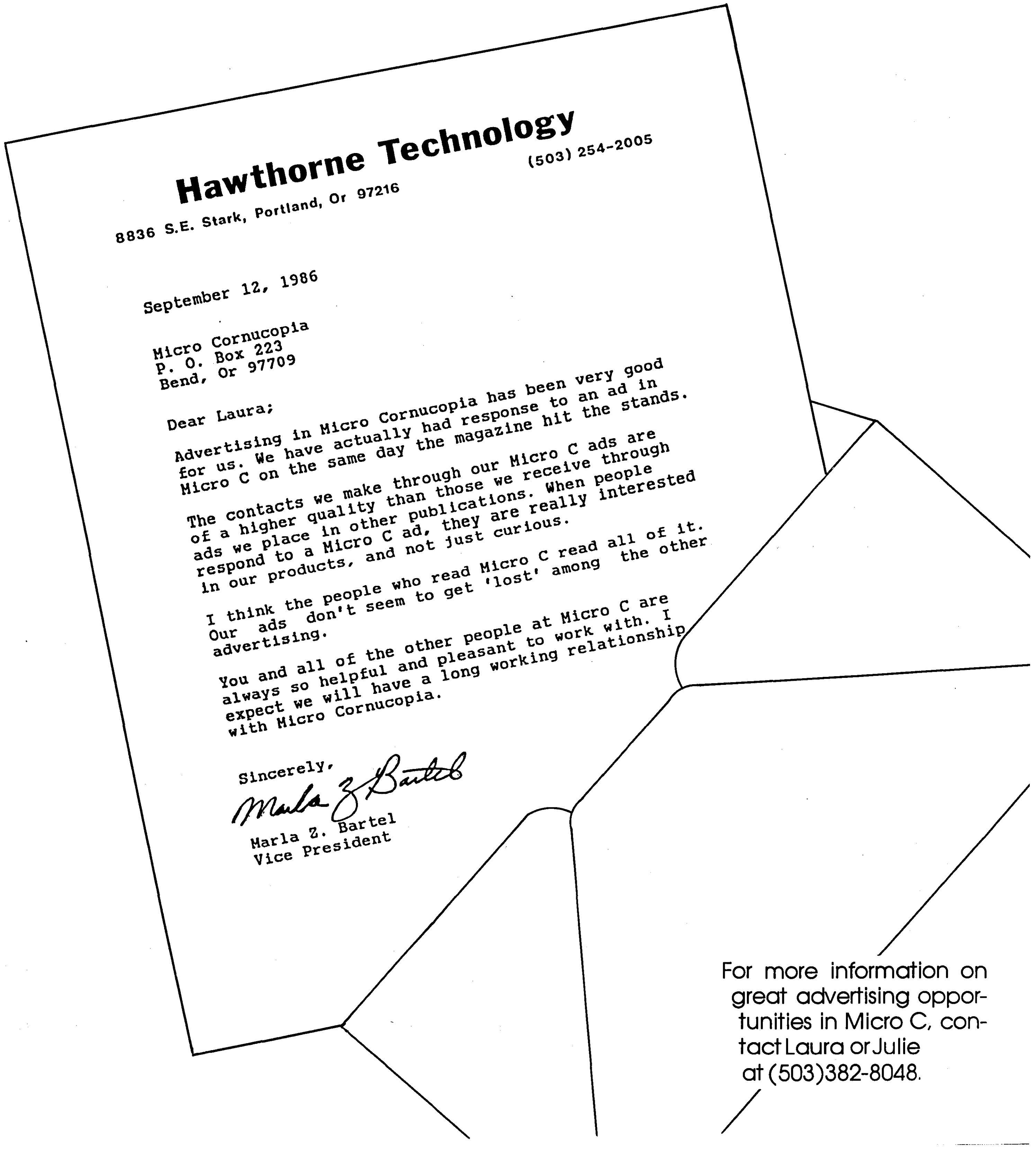
The only problem I have now is that COPY won't even work at 3.5 MHz, but I'm getting a faster program from a neighbor. After an hour's operation at 7 MHz, I'm very pleased with the increase in speed.
Norris Bundy
P.O. Box 29
Alsea, OR 97324

Editor's note:

Sorry for the confusion, Norris. Quite a few boards do have the 74HCU04 chip in U2. Whatever chip is installed in U2 should be left there. Regardless of the type of chip, 14 MHz will always be available on pin 8.

■■■

We can help you sell your products.



Hawthorne Technology

(503) 254-2005

8836 S.E. Stark, Portland, Or 97216

September 12, 1986

Micro Cornucopia
P. O. Box 223
Bend, Or 97709

Dear Laura;

Advertising in Micro Cornucopia has been very good
for us. We have actually had response to an ad in
Micro C on the same day the magazine hit the stands.

The contacts we make through our Micro C ads are
of a higher quality than those we receive through
ads we place in other publications. When people
respond to a Micro C ad, they are really interested
in our products, and not just curious.

I think the people who read Micro C read all of it.
Our ads don't seem to get 'lost' among the other
advertising.

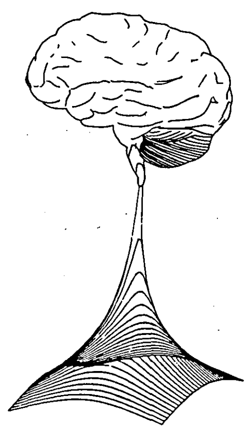
You and all of the other people at Micro C are
always so helpful and pleasant to work with. I
expect we will have a long working relationship
with Micro Cornucopia.

Sincerely,

[illegible]

Marla Z. Bartel
Vice President

For more information on great advertising opportunities in Micro C, contact Laura or Julie at (503)382-8048.



THE MICRO C LOGICAL CONTEST

IF You're tired of wondering what the AI hullabaloo is all about?

OR Convinced you're more than artificially intelligent?

THEN Prove you're logical. Write a program in PROLOG and enter it in the third annual Micro C Programming Contest.

Any PROLOG brand will do, and length is no object.
Programs will be evaluated on the basis of function, ease-of-use, and code & documentation readability.
Winners will be announced in the February '88 issue of Micro C.
Good luck and good programming.

1st Prize

--10 MHZ AT clone board with 1 MByte of RAM from MicroSphere, Box 1221, Bend, OR 97709.

-- micro einstein - An expert system development tool from Gary Entsminger's Acquired Intelligence, 1912 Haussler Dr., Davis, CA 95616.

-- Turbo PROLOG Toolbox from Boralnd International, 4585 Scotts Valley Df Scotts Valley, CA 95066

-- 3 yr. subscription to Micro C.

2nd Prize

-- micro einstein from Acquired I Intelligence.

-- Turbo PROLOG Toolbox from Borland International

-- 2 yr. subscription to Micro C.

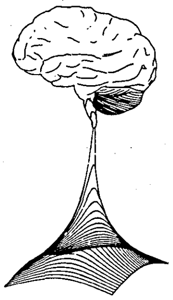
3rd Prize

-- Turbo PROLOG Toolbox from Borland International

-- 1 yr. subscription to Micro C.

--------CONTEST DEADLINE Nov. 1, 1987--------

ENTRY FORM



Program Title ____________

Purpose ____________

Written for which version of PROLOG __

____

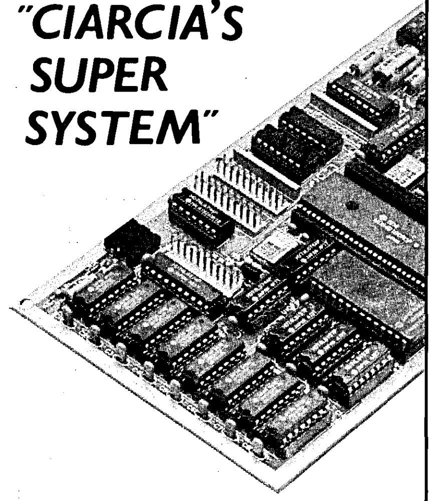
NOTE:
I hereby release this program to the public domain and give Micro Cornucopia the right to print the listing.

Signature ____________

Name ____________
Address ____________
City ______ State ____ Zip ______

Micro Cornucopia
P.O. Box 223
Bend, OR 97709

Byte Magazine called it,
"CIARCIA'S SUPER SYSTEM"



The SB180 Single Board Computer

Featured on the cover of Byte, Sept. 1985, the SB180 lets CP/M users upgrade to a fast, 4" x 7½" single board system.

- **6MHz 64180 CPU**
  (Z80 instruction superset), 256K RAM, 8K Monitor ROM with device test, disk format, read/write.
- **Mini/Micro Floppy Controller**
  (1-4 drives, Single/Double Density, 1-2 sided, 40/77/80 track 3½," 5¼" and 8" drives).
- **Measures 4" x 7½," with mounting holes**
- **One Centronics Printer Port**
- **Two RS232C Serial Ports**
  (75-19,200 baud with console port auto-baud rate select).
- **ZCPR3 (CP/M 2.2/3 compatible)**
- **Multiple disk formats supported**
- **Menu-based system customization**

New Low Prices

SB180-1
SB180 computer board w/256K bytes RAM and ROM monitor $299.00

SB180-1-20
same as above w/ZCPR3, ZRDOS and BIOS source $399.00

COMM180-S
SCSI interface $150.00

Now Available

TURBO MODULA-2.........$69.00
TURBO MODULA-2 with Graphix Toolbox $89.00

TO ORDER CALL TOLL FREE 1-800-635-3355

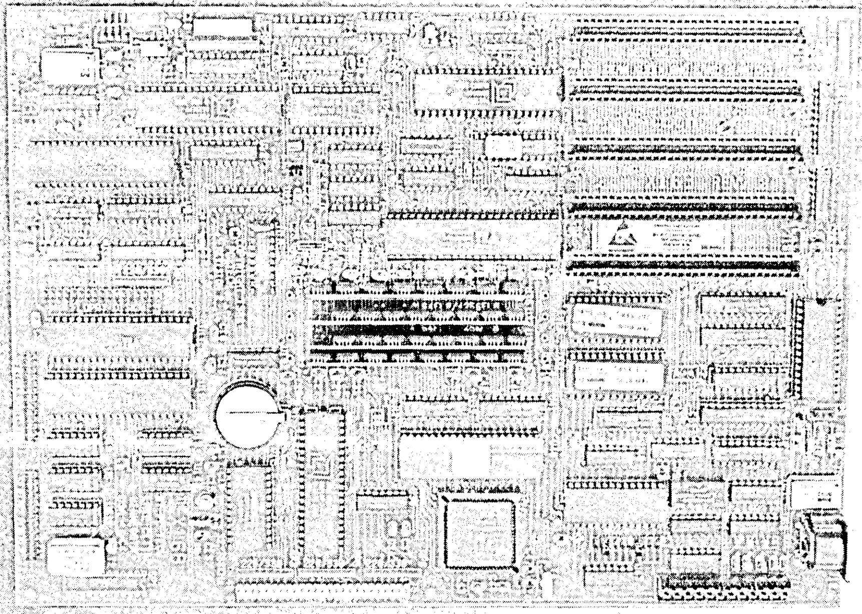
TELEX 643331

For Technical Information or in CT, call:
1-203-871-6170

MICRO MINT

Micromint, Inc.
4 Park Street
Vernon, CT 06066

10 MHz, 0 Wait States, American Made!



The X16B offers the highest performance and integration of any PC/XT compatible. With its 10 MHz, zero wait state operation it walks away from AT compatibles as well. On board is one megabyte of DRAM, a real time clock, floppy disk controller, and optional one or two serial ports, SCSI port and 8087.

The PC Tech SmartBIOS provides PC compatability with ease of use. We wrote it and we support it!

OTHER PRODUCTS

8087-1 Math Coprocessor for X16B

The X16 8087 Math Coprocessor runs at full CPU speed. That's 10MHz of number smashing power! The Math Coprocessor on the AT only runs at 2/3 of the CPU clock.

SCSI Port Option for X16B

Full SCSI port using the 5380. Software built into ROM BIOS for the OMTI 3100 hard disk controller achieves a 1 to 1 sector interleave.

FOUR MEGGER

Running out of spreadsheet room? Need something faster than a hard disk for those long compiles? The Four Megger is the answer. The Four Megger meets the Lotus/Intel/Microsoft expanded memory specifications and works in all PC and XT computers. The Four Megger also works in AT computers as expanded memory only.

2S1P

We found ourselves lacking for a flexible two serial ports and parallel printer port. So we designed the 2S1P (which is short for 2 Serial, 1 Parallel) board. It's small, inexpensive and made here, of course!

Memories

The PC Tech Memories board allows additional ROM or static RAM to be put in a PC, XT, AT, X16B or compatible. We designed it for diagnostics. You can use it for booting without disks, scraping your windshield or whatever!

PRICES!

X16B / 1 Meg / RTC	$600.00
8 MHz version	$540.00
SCSI option	$25.00
Serial ports (2)	$38.75
8087-1/82188	$340.00
8087-2/82188	$260.00
Four Megger	$850.00
2S1P	$99.00
Memories	$95.00
Systems:	
The Box	$799.00
Mono System	$1,099.00
EGA System	$1,849.00

Call for system configuration details. PC Tech also sells hard drives and controllers, video boards and monitors, etc.

PC Tech

P.O. Box 128
904 North 6th Street
Lake City, MN 55041
(612) 345-4555

Watch for more innovative products designed and built by PC Tech!!!

COD, VISA, MASTER CARD, CERTIFIED FUNDS

PC, XT and AT are trademarks of International Business Machines

PC Based EPROM Burner

By Larry Fogg
Micro C Staff

Gang Programmer For Under $200

I remember building Micro C's first EPROM programmer. I built it back in 1981, a prototype for an article, and it's been solid ever since. Maybe because it was connected to a Big Board. We've since purchased two $1,000+ 8-hole commercial programmers. I've mentioned their shortcomings in Micro C from time to time. However, Larry has found one that's solid, easier to use, more versatile and downright cheap. I'll let him tell you about it.

We've had a history of trouble with EPROM burners here at Micro C. They either write garbage or they read garbage. Sockets fail one by one. Burners refuse to talk to host computers. One burner even talked dirty to its host in the form of 120 volts AC on an RS-232 line. It was shocking.

It got so that Tammy (Micro C's EPROM creator extraordinaire) would have to call Dave or me back at least once a week for a little burner doctoring. Some of the doctoring was major and the cursed machine did a lot of travelling back and forth to the factory. It spent more time on vacation than our circulation manager, Tracey!

So I was happy to see Dick Borden walk in the door a while back. He had a PC based EPROM burner for us to play with. It didn't take long to decide that this was the burner for us.

Description

The Sunshine EPROM Writer hails from Taiwan. Its half-length controller card drives either a 1 hole (EW-901B) or a 4 hole (EW-904B) burner that sits neatly on top of the PC. We've only tested it with 2716, 2732 and 2764 series EPROMs, but the specs say it will program anything from 2716 through 27512 parts.

Menu-driven software provides most every function you'd want: reading and writing EPROMs of course, verification of both programmed EPROMs and blanks, storage and recall of object files, and editing of the object file currently in memory. (Sunshine refers to the memory used by the object file as the "buffer".)

Both the documentation and the menu use a good approximation of English. However, some familiarity with EPROM programmer functions eases the first few programming sessions. For example, I had trouble verifying any EPROMs which weren't in the first socket. It turns out that the menu option, "Set textool size," must be used to change the default gang size of 1. Silly me, I should have known that.

In a normal copy session, verification shows only the address of the first error in each EPROM copied. If you want a more complete display of errors, use the "Verify & display errors" option. This choice displays the address, EPROM value, and buffer value (in parentheses) for each error.

It also lets you read object files into the burner's buffer beginning at any address. For example, the code from a 2716 can easily be burned into the top half of a 2732. Just set the starting address at 0800h when reading the 2716.

Performance

No one wants to sit around and watch EPROM burners run. Speed of execution turns out to be one of the Sunshine programmer's strong points. The highly subjective Tammy Speed Rating shows that it runs twice as fast as our old Southern Computer Corporation Model 512A. In reality, it performs even better.

I used 2764A EPROMS to test the two programmers. A look at the results (see Figure 1) shows that the Sunshine is almost 4 times faster. Notice that gangs of 1 and 4 program at close to the same speed. This makes sense since the programming is done simultaneously. Only the verification process (very quick compared to programming) goes one by one.

	SCC	SUNSHINE
Read (2764A)	0:06	0:01
Write/Verify (1)	1:35	0:23
Write/Verify (4)	1:37	0:25

Drawbacks

None. Seriously. Sure, it would be nice to have 37 sockets and a price tag of $19.99. And maybe they could throw in one of those oriental cooking knife sets. *(Editor's note: I think they did, but someone beat Larry to it.)* Aside from the somewhat inscrutable documentation and an occasional lapse into questionable spelling, I can't complain.

Well, maybe I should complain about one thing. When specifying starting and stopping addresses for a dump of the buffer contents, "D0000.00FF" cannot be shortened to "D0.FF". All addresses must be a full 4 characters. A minor drawback, but when you club the keyboard as poorly as I do, every little bit of shorthand helps.

Availability

And now, the bottom line. Stand-alone multi-hole EPROM programmers go for $800 and up. The best price I've found for the Sunshine 1-holer is $119 from McTEK in Berkeley. We bought their 4-holer for $169. The thing was DOA but the hassle-free replacement has been burning 'em for four months now.

So, I'll give the Sunshine an "A." Its ease of use and speedy programming make it a good choice for anyone with a burning desire for cheap EPROM programming.

Sources

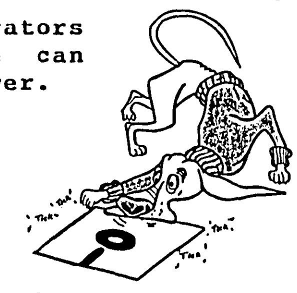
McTEK
2316 Fourth St.
Berkeley, CA 94710
(415) 549-3472

JDR Microdevices
110 Knowles Dr.
Los Gatos, CA 95030
(800) 538-5000

■■■

Ever Wondered What Makes CP/M® Tick?

Source Code Generators by C. C. Software can give you the answer.



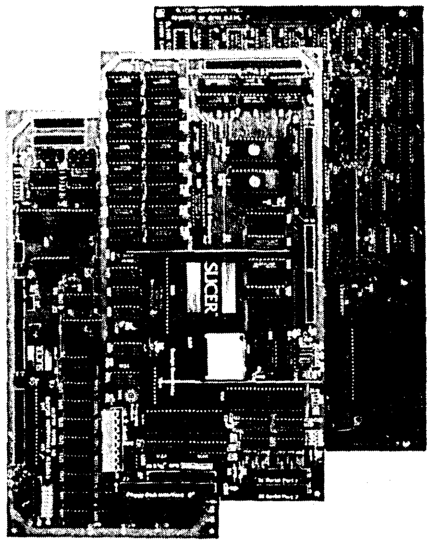
"The darndest thing I ever did see..." "... if you're at all interested in what's going on in your system, it's worth it." Jerry Pournelle, BYTE, Sept '83

The S.C.G. programs produce fully commented and labeled source code for your CP/M system (the CCP and BDOS. areas). To modify the system to your liking, just edit and assemble with ASM. CP/M 2.2 $45, CP/M+ $75, + $1.50 postage (in Calif add 6.5%).

C. C. Software, 1907 Alvarado Ave.
Walnut Creek, CA 94596 (415)939-8153

CP/M is a registered trademark of Digital Research, Inc.

SLICER...THE TRUE LOW COST MULTI-USER MICRO



THE SLICER
Real 16 Bit Power on a Single Board Featuring the Intel 80186 (C) Step CPU

- Complete 8 MHz 16-bit microprocessor on a 6" x 12" board
- 256K RAM, plus up to 64K EPROM
- SASI port for hard disk controller
- Two full function RS232C serial ports with individually programmed transmission rates — 50 to 38.4K baud
- Software compatibility with the 8086 and 8088
- 8K of EPROM contains drivers for peripherals, commands for hardware checkout and software testing
- Software supports most types and sizes of disk drives
- Source for monitor included on disk
- Bios supports Xebec 1410 and Western Digital WD 1002 SHD controller for hard disks

Fully assembled and tested only $445

THE SLICER 1-MByte EXPANSION BOARD
For expanded memory, additional ports and real time clock

- 1-MByte additional dynamic RAM
- 2 RS232C asychronous ports with baud rates to 38.4K for serial communication
- 2 additional serial ports for asynch (RS 232) or synch (Zilog 8530 SCC) communication
- Real Time Clock with battery backup
- Centronics type parallel printer port

Fully assembled and tested only $450

SLICER/1MByte EXPANSION COMBO
The Slicer (without RAM or RAM controllers) with Full 1-MByte Memory Expansion

CCP/M (Digital Research) $950

THE SLICER PC EXPANSION BOARD
Gives your Slicer high performance video capability

- IBM campatible monochrome video
- Video memory provides 8 pages of test or special graphics capability
- 2 IBM type card slots for color video, I/O expansion, etc.
- IBM type keyboard port

Fully assembled and tested only $495
All boards available in kit forms
Runs MS DOS generic software; PC DOS program operation not guaranteed

Also available: The uSlicer 188 $400; 8087 Math Co-Processor Bd. (call); 10 MB Hard Disk $465; W.D. 1002-SHD H.C.D. Bd. $200; Enclosures, Power Supply, and Support Hardware.
CP/M 86 $85, CCP/M $250 (Digital Research, Inc.); MS DOS $175 (Microsoft Corp.)

MasterCard, Visa, Check, Money Order, or C.O.D. Allow four weeks for delivery. Prices subject to change without notice.

NOTE NEW ADDRESS & PHONE NO.

SLICER
Slicer Computers Inc.
3450 Snelling Ave. So.
Minneapolis, MN 55406
612/724-2710
Telex 501357
SLICER UD

By Gary Entsminger
1912 Haussler Dr.
Davis, CA 95616

Happiness, The Too Much Information Blues, & micro einstein

Gary tries to confuse Taskview by loading it up with concurrent processes. The score? Taskview 1, Gary 0. He also discusses WindowDOS and the PROLOG competition.

Last issue I wrote about Taskview, a useful (and unusual) menu-driven command shell. Under Taskview command, you set up your own menu of programs which you run by entering the first letter or by moving the cursor. You can suspend a running program in execution, and switch back to it later or abandon it entirely.

I've been using Taskview again this issue and have discovered that it suspends programs in the background only some of the time - when a program is too large to retain in memory.

If a program is swapped to disk, it's suspended (and nothing new happens while you're away). However, if the program is small enough (Turbo Pascal qualifies; Turbo PROLOG doesn't), and doesn't need to be swapped to disk, it will continue to run in the background, more-or-less concurrently. It still must compete for a single CPU's time, but does continue to compile, write to disk, etc...

For example, I started Turbo Pascal, loaded a file, switched (via Taskview) to another copy of Turbo Pascal, started it, loaded a file, and then started yet a third copy of Turbo Pascal, and loaded a file.

To keep the test simple (I also hoped to confuse Taskview), I loaded the same Pascal source file (2000 lines including INCLUDE files) each time. To keep tabs on each program while it compiled, I switched between tasks by pressing -

<CTRL> <SHIFT> <N>

where N = the number of the task. When programs are small enough, the switch is virtually instantaneous (since everything is running in memory).

So, first I compiled the source in copy 1. Then, I compiled the source in copy 2 (just for a check, and no surprise - they compiled within a few tenths of a second of each other).

Then, I began a compile in one, switched to another, began a compile, switched to another, started an edit, and then alternately (and merrily) switched back and forth, and back and forth to watch the action.

Happiness! Everything worked, although it did take a few more seconds (but just a few more) to compile the files concurrently with Taskview than it took to compile them separately.

Next, I tried to confuse Taskview by writing concurrently to the same file. If you're following closely, you've probably already guessed the result - Taskview had no problem (since it really processes sequentially) writing and closing the files, one by one.

So, no luck at attempted confusion, and Taskview continues to intrigue me. If you want to come as close as you can to concurrency on the PC without adding a second processor, call Sunny Hill for more details.

$69.95 from -
Sunny Hill Software
P.O. Box 33711
Seattle, WA 98133-3711
(206) 367-0650

WindowDOS 2.0

Directories have been essential programs since the beginning. And since the beginning, programmers have been trying to improve DIR, the MS-DOS directory program (see the Culture Corner, Micro C, #35 for a somewhat redundant, and yet less-than-thorough, history).

WindowDOS (first released in 1984) was one of the first DOS shells to substantially improve on DIR. And (as far as I know) it was the first to do it in TSR (Terminate-But-Stay-Resident) fashion.

You load WindowDOS, and whenever you need it (even when you're using another program), call it up by pressing -

<SH> <INS>

It automatically sorts the current directory and writes it to the screen, where you can manipulate individual or groups of tagged files by moving a highlight with the arrow keys.

But that's really just the tip of the iceberg.

WindowDOS 2.0 lets you format disks, copy and erase files, make directories, list files (in ASCII or text), sort files (by filename, extension, creation date, and size in ascending and descending order), hide and unhide, protect and unprotect, and password lock files. You can set the time and date and set up macros.

Although I use several directory programs (including XTREE), I prefer WindowDOS when I'm working with a hard disk. Why? Because it doesn't read the entire disk structure until I tell it to. Unless I specify otherwise, it reads the current directory, period.

When a hard disk fills to 1000+ files and 19+ megabytes, a reading of the entire disk just takes too long (even on X16s and ATs). I call this the "Too much information blues," and will do almost anything to avoid it.

If "the blues" is your problem, try WindowDOS. It could help. Sells for $49.95, and includes a nifty screen capture utility

WindowDOS Associates
Box 300488
Arlington, TX 76010
(817) 467-4103

The PROLOG Programming Contest

We're into month two in the third annual Micro C programming contest, and if you haven't heard, we're doing PROLOG. So put on your thinking caps and logical scarves, and start programming in this blue plate special dance of a language.

I've been working hard to perfect my own skills in the AI arena, and am very pleased to be donating two copies of my expert system shell, "micro einstein," (written in Turbo PROLOG and C) to the first and second place winners. For a complete list of prizes (including a 10 MHz AT board from MicroSphere), see the PROLOG contest information elsewhere in this issue.

If you're having trouble deciding what to write in PROLOG, you might look over Herbert Schildt's "Advanced Turbo PROLOG." Although not really an "advanced" text, it does offer several good programming leads, including: vision and pattern recognition, natural language processing, expert systems, and robotics.

I would love to see a good two (or even three) dimensional pattern recognition program.

For a preliminary study of PROLOG, read Clocksin and Mellish (*Programming in PROLOG*), Schildt, or the Turbo PROLOG reference manual. If you want to hone your technique, read (and study) *PROLOG Programming For Artificial Intelligence* by Ivan Bratko. It's the best "advanced" book and the best "general" textbook of PROLOG programming I've seen.

Bratko leads the AI groups at the Josef Stefan Institute and the E. Kardelj University in Ljubljana, Yugoslavia, and has applied PROLOG in medical expert systems and computer chess research.

His book has a practical flavor which I found (and continue to find) appealing.

It's $25.95 from Addison-Wesley.

And that, folks, is Tidbits. See you at SOG VI and in the funny papers.

■■■

CALL FOR FREE CATALOG

ALL SALES SUBJECT TO THE TERMS OF OUR 90 DAY LIMITED WARRANTY. FREE COPY UPON REQUEST.

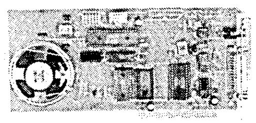
TEXT TO SPEECH BOARD!

PC/XT COMPATIBLE. MAKE YOUR COMPUTER TALK!

A VERY POWERFUL AND AMAZING SPEECH CARD. USES THE NEW GENERAL INSTRUMENTS SPO256-AL2 SPEECH CHIP AND THE CTS256A-AL2 TEXT TO SPEECH CONVERTER.

THIS BOARD USES ONE SLOT ON THE MOTHERBOARD AND REQUIRES A COM SERIAL PORT. BOARD MAY ALSO BE USED IN A STAND ALONE ENVIRONMENT WITH ALMOST ANY COMPUTER THAT HAS A RS232 SERIAL PORT. FEATURES ON BOARD AUDIO AMP OR MAY BE USED WITH EXTERNAL AMPS.

DEMONSTRATION SOFTWARE AND A LIBRARY BUILDING PROGRAM ARE INCLUDED ON A 5¼ INCH PC/XT DISKETTE. FULL DOCUMENTATION AND SCHEMATICS ARE ALSO INCLUDED.



NEW!

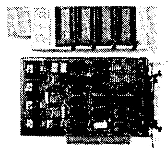
$89.95
ASSEMBLED & TESTED

CANON 80 COLUMN PRINTER - $39.95

ORIGINALLY MANUFACTURED FOR THE PC JR. BUT WITH OPTIONAL CONNECTOR WILL WORK WITH PC, XT, OR AT. REQUIRES SERIAL I/O. THIS THERMAL PRINTER IS QUIET AND USES EASY TO GET 8½ IN. ROLLS OF PAPER. 50 C.P.S., UPPER AND LOWER CASE, PLUS GRAPHICS. ORIGINAL LIST PRICE $199.00. ADD $3.00 FOR PC/XT CONNECTOR. ADD $5.00 UPS.

PC/XT EPROM PROGRAMMER

$199



NEW!

★ LATEST DESIGN ★ PROGRAMS UP TO 4 DEVICES AT ONE TIME ★ FEATURES EASY TO USE MENU DRIVEN SOFTWARE THAT RUNS UNDER PC OR MS-DOS. ★ USES AN INTELLIGENT PROGRAMMING ALGORITHM FOR SUPER FAST (8X) EPROM BURNING. ★ THIS PLUG-IN BOARD ATTACHES TO AN EXTERNAL MINI CHASSIS CONTAINING 4 TEXTOOL Z.I.F. SOCKETS. ★ NO PERSONALITY MODULES REQUIRED ★ AUTOMATIC VPP SELECTION: 12.5V, 21V, OR 25V. ★ EPROM DATA CAN ALSO BE LOADED FROM OR SAVED TO A DISKETTE. ★ PROGRAMMING SOFTWARE SUPPORTS: 2716, 2732, 2732A, 2764, 2764A, 27128, 27128A, 27256, 27256A, 27512, AND 27512A. ★ ASSEMBLED AND TESTED, BURNED. IN WITH MANUAL. $199 WITH SOFTWARE.

Digital Research Computers
(OF TEXAS)
P. O. BOX 381450B • DUNCANVILLE, TX 75138 • (214) 225-2309

TERMS: Add $3.00 postage. We pay balance. Orders under $15 add 75¢ handling. No C.O.D. We accept Visa and MasterCard. Texas Res. add 6-1/4% Tax. Foreign orders (except Canada) add 20% P & H. Orders over $50 add 85¢ for insurance.

Quelo(R) 68000 Software Development Tools

First release 1983 - MOTOROLA compatible - produces ROMable code, S-records, extended TEK hex, UNIX COFF. Portable SOURCE CODE. Native and cross versions on: ATARI ST, AMIGA, Masscomp, Sun, Apollo, Charles River, VAX VMS and UNIX.

68020 Cross Assembler Package
Supports 68000, 68010, 68020, 68881 and 68851
For CP/M-68K and MS/PC-DOS - $750

68000/68010 Cross Assembler Package
For CP/M-80, -86, -68K and MS/PC-DOS - $595

68000 "C" Cross Compiler
For MS/PC-DOS by Lattice, Inc. - $500

NEW ITEMS

68020 Disassembler
Supports 68000, 68010, 68020, 68881, 68851
For CP/M-68K and MS/PC-DOS - $495/295,
Amiga and Atari ST - $119/79, CRDS UNOS - $995/595

68000/68010 Software Simulator
For MS/PC-DOS by Big Bang Software, Inc. - $285, VAX - $1900

NEW ITEMS

Call **Patrick Adams** *today:*

Site, Corporate, OEM licenses
COD, Visa, MasterCard

Quelo, Inc.
2464 33rd. West, Suite #173
Seattle, WA USA 98199
Phone 206/285-2528
Telex 910-333-8171

TM: Quelo, Quelo, Inc.; MS, Microsoft Corporation; CP/M, Digital Research

Writing Portable 8086 Assembly Language Functions

By Ken Berry
P.O.Box 966
Jackson, CA 95642-0966

Porting Assembly From C To Shining C

OK, folks. You wanted assembly language? Here's assembly language. Want to port your assembly language routines from one C to another? Or, from one high level language to another?

Well, continuing Micro C's tradition of addressing the needs of many levels of programmers and developers, we offer Ken's in depth study. Developers working in any high level language will no doubt learn a lot from Ken's experiences.

Soon after Microsoft released version 4 of its C compiler, they published benchmarks showing its sterling performance. I had been using the Lattice compiler, which took longer to compile and produced less efficient code as well, so I bought Microsoft C, version 4.0.

I knew that Microsoft had previously licensed the Lattice compiler and marketed it under their own name, so I naively assumed that their new, fast compiler would be compatible with my existing source code.

I was wrong. The C code ported easily, though it needed to be recompiled (it wasn't possible to use the Lattice object module libraries). Fortunately, I still had the source code to every program I needed.

Assembly language portions of the programs are more difficult to port, however. The two C compilers use different segment names and different conventions for stack usage, as well as different protocols for calling assembly code.

So I had to make many changes to the source code, without changing the way the functions operated. A situation of this nature is time-consuming and promotes unreliability.

Although I was tempted to blame Microsoft, I realized I had set my own trap by assuming the Lattice conventions were a standard. It would be much better if my assembly programs could be used not only by different compilers for the same language, but also by different languages.

Accompanying this article is a system of macros that allows assembly programs to interface to any reasonable calling protocol, segment structure, and memory model. Only a few of the most interesting macros are listed in this article. The complete set can be downloaded from the Micro C BBS. They exercise the macro facility of the Microsoft MASM version 4.0 assembler; I have no experience using them with any other assembler.

Usage

Figure 1 contains a test program. If you run it and have the assembler make a listing, you can see how the code generated changes for different calling protocols and memory models.

After you've included the macro definitions, the _cmplr macro will select a specific high level language compiler and memory model. Table 1 identifies the special symbols to use.

_cmplr defines internal symbols based on the specific selection symbols we define. I define the appropriate symbols on the command line invoking MASM if I want to override the default selections. For example, the following command line will assemble with the Lattice compiler conventions:

```
masm /dLTC syhlt0;
```

_cmplr invokes another macro specific for each compiler supported.

_lc defines several symbols according to the conventions of the Lattice compiler. _msc does the same for Microsoft. Those are the only two I've had to figure out for my work. But it's easy to add other compilers if you realize there are two independent issues: segment structure and calling protocol.

Segment Structure

The compiler designer chooses the segment structure. Predictably, there's wide variation, but all structures are logically similar.

The _crtseg macro creates a segment and _defgrp defines a group of segments. If you use a compiler I haven't tested, I hope reading the _lc and _msc definitions will help you figure out the appropriate calls to _crtseg and _defgrp for yours. Once you have a correct macro corresponding to _lc and _msc, you can forget almost all about segments.

The _begs and _ends macros are used to bracket segments in assembly programs. There are only two segments to worry about: one called "data" and the other "code."

A "_begs data" instruction must appear immediately before the first static data definition, and "_ends" immediately after the last. Similarly, a "_begs code" instruction immediately precedes the first executable instruction. Another "_ends" terminates the code segment. In effect, the _cmplr macro maps the symbols "code" and "data" to the name appropriate for the current compiler and memory model.

Data Types & Memory Models

If you apply these macros to existing assembly code, you'll have to systematically change many statements. But it's no more difficult to write new programs using the macros, because macros simply replace many assembler instructions.

For example, seven data types are supported: byte, word, double word, quad word, ten byte, code pointer, and data pointer. Where an assembly program might ordinarily use the "dw" instruction to define a word of data, using the "_dw" macro makes the word easy to use in subroutine calls. It's not a big deal to provide an alternative to the "dw" instruction.

MASM doesn't provide instructions for all data types, so it's necessary to define macros in order to consistently determine the length of the item.

Some data elements depend on the memory model being used, and the macros allow you to reassemble programs for any memory model without making changes to the source.

The differences between memory models are reflected in the size of pointers. Small pointers use single words. Large pointers are double words.

All seven data types can be used in five ways: external and internal labels, static data definitions, parameter definitions, and temporary data definitions. The sample program in Figure 1 shows the complete set of external and data macros.

Procedures & Functions

Procedures can be defined as near, far, or according to memory model. The default for small code memory models is near, while that for large code models is far. You can override the defaults for special applications.

The 8086 family architecture is well suited to recursion because it uses a stack and can easily manipulate data on the stack. The stack is used for passing parameters during function calls as well as for local variables and hardware operations. A program may call itself (recurse) because each invocation uses a unique stack area.

Local variables are temporary data. Such data is discarded when the subroutine returns to its caller. Static data is different because it exists whether or not the code with which it's defined is being executed. Parameters and temporary data are defined at the time pro-

(continued next page)

Figure 1 - Test Program

```
comment ~
Sample Program (file syhlt0.asm)

(C) Copyright 1987 Ken Berry- All rights reserved.
Copies may be made for non-commercial, private use only.
~

        title syhlt0.asm porting macros test program
        name syhlt0

        include syhl.mac            ; include porting definitions

        _cmplr                      ; specify compiler and memory model


; external declarations
        _xb  xbyte                  ; external byte label
        _xw  xword                  ; external word label
        _xd  xdword                 ; external double word label
        _xq  xqword                 ; external quad word label
        _xt  xtbyte                 ; external ten byte label
        _xnp xnearp                 ; external near procedure label
        _xfp xfarp                  ; external far procedure label
        _xp  xproc                  ; external procedure label (depends on code size)
        _xcp xcodept                ; external code pointer label
        _xdp xdatapt                ; external data pointer label


; static data

        _begs data                  ; begin static data segment

        _db dbyte,0,0,public        ; byte
        _db dstring,1,30,public     ; data string (30 characters)
        _dw dword,2                 ; word
        _dd ddword,3                ; double word
        _dd dqword,4                 quad word
        _dt dtbyte,5                ; ten byte
        _dcp dcodept,6              ; code pointer
        _lbw ,ddatawpt              ; public symbol ddatawpt = ddatapt
        _ddp ddatapt,7              ; data pointer
        _ends                       ; end static data segment


; code
        _begs code                  ; begin code segment
        _dclp subr0,far             ; subr0 function declaration
        _pb b0                      ; parameter b0
        _pw w1                      ; parameter w1
        _lw w2                      ; local w2
        _begp                       ; subr0 entry code
        _call subr1,w2,w2           ; call subr1
        _endp                       ; subr0 exit code
        _dclp subr1                 ; subr1 function declaration
        _pw w0                      ; parameter w0
        _pw w1                      ; parameter w1
        _begp                       ; subr1 entry code
        mov ax,w0                   ; compute w0 + w1
        add ax,w1
        _endp                       ; subr1 exit code
        _ends                       ; end code segment
        end                         ; end of program


comment ~

end of Sample Program

~
```

(continued from page 71)

cedures are declared.

Procedures correspond to individual functions or subroutines. They're declared by the _dclp macro. The entry code is generated by a subsequent _begp macro. Exit code is generated by an _endp macro. This assumes that control enters the assembly function at the top and only exits through the bottom. Embedded returns use the _xitp macro.

Function arguments and temporary data are defined between the _dclp and _begp macros. After you define a symbol as an argument or local variable, you can use the name with no special qualification (such as "ptr" operators). The assembler will automatically access an object of the proper size on the stack.

Calls to other functions (which may also be in assembly or the high level language) are supported by the _call macro. You list the arguments to the function in the _call instruction, and the assembler will automatically place them on the stack. The called function must list its arguments in the same order as they appear in the _call.

Two additional macros are used in association with _call. _save takes a list of registers that are to be preserved across the call. _altr defines the registers that will be altered by a function (which may be either internal or external). _call will protect all (and only) those registers specified in both macros.

Protocol Control

The calling protocol is usually unique to a vendor, as well as to a language. It's generated by the _begp, _endp, and _call macros, depending on symbols defined in the compiler selection macro (_lc or _msc). Table 2 summarizes the differences between the Lattice and Microsoft compilers in terms of the symbols described in the following paragraphs.

If "?ldul" is 1, an underline character is prefixed to every external and public symbol. Microsoft says this convention is for compatibility with Xenix, but that doesn't justify Xenix having it. I find systematic names very important for keeping complex software systems organized. It's no help to have irrelevant conventions imposed by a tool. ?ldul allows me to ignore Microsoft's convention and fully control what names are used in the source code.

If "?nglclo" is 1, local variables will be accessed as a negative offset from the BP register (BP is used to access all data on the stack). If it's 0, the local variables are accessed as positive offsets. The function arguments are always accessed with positive offsets.

"?fnp" determines the order in which the function arguments are placed on the stack. Is the argument written first in a call statement stored with the least or greatest offset from BP?

In C, the first argument takes the smallest offset, but other languages differ. This is a significant point in language design, only the C convention allows functions to have a variable number of arguments. Users of the "printf" function appreciate variable argument lists.

"?scs" determines how stack housekeeping is performed. When a function terminates, should it erase its parameters and local variables from the stack? Or should it just erase its local variables and let its caller erase the arguments? In either case, the final code also depends on whether the locals have positive or negative offsets.

If "?scs" is 1, a "ret n" instruction terminates the subroutine. This results in slightly smaller code, but is incompatible with a variable number of function arguments.

In C, there's no way to determine the number of arguments unless the caller specifically passes that information. Therefore, there's no way for the called function to know how much to

(continued on page 74)

Table 1 - Compiler & Model Selection Symbols

LTC	Lattice C compiler (version 3.0)
MSC	Microsoft C compiler (version 4.0)
SCSD	small code, small data
LCSD	large code, small data
SCLD	small code, large data
LCLD	large code, large data

Table 2 - Lattice & Microsoft Conventions

Protocol

Symbol	Meaning	Lattice	Microsoft
?ldul	leading underline	0	1
?nglclo	negative local offset	0	1
?scs	subroutine clears stack	0	0
?fnp	fixed number of parameters	0	0

Segments

Segment		Lattice	Microsoft
Code	(small code, small data)	prog	_TEXT
	(large code, small data)	_code	_TEXT
	(small code, large data)	code	_TEXT
	(large code, large data)	_prog	_TEXT
Static Data		data	_DATA
Constant (read only static data)			CONST
Uninitialized Static Data		udata	_BSS
Stack (as named in Tele)		xstck	_XSTCK

Figure 2 - Example Macros

```
(C) Copyright 1987 Ken Berry- All rights reserved.
Copies may be made for non-commercial, private use only.

This listing is incomplete. The Micro Cornucopia BBS has
the complete file.
Segment Control
~

; create segment
_crtseg   MACRO n,ln,t,p,c,g
  ifnb <g>
          ?addseg g,n    ;; add segment to group
  endif
  ifnb <c>
n         segment t p '&c' ;; define segment in class
  else
n         segment t p    ;; define independent segment
  endif
n         ends           ;; close segment
          ?cs1 n,ln
          ENDM

?cs1      MACRO n,ln
?b_&ln    &MACRO
          ?cs2 n,ln
n         segment
          &ENDM
          ENDM

?cs2      MACRO n,ln
_ends     &MACRO
  if2
%out end segment &ln
  endif
n         ends
          &ENDM
          ENDM

_begs     MACRO ln
  if2
%out begin segment &ln
  endif
          ?b_&ln
          ENDM

; define group

_defgrp   MACRO g
          ?addseg g      ;; define group
          ENDM

; add segment to group

?addseg   MACRO g,n      ;; add segment to group
.xcref
.xcref ?p_&g
.cref
  ifndef ?p_&g
?p_&g     = 0            ;; define pass control variable
  endif
  if ?p_&g ne ?pass      ;; test for first time this pass
?add_&g   &MACRO s
          ?in_&g <n>,s
          &ENDM
?in_&g    &MACRO sl,s
    ifb <s>
g         group sl
    else
?add_&g   &MACRO ns
          ?in_&g <sl,s>,ns
          &ENDM
    endif
          &ENDM
?p_&g     = ?pass        ;; update pass flag
  else
          ?add_&g n
  endif
          ENDM
```

```
comment ~
Procedure Calls
~

; procedure call

_call     MACRO n,a      ;; call high level function
.xcref
.xcref ?actr
.cref

  ifnb <a>               ;; test for parameters specified
          _arg <&a>      ;; process argument list
  endif
?argl     = 0            ;; initialize argument length
  if ?fnp                ;; fixed number of arguments
?actr     = 0            ;; initialize argument count
  else                   ;; variable number of arguments (C protocol)
?actr     = ?argc        ;; initialize argument count
  endif

  if ?rsv                ;; test for registers to be saved
    ifdef &a_&n          ;; test for altered registers defined
?rsav     = ?rsv and ?a_&n ;; define registers to be saved
    else
?rsav     = ?rsv         ;; define registers to be saved
    endif
    if ?rsav             ;; test for registers to be saved
          ?mpush ?rsav   ;; protect registers
    endif
  endif

  if ?fnp                ;; fixed number of arguments
    rept ?argc           ;; scan argument list from front
?actr     = ?actr+1      ;; decrement argument counter
          ?call %?actr   ;; put argument on stack
    endm

  else                   ;; variable number of arguments (C protocol)
    rept ?argc           ;; scan argument list from back
          ?call %?actr   ;; put argument on stack
?actr     = ?actr-1      ;; decrement argument counter
    endm
  endif

  if ?ldul               ;; test for leading underline convention
          call _&n       ;; call procedure
  else                   ;; no leading underline
          call n         ;; call procedure
  endif

  if ?argl               ;; test for arguments on stack
    if ?scs              ;; test for subroutine clears stack
      ifdef ?rsav        ;; test for registers to be saved
          ?mpop ?rsav    ;; restore registers
      endif
    else                 ;; caller clears stack
      if ?nglclo         ;; test for negative local offset
            add sp,?argl ;; unload stack
        ifdef ?rsav      ;; test for registers to be saved
            ?mpop ?rsav  ;; restore registers
        endif

      else               ;; positive local offset
        ifdef ?rsav      ;; test for registers to be saved
            add sp,?argl ;; uload stack
            ?mpop ?rsav  ;; restore registers
        else             ;; no registers to be saved
            mov sp,bp    ;; unload stack
        endif
      endif
    endif
  endif

?argc     = 0            ;; reset argument count
?argl     = 0            ;; reset argument length
          ENDM
```

(Figure 2 continued next page)

Figure 2 - Continued

```
; put argument i on stack

?call       MACRO i         ;; put argument on stack
            ?a_&i           ;; expand storage macro
            purge ?a_&i     ;; purge storage macro
            ENDM



; determine arguments

_arg        MACRO a         ;; process argument list
  irp x,<&a>                ;; scan argument list
?argc       = ?argc+1       ;; increment argument count
            ?arg1 <&x>,%?argc ;; define parameter store macro
  endm
            ENDM



; define argument storing macro

?arg1       MACRO n,i       ;; process argument
.xcref
.xcref ?a_&i
?a_&i       &MACRO          ;; store argument
            ?arg2 n         ;; store argument
            &ENDM
.cref
            ENDM



; store argument on stack

?arg2       MACRO n         ;; store argument
?argl       = ?argl+2       ;; increment argument length
  ifdef ?t_&n               ;; test for size defined
    ife ?t_&n               ;; test for type = 0
            push word ptr (n) ;; store word argument on stack
            EXITM           ;; exit macro
    else
    ife ?t_&n-1             ;; test for byte (type = 1)
            push word ptr (n) ;; store byte argument on stack
            EXITM           ;; exit macro
    else
    ife ?t_&n-2             ;; test for word (type = 2)
            push word ptr (n) ;; store word argument on stack
            EXITM           ;; exit macro
    else
    ife ?t_&n-4             ;; test for double word (type = 4)
            ?arg2a n        ;; store double word argument on stack
            EXITM           ;; exit macro
    else
    ife ?t_&n-8             ;; test for quad word (type = 8)
            ?arg2b n        ;; store quad word argument on stack
            EXITM           ;; exit macro
    else
    ife ?t_&n-10            ;; test for ten byte (type = 10)
            push word ptr (n)+8 ;; store least significant bytes
            ?arg2b n        ;; store quad word argument on stack
            EXITM           ;; exit macro
    endif
    endif
    endif
    endif
    endif
    endif


  else                      ;; size not defined
            push n          ;; store argument on stack
  endif
            ENDM

?arg2a      MACRO n
            push word ptr (n)+2 ;; *  store double word argument on stack
            push word ptr (n) ;; *
            ENDM

?arg2b      MACRO n
            push word ptr (n)+6 ;; *  store quad word argument on stack
            push word ptr (n)+4 ;; *
            arg2a n         ;; *
            ENDM


comment ~
end of Listing
~
```

(continued from page 72)

remove from the stack. Languages that don't support variable argument lists may have "?scs" set, but successful C compilers won't.

"?sxr" is 1 for the Microsoft compiler. It causes the index registers SI and DI to be saved.

The Microsoft compiler automatically saves the general byte/word registers AX, BX, CX, and DX; it does not protect SI and DI. Perhaps the compiler designer didn't realize that, in addition to their roles as index registers, SI and DI are general purpose 16-bit registers. If your assembly function doesn't use SI or DI, you can add a parameter to the _begp and _endp macro calls to stop protecting them.

The Lattice compiler automatically saves SI and DI if it's using them. So they're not saved by the _begp and _endp macros.

Macro Coding Techniques

If you don't use the Microsoft MASM version 4 assembler, you may have trouble with some macros. MASM offers most of the features I enjoyed on mainframes, and I recommend its use with these macros. If you use another one, your assembler must support conditionals and have a macro facility.

Furthermore, its symbol definition statement must act like a preprocessor definition. That is, a symbol is defined as a string; the string replaces the symbol wherever it appears in subsequent code. Some assemblers define a symbol as having the value evaluated from a string when it is defined; they may be difficult to use.

Even if you do use MASM, you may find some code difficult to modify - the macro facility is a language in itself, and it's beyond the scope of this article to explain macro programming in detail. The following comments explain some techniques that I have rarely encountered outside systems programming.

Advanced System Techniques

One technique involves defining one macro during the expansion of another. The segment creation macro, _crtseg, is the first example of this (it may be found about 100 lines into Figure 2).

Two names are specified for _crtseg. The first is the name of the segment as used by the linker (and varies between compilers). The second is the generic name that does not change.

For example, if you refer to the segments "code" and "data" in your source code, the _crtseg macro associates that name with the link segment name appropriate for the current compiler and memory model.

_crtseg associates the names by defining a macro with the name "?b_ln" where "ln" is the logical name ("code" or "data"). The expansion of ?b_ln produces a segment statement with the appropriate link name. The _begs macro uses its parameter to expand the appropriate ?b_ln macro. For example, the statement "_begp code" will cause the macro "?b_code" to be expanded.

Also notice that when ?b_ln is expanded, the _ends macro is defined with the appropriate link name to close the segment. Therefore, you don't have to specify a segment to _ends: it automatically closes the last one opened.

_crtseg also allows segments to be grouped together. You do this by specifying a group name. After all of the segments for a group have been created, the _defgrp macro is invoked to generate a group statement.

The ?addseg macro is used by both _crtseg and _defgrp, and relieves you of the need to specify any segment names to _defgrp by accumulating link names used in _crtseg macros. That way you don't have to keep track of which segments are being used (it could vary depending on some parameter).

?addseg takes two parameters, the group name and the link segment name. Two additional macros are defined, ?add_g and ?in_g (where "g" is the group name).

?add_g takes the segment name as its only parameter. It expands to a call of the ?in_g macro, which takes two parameters. The first is the list of segments already declared in group g, and the second is the name to be added to the list. If the new name is null, the group statement is generated. Otherwise a new ?add_g macro is defined with the expanded list. Thus, as names are added, the predefined list in the ?in_g call gets longer.

If this sounds confusing, welcome to the club (The Confusing Club). It really is helpful to make up small test programs to run through the assembler and see what is actually produced. You can instruct MASM by command line switches to generate very detailed listings that show how it processes macros and conditionals.

Another technique for variable lists of names is illustrated by the _call macro. _call takes two parameters, the name of the function being called and a list of parameters to pass on the stack. The list can be any length. The point is to write the _call macro as similar as possible to a high level language function call.

The _arg macro processes the parameter list and defines a new macro for each parameter, counting each parameter as it goes. It defines macros with names of the form "?a_i". "i" here represents the parameter number; for example, macro ?a_3 is defined for the third parameter.

Once it knows the number of parameters, the _call macro invokes the ?call macro for each argument.

The ?call macro simply expands the appropriate ?a_i macro. The order in which the arguments are placed on the stack is controlled by whether "i" is incremented or decremented for each parameter.

The following illustrates how ?call is invoked:

```
?call %?actr
```

Here the value of symbol "?actr" is the number of the parameter currently being processed. The "%" character causes MASM to expand ?call with a single argument which is a character string representing the value of ?actr.

Hope To Hear From You

These macros are part of the Tele Development Utilities available from Berry Computer. They're also included in the Tele operating system kernel. If you have any comments or want additional information, contact:

Berry Computer
P.O. Box 966
Jackson, California
95642-0966
(209) 223-0993

■■■

FORTHkit

5 Mips computer kit

$400

Includes:

Novix NC4000 micro
160x100mm Fk3 board
Press-fit sockets
2 4K PROMs

Instructions:

Easy assembly
cmFORTH listing
shadows
Application Notes
Brodie on NC4000

You provide:

6 Static RAMs
4 or 5 MHz oscillator
Misc. parts
250mA @ 5V
Serial line to host

Supports:

8 Pin/socket slots
Eurocard connector
Floppy, printer,
video I/O
272K on-board memory
Maxim RS-232 chip

Inquire:

Chuck Moore's

Computer Cowboys

410 Star Hill Road
Woodside, CA 94062
(415) 851-4362

Here's a HOT TIP!

Micro C subscribers pay only $6.00 a disk.

If you're a current U.S. subscriber to Micro C Magazine you're eligible to buy Micro C's public domain disks at a reduced rate, 25% off the regular $8.00 price. Just $6.00 each for disks full of programs that we've collected from everywhere. There's over 100 to choose from. If you're not a current subscriber, this is a bargain that's worth subscribing for. Just write or call Micro C for the latest Spring/Summer catalog and discover new disks at great savings. (Note: Non-subscribers ordering at the $6.00 rate will have the option of adding a subscription to their order or re-ordering at the $8.00 rate.)

MICRO CORNUCOPIA
P.O. Box 223
Bend, OR
97709

SCIENTIFIC SOFTWARE

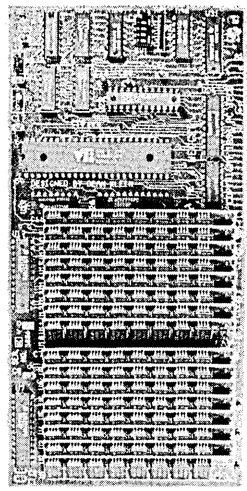
SCI-GRAF *$99.95*	Create huge hi-res plots with log or linear scaling. Screen and printer output. Automatic legends and labels. Flexible ASCII input. Works with CGA, EGA, Hercules, and mono cards. (*Plotter version forthcoming!*)
SCI-GRAF MODULES *$250.00*	Create custom hi-res graphs from within your own programs by linking to our object code. Supports all *SCI-GRAF* features, plus more! Microsoft C and Aztec C versions. No royalties.
FONTEDIT *$39.95*	Create custom Greek, math, or other symbols for use with *SCI-GRAF* or *SCI-GRAF MODULES.* Requires IBM compatibility and CGA.
SCI-CALC *$79.95*	Pop-up scientific expression evaluator, more powerful than other pop-up calculators. Complete expression editing facility. Full range of functions: scientific, statistical, logic. Requires IBM compatibility.
NanoLISP *$99.99*	New Common LISP interpreter, a subset that strictly adheres to the standard. Includes structures, bit arrays, *most* Common LISP functions, plus graphics and DOS extensions. Sample AI application programs.

Free shipping on prepaid orders. No credit cards.

MSC Microcomputer Systems Consultants

32 West Anapamu, Suite 190, Santa Barbara, CA 93101
805-963-3412

FOUR MEGABYTES LESS THAN ONE KILOBUCK!



The PC Tech Four Megger is long on a lot of things, like memory, reliability and performance. It's short on cost, power and length. The Four Megger comes complete with software for the Lotus, Intel, Microsoft Expanded Memory Specification as well as a fast memory disk program and an extensive diagnostic program. The Four Megger works in the IBM PC/XT/AT and compatibles at 4.77 or 8 MHz. All this for $850.

Another smart idea from PC Tech.

Now Available: The 16 Megger. Expanded memory for AT and XT, as well as extended memory for the AT. Call!

PC Tech

**904 N. 6th St.
Lake City, MN 55041
(612) 345-4555**

Designers of the X16 and Other Fine Computer Products

Ever Program On A Silver Platter??

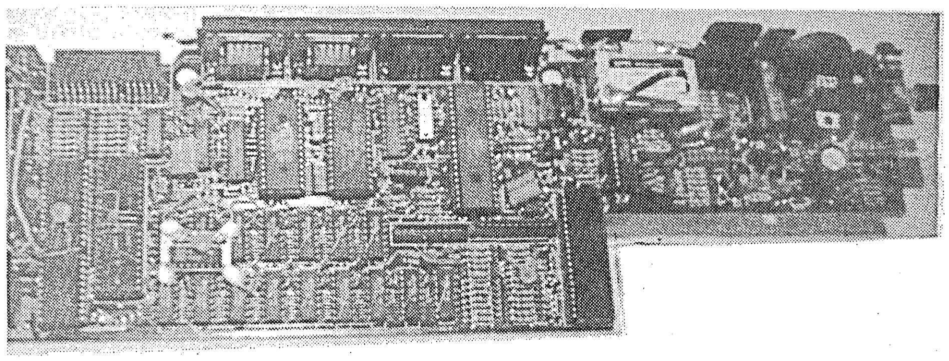
How much would you expect to pay for a 32 bit MC 68000 computer that's a mainframe condensed down into a keyboard? How about $389.00!!?? If it makes you feel any better simply add a zero to the price when you order! But that's actually our price!!! The most powerful computer money can ever buy is now the most inexpensive computer money can buy!!! So don't buy the name! Buy the power!! The power is **not** in the name!

If **you** had the opportunity to work amongst Machine Code ROM Designers, VAX & UNIX wizards in a research laboratory, designing an MC 68000 based computer that's 2nd to none. . .

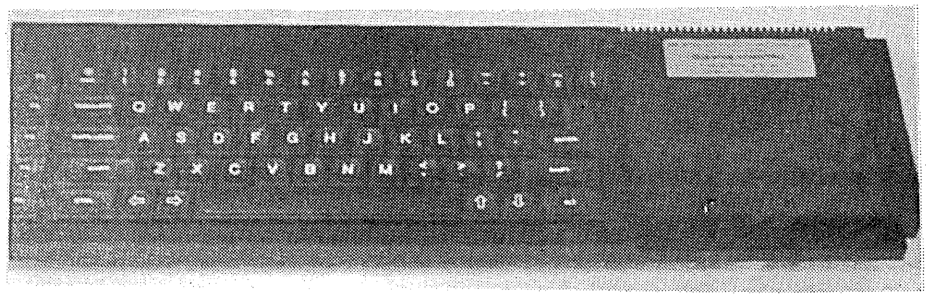
What would you come up with?? And what would you call it??

Well It's Already Been Done!!
They Called It The QL For The Quantum Leap It Is!!

Absolutely a Quantum Leap beyond what you know & use – and it's truly like Programming on a Silver Platter!!



✓A+ Computer Response Authorized Dealer



The QL Desktop Minicomputer: Designed by SRL Labs, manufactured by Samsung. An absolute Quantum Leap beyond all the rest! The phenomenal open architecture QDOS: with Virtual Memory RAM, Multitasking Job Control, Multiuser Networking. It'll Cache Files into unused Memory and create/delete Directories Automatically! Even allows File Names up to 36 characters long! Everything is built into ROM here: QDOS, Networking, Windowing, & 32 Bit SuperBasic, all in a totally concurrent non-destructive environment. Unlimited quantities & lengths allowed with: Variables, Program Lines, CONsoles & Buffers. Dynamic non-destructive virtual RAM Disking & Networking buffers too! Even a System Variables Brain Page Screen! Built-in DCE & DTE Serial Ports.

Language Environments:
Metacomco's "C", LISP, BCPL, 68000 Assembler, APL, Development Kits. Prospero's Pro Pascal & Pro Fortran 77. Digital Precision's Forth-83. QJUMP's 65C02 or 8088 Cross Assembly ROMs. Everything generates native 68000 Compiled Code. ROM Firmware & Software Package is now available which will even bring it up in CPM!

Imagine working with a 32 bit SuperBasic that's structured like Turbo Pascal, powered beyond PIC Basic, in an interpretor always present with QDOS, all concurrently running in a built-in UNIX-like multitasking job controlled environment with access to 360 fully channeled windows, devices & files by EACH job! 3 Major Compilers already exist for the SuperBasic source alone! TURBO, SUPERCHARGE, QLIBERATOR! The compiled SuperBasic code or ANY other language will multitask and control with QDOS and SuperBasic. The list of ALL the Superior Features would fill this entire publication!

The QL comes bundled WITH PSION Integrated Word Processor, Spreadsheet, Database and Presentation Graphics Programs. PLUS: Our FREEWARE Demos & Utilities with all purchases! **plus $12 ship. & handl.**

Call: (201) 328–8846

QLine BBS: 328–2919

Technical Info & Assistance – –
Telex: 9102500026
Compuserve ID # 76625,2214

Quantum Computing, Box 1280, Dover, NJ 07801

By John Paul Jones
6245 Columbia Ave.
St. Louis, MO 63139

Very Proper Procedures

We spend our whole lives learning procedures. We learned procedures when we were first tall enough to sit straight at the table, and we've been learning them ever since. Procedures lie at the heart of Pascal and Modula, and when you understand procedures you have an excellent start toward programming manners. Functions, on the other hand, were learned much earlier in life and as such won't require such close inspection.

The last portion of the declaration part of a Pascal or Modula-2 program is the procedure declaration. A procedure (most of this also applies to functions) is a subprogram which can be invoked simply by using its name. Its structure closely follows that of an entire program or module, with header, declaration, and statement parts. This implies that types, variables, and procedures can be nested within a procedure. These nested definitions will NOT be visible outside the enclosing procedure.

Figures 1 and 2 show the procedure definitions in Pascal and Modula-2 to display a greeting on the current output device. They could have been coded more simply, but I wanted to show you some of the differences between the languages.

The reserved word PROCEDURE begins the header part. Following this is the optional formal parameter list. The formal parameters must have an associated type, which can be either predefined or user defined. When a procedure is invoked with a statement like:

```
Hello (TimeOfDay, MessageSent);
```

the parameters TimeOfDay and MessageSent will be passed to the procedure.

You should also note the reserved word VAR in the formal parameter list of the examples. This signals to the compiler that the parameter is passed by reference (the variable's location in memory is passed to the procedure) so that the procedure can alter the original variable.

When a variable is passed by value (no VAR prefix in formal parameter list), the procedure receives only the current value of the variable. Any changes the procedure makes to the value will have no affect on the original variable.

Although it is permitted, and in some situations necessary, it is not considered the best practice to modify global variables within a procedure. It is better to pass variables as VAR parameters. Since it requires a conscious decision to do so, unexpected side effects are less likely.

Functions

A function is a procedure which has a type and returns a value. In general, a function can be used anywhere an expression of that data type is valid. Figures 3 and 4 are Figures 1 and 2 rewritten as functions. Pascal uses the reserved word FUNCTION in the header; Modula-2 gets double duty from the word PROCEDURE. Somewhere within the body of a Pascal function, the function's return value must

Figure 1 - Pascal Version of Hello

```
procedure
   Hello( hour : integer; var ok : boolean);
type
   message : string[20];
var
   greeting : message;
begin
   if hour < 12 then
   begin
      greeting := 'Good Morning';
      ok := true;
   end
   else
      if hour < 24 then
      begin
   greeting := 'Good Afternoon';
   ok := true;
      end
      else ok := false;
   if ok then writeln(greeting);
end;
```

Figure 2 - Modula-2 Version of Hello

```
PROCEDURE Hello
   (Hour : INTEGER; VAR ok :
   BOOLEAN);
(* the following IMPORT is more likely
done  in the enclosing Module's import
list *)

FROM InOut IMPORT WriteString,
WriteLn;
TYPE
   Message : ARRAY [0..19] OF CHAR;
VAR
   Greeting : Message;
BEGIN
   IF Hour < 12 THEN
      Greeting := "Good Morning";
      ok := TRUE;
   ELSIF Hour < 24 THEN
      Greeting := "Good Afternoon";
      ok := TRUE;
   ELSE
      ok := FALSE;
   END;
   IF ok THEN
      WriteString(Greeting);
      WriteLn;
   END;
END Hello;
```

be assigned:

```
Hello := ok;
```

In Modula-2, the RETURN statement assigns the return value and also immediately exits the function. More than one RETURN statement is permitted in a function procedure to allow for multiple exit points. A Modula-2 function should NEVER "fall out the bottom" since its return value will be undefined. A RETURN statement can also be used to exit a procedure at any point:

```
IF CurrentValue = MaxAllowed
    THEN RETURN END;
```

A function is invoked by using its name on the RIGHT side of an assignment statement:

```
MessageSent := Hello(TimeOfDay);
```

In both Pascal and Modula-2, you can't use the variable which the function name represents within the function (you can't use the variable name 'Hello' within function 'Hello()'), other than to assign the returned value in Pascal.

This is because it would be interpreted as a recursive call to the function (a function calling itself). That's why the temporary var "ok" was declared in the function versions of Hello. See the comment in the listings.

A definition of, or a call to, a parameterless function in Modula-2 requires a pair of empty parentheses:

```
PROCEDURE Random();...
NewValue := Random();
```

Pascal provides a large number of built-in procedures and functions, while Modula-2 has only a few. Modula's "missing" procedures need to be IMPORTed from the standard library modules, user-written modules, or defined in the current module. The IMPORT statement can take two forms:

```
FROM ModuleName IMPORT Id1, Id2,..Idn;
IMPORT ModuleName;
```

In the first form, only the specified identifiers are available for use (one exception, IMPORTation of an enumerated type also imports the constants which make up the enumeration). In the second form, all of the EXPORTED identifiers become available. To use them, however, their names must be qualified with the source module's name:

```
Result :=
ModuleName.FunctionName(Parameter);
InOut.WriteString('This is a test.');
```

Two procedures with the same name from different modules can be used in this way.

Standards

There are efforts (mainly by the British Standards Institute) in the works to define an international standard for Modula-2. I hope the final recommendations will be reasonable and completed before a de facto standard has emerged. (As Turbo Pascal has become a de facto standard for small computers.) The ISO standard for Pascal came much too late to be of significant impact. Perhaps once published, the major Modula-2 compilers will be updated to match the standard.

There are several of the BSI's preliminary proposals which could significantly affect existing compilers and programs. This may make their adoption less likely, despite the fact that the standards should benefit the entire Modula community.

Some of the notable proposals are:

1. Adding a string data type
2. Removing several built-in functions
3. Expanding SETs to provide SET OF CHAR
4. Functions returning values of ANY type
5. Explicit EXPORTs required in definition modules
6. Removing NEW and DISPOSE from the language (this last agrees with Wirth's 3rd edition of "Programming in Modula-2").

Assembly Language Modules

Most of the programming that I get paid for is done in assembly language (multi-tasking, real time telephone call processing), so I consider a convenient interface to assembly language modules or code fragments an important feature of a high level language. I realize that Modula-2 is designed with facilities that should make assembly coding unnecessary, but in the real

Figure 3 - Pascal Function,Hello.

```
function Hello( hour : integer) : boolean;
type
   message : string[20];
var
   greeting : message;
   ok : boolean;
begin
   if hour < 12 then
   begin
   greeting := 'Good Morning';
   ok := true;   { could use 'Hello' here }
   end
   else
      if hour < 24 then
      begin
   greeting := 'Good Afternoon';
   ok := true; { could use 'Hello' here }
   end
   else ok := false; {could use 'Hello' here}
   if ok then writeln(greeting);
                        {cannot use 'Hello' here}
   Hello := ok;
end;
```

Figure 4 -Modula-2 Module, Hello

```
PROCEDURE
   Hello (Hour : INTEGER) : BOOLEAN;
FROM InOut IMPORT WriteString,
WriteLn;
TYPE
   Message : ARRAY [0..19] OF CHAR;
VAR
   Greeting : Message;
   ok : BOOLEAN;
BEGIN
   IF Hour < 12 THEN
      Greeting := "Good Morning";
      ok := TRUE;
   ELSIF Hour < 24 THEN
      Greeting := "Good Afternoon";
      ok := TRUE;
   ELSE
      ok := FALSE;
   END;
   IF ok THEN (* cannot use 'Hello' here *)
      WriteString(Greeting);
      WriteLn;
   END;
   RETURN ok;
END Hello;
```

(continued next page)

(continued from page 79)

world, it just 'taint so.

Turbo Pascal allows imbedded machine code with its INLINE statement, and the compiler will even plug in the actual values for defined identifiers for you. Also, the MS-DOS and CP/M-86 versions of Turbo allow loading of machine code files (position independent code only!) at run time.

Logitech's Modula-2/86 has a primitive version of Turbo's INLINE statement, CODE. This allows machine code to be imbedded in the module, but symbolic identifiers cannot be used. I assume this is because the compiler has no idea what their run-time values will be, as they are not totally defined until either the link or the run-time load.

An assembler module must be incorporated into the run time system with routines included to initialize an entry point table (via a software interrupt) and linked with the other assembler modules the system needs. Then a definition module is written in Modula-2. After this, the definition module is compiled and the EXPORTed identifiers can be used. I find this process cumbersome at best.

FTL Modula-2 makes assembler modules easy. The compiler includes an assembler which generates standard .SMR files (the normal output from implementation modules) from assembly language source. The only thing left to do is write and compile a definition module in Modula-2. (Actually, this has to be done first.) This provides the best of both worlds - assembler is easily used for the time critical portions of a program, while the "leisurely" portions can be rapidly coded in Modula-2.

FTL Modula-2 Revisited

FTL Modula-2 is undergoing a series of planned improvements. I reviewed version 1.0 in issue 34, but currently the compiler is up to version 1.21. There have been significant upgrades.

They've added data types LONGINT, LONGCARD and LONGWORD, all 32-bits long. They've also added the associated conversion functions SHORT, LONG, and LONGTRUNC.

The compiler can now (optionally) include range-checking code. This is especially important during the early phases of module development.

Unexpected things happen to the best programmers and often these can

Figure 5 - Search Disk For Text String

```
program find;
{Search a set of files for a text string}
{Output is all lines in text files that contain}
{search string, each numbered with position}
{in file Input is file made up of unambiguous}
{file names each on a separate line. The last}
{line should NOT be terminated with a}
{carriage return / line feed.}
{The program is set up to run as a .COM file}
{with command line switches as below}
{find pfu validfilename}
{P switch sends program's output to lst: device,}
{F causes prog. to expect command line input file}
{list file name & U indicates case should be ignored}
{If F switch not entered, default file list file used}
{Only minimal error checking performed}
TYPE
  parameter = string[20];
  bigstring = string[255];
  path_n_file = string[32];
VAR
  NameList, CurrentFile : text;
  InputString : bigstring;
  st_to_find : string[80];
  stlen : byte absolute st_to_find;
  listname, filename : path_n_file;
  first_char : integer;
  found, ignore_case, print, NamesFromFile :
       boolean;
  i : integer;
  ch : char;

PROCEDURE SetFlags;
{ Set global flags based on command line, }
{ If F flag, get input file name from command line }
VAR
  i : integer;
  CmndLine : parameter;
BEGIN
  IF paramcount <> 0
  THEN BEGIN
     CmndLine := paramstr(1);
     { if have switches, convert to caps }
     for i := 1 to length(CmndLine) do
  CmndLine[i] := upcase(CmndLine[i]);
  END
  ELSE CmndLine := '';
  NamesFromFile := pos('F',CmndLine) <> 0;
  print := pos('P',CmndLine) <> 0
  ignore_case := pos('U',CmndLine) <> 0;
  IF NamesFromFile
     THEN listname := paramstr(2);
END;

PROCEDURE OpenFileList;
BEGIN
  IF not NamesFromFile
     THEN listname := 'file.lst';
     { use default if no input }
  {$i-}
  assign(NameList,listname);
  reset(NameList);
  {$i+}
  IF ioresult <> 0 THEN { if no input file list }
  BEGIN
     writeln('Input file list not found!');
     halt;
  END;
END;

PROCEDURE checkline;
{ The input line could be passed as a parameter,
but do you realize how much time it takes to put all
that data on the stack? }
VAR
  localline : bigstring;
  linelen : byte absolute localline;
  local_st : string[80];
  i : integer;
BEGIN
  localline := InputString;
  IF ignore_case THEN for i := 1 to linelen do
     localline[i] := upcase(localline[i]);
  found := pos(st_to_find,localline) <> 0;
END;

PROCEDURE expand_tabs;
{ many printers don't understand tabs }
{ while tab present,  insert spaces to tab stop
     replace tab char with a space }
CONST
  tabstop = 8;
BEGIN
  while pos(^I,InputString) <> 0 do
  BEGIN
     while pos(^I,InputString) mod tabstop <> 0 do
        insert(' ',InputString,pos(^I,InputString));
     InputString[pos(^I,InputString)] := ' ';
  END;
END;

PROCEDURE process_file;
{ read file a line at a time
     if search string present, output the line }
VAR
  file_line : integer;
BEGIN
  {$i-}
  assign(CurrentFile,filename);
  reset(CurrentFile);
  {$i+}
  IF ioresult = 0 THEN { if file found }
  BEGIN
     file_line := 1;  { line # in file }
     while not(eof(CurrentFile)) do
     BEGIN
        readln(CurrentFile,InputString);
        checkline;  { search string present? }
        IF found THEN
        BEGIN
           IF print THEN expand_tabs;
           writeln(file_line:5,': ',InputString);
        END;
        file_line := succ(file_line);
     END;
     close(CurrentFile);
     writeln;
  END;
END;

BEGIN
  SetFlags;
  OpenFileList;
  Writeln;
  write('Enter string to find: ');
  readln(st_to_find);
  IF ignore_case THEN for i := 1 to stlen do
     st_to_find[i] := upcase(st_to_find[i]);
  IF print THEN   { re-direct output to printer }
     conoutptr := lstoutptr;
  IF print THEN
  BEGIN
     writeln('Searching for: ',st_to_find,'');
     writeln;
  END;
  while not(eof(NameList)) do
  BEGIN
     readln(NameList,filename);
     writeln(filename);
     process_file;
     writeln;
     IF not(print) THEN {if screen output, read it}
     BEGIN
        writeln('Press any key...');
        read(kbd,ch);
     END;
     writeln;
  END;
  close(NameList);
END.
```

be caught with range checks. Once debugged, the module can be re-linked without the range-checking code.

Optional 8087 numeric coprocessor support ($30) is now available. Preliminary results show that transcendental functions are faster than Logitech's Modula-2/86, but that standard +, -, * and / are a bit slower. These are still massively faster than the software floating point routines for either compiler.

The editor has been improved. One feature I especially like is autotab (like Turbo Pascal's editor which on carriage return can automatically tab to the position of the first non-blank, non-tab character on the previous line). The second is a choice of tab widths (1,2,4 or 8). Automatic line wrap has also been added. Screen updates can be delayed until vertical retrace, thus eliminating the "snow" associated with some video cards.

Not new, but one FTL extension I did not mention in issue 34 is that SETs can contain as many as 1024 elements. I like this extension since I often use SET OF CHAR.

The large memory model of FTL is in the works. When completed, the entire address space of the 8086/8088 will be available for code and data. The basic floating point math package is being rewritten to improve speed. Also a 68000 implementation is in development, but I don't know for which machine. It would be a nice package to run on a DSI coprocessor board, wouldn't it?

Registered owners can upgrade to newer versions for $15 and purchase the package for a different processor for a reduced price.

Useful Stuff

As I mentioned, I spend most of my profitable time doing assembly language coding. Although much of what I'm doing is new, I do have to spend a considerable amount of time maintaining and upgrading an existing software base. I wrote the Turbo Pascal program in Figure 5 to help in the maintenance. When you've got nearly half a megabyte of Z80 assembler to search for a particular global label, it can take quite a bit of time, even with a fast editor.

The program can search all my source files in a couple of minutes on a stock PC with a 20 Meg hard drive. Keep in mind that this program is *not* an example of good programming style, but a "quick and dirty" solution to a problem. Also, it leans very heavily on Turbo-specific extensions and would require modification for another compiler. The program runs equally well under both CP/M-80 and MS-DOS.

If you feel that you're far enough along, you might tackle translating it into Modula-2.

Next Time

Next time I'll take a quick spin through the statements available in the two languages, including the very important looping statements, flow control, and multi-way branch statements.

Products Mentioned:

Turbo Pascal
(CP/M-80, MS-DOS, CP/M86)
Borland International
4585 Scotts Valley Drive
Scotts Valley, CA 95066

Logitech Modula-2/86 (MS-DOS)
Logitech Inc.
805 Veterans Blvd.
Redwood City, CA 94063

FTL Modula-2 (CP/M-80, MS-DOS)
Workman and Associates
1925 East Mountain Street
Pasadena, CA 91104

■■■

QUALITY SOFTWARE YOU CAN AFFORD!

NEW Z80 Symbolic Debugger

- Only $49.95 plus shipping.
- Screen oriented with a simultaneous display of instruction mnemonics, register, stack, and memory values.
- Breakpoints may be set on any combination of fixed memory address, régister values and/or memory values.
- Uses Digital Research compatible SYM files.
- Supports Hitachi HD64180.

Relocatable Z80 Macro Assembler

- Only $49.95 plus shipping.
- 8080 to Z80 Source Code Converter.
- Generates Microsoft compatible REL files or INTEL compatible hex files.
- Compatible with Digital Research macro assemblers MAC & RMAC.
- Generates Digital Research compatible SYM files.
- Conditional assembly.
- Phase/dephase.
- Cross-reference generation.
- Full Zilog mnemonics.
- INCLUDE and MACLIB FILES.
- Separate data, program, common, and absolute program spaces.
- Supports Hitachi HD64180.
- Z80 Linker and Library Manager for Microsoft compatible REL files available as an add-on to Assembler.

ATTENTION Turbo Pascal Users: Assembler will generate Turbo Pascal in-line machine code include files.

TO ORDER, CALL TOLL FREE:
1-800-367-5134, ext. 804
For information or technical assistance:
(808) 623-6361
Specify desired 5¼" or 8" format. Personal check, cashier's check, money order, VISA, MC, or COD welcomed.

MITEK

PRICE LIST
Z80 Macro Assembler: $49.95
Assembler, Linker, and Library Manager: $95.00
Manual Only: $15.00
Z80 Symbolic Debugger: $49.95
Manual Only: $15.00
Assembler, Linker, Library Manager, and Debugger: $134.95
Include $5 for shipping and handling.

P. O. Box 2151
Honolulu, HI 96805

Z80 is a trademark of Zilog, Inc. MAC and RMAC are trademarks of Digital Research, Inc. Turbo Pascal is a trademark of Borland International, Inc.

By Frank A. Kurucz
2106 Via Robles
Oceanside, CA 92054

Z80 SIO Interrupts On The Kaypro 4

I remember when I was first introduced to interrupts (the microprocessor kind). They seemed very mysterious and very magical. They still seem that way, but I also know they are very powerful and a lot of fun. (Plus Powdermilk interrupts give shy programmers the will to do what needs to be done.)

Some months ago I was asked to write a program to simulate a dispatcher's console. The console had to communicate with a channel interface monitor via an RS-232 serial interface. Since the monitor could send data to the console at any given moment, polling the serial port was unacceptable because the console often had to perform other functions such as editing messages to send to the monitor.

To my dismay, I discovered that neither PC-DOS nor CP/M had a built-in way of convincing serial ports that they should create interrupts.

Since I was unable to find any code on the market that supports interrupt-driven serial ports for a Kaypro 4/83, I decided to write my own.

Why Interrupt-Driven Serial Ports?

Suppose you will be receiving data through a serial port but can't always be polling it. Interrupts make it easier for your program to synchronize with the external device transmitting data.

One common complaint heard is that the Kaypro 4 scrolls very slowly (the scroll is performed by software). Another is that if you are echoing data read from a serial port at a rate of more than 2400 baud to the screen, you might lose a character during a scroll because more than 3 characters are waiting to be read from the serial port (the Z80 SIO has a 3 byte buffer). By using interrupts I have been able to echo characters at a rate of 19200 baud without dropping characters.

Microprocessors are endowed with interrupts to synchronize software with uncontrolled external events. Interrupts are perhaps one of the most neglected features that a computer has. They are hardly ever covered in university courses. Ask a computer science major what an interrupt is and he'll probably say he doesn't appreciate being interrupted. (No offense intended to C.S. majors, after all I majored in C.S. myself.)

Interrupts can also improve the efficiency of a program by relieving it of the drudgery of polling.

Editor's note: Assume that your system will be receiving random spurts of data at 9600 baud. That means that when data's coming in, the SIO would have a new 8-bit character about 960 times a second. So your processor would have to be polling the SIO at least 960 times a second and risk losing characters. In an interrupt-driven system, the Z80 ignores the SIO until the serial chip yanks on the Z80's interrupt line saying, "I have a character, come and get it." If the interrupt routine is quick, there's no chance of losing any characters, and, the processor doesn't waste all of its time asking the SIO if it has a new character.

Thus, interrupts make software elegant and reliable. Of course, the next question is how to use them.

Z80 SIO

The Kaypro 4/83 has one Z80 SIO (the 84 series Kaypros have two SIOs). The SIO has two serial channels which can function simultaneously. They are referred to as channel (or port) A and channel B. Each channel has Read and Write registers used for programming the respective ports and for reading the status of the channel. There are three Read registers (0 - 2) and eight Write registers (0 - 7).

Channel A is missing the #2 Read and Write registers because the data associated with these two registers is common to both channels (it is the interrupt vector). Both channels have a 3 byte buffer. If it overflows, the incoming byte overwrites the last byte entered. This is why you can type ahead only 3 characters.

The keyboard is connected to channel B of the SIO, and it is polled rather than interrupt-driven. The other port, channel A, attaches to the serial port connector on the back of the machine. We can do whatever we want with this channel.

Programming The SIO

You will need only the Write registers to program the SIO. All of the write registers are accessed through the control or command port, which in the Kaypro has an address of 6 for channel A and 7 for channel B. Channels A and B also have data ports with respective addresses 4 and 5.

Control Register 0 is used for selecting the other seven registers, and it has a few functions of its own, one of which is resetting the channel. To output data to one of the other registers, first output the register number to register 0 (simply output it to the command port), then output the desired data for that register to the command port.

After data is output to the desired register, register 0 is automatically reselected for the next output. I won't explain each register in detail, but the programming examples should help in illustrating their use.

Interrupt Vectors

The Z80 has three interrupt modes -

0, 1 and 2. We will use mode 2 interrupts to implement our system. Mode 2 interrupts allow us to locate the interrupt vector anywhere in memory. When we use this mode, it is necessary to set the interrupt register in the CPU to the desired page. A page is a 256 byte portion of memory that begins at an address that is a multiple of 256 (100H), thus we say that a page begins at some address XX00H.

Editor's note: There are 256 pages (8 bits = 0 - 255) of 256 bytes each. The total is, of course, a 16-bit address or 64K bytes.

The interrupt vector contains the addresses of the different interrupt service routines. When a device causes a mode 2 interrupt, the following occurs: the device outputs 8-bits onto the data bus. That 8-bits is taken by the Z80 as the least significant 8-bits of the address to the interrupt routine. The 8 most significant bits come from the Z80's interrupt register.

With these two bytes an address is formed. At this composite address, the Z80 expects to find another address, the address of the code which will service the interrupt. (Also, the return address is stored in the stack.) Control is transferred to the interrupt code.

The Z80 SIO has four interrupt vectors per channel: output data, status, input data, and input error. Channel B starts at address XX00 and channel A starts at address XX08. Note that a normal RET (return) instruction is inadequate for returning from an interrupt, therefore the RETI (return from interrupt) instruction must be used (I implemented it in 8080 using constants).

The Interrupts

There are four interrupt routines implemented in assembler. They are:

INPINT - This routine handles data received by the SIO, storing it if possible in a circular queue (FIFO - first in, first out) from where it can be retrieved at the system's leisure.

OUTINT - This routine outputs a byte to the SIO if there is any data present in the output. If the FIFO is empty, OUTINT resets the interrupt until more data is put into the FIFO. A state variable is used to keep track of the interrupt status. If the interrupt is disabled when attempting to load the FIFO, OUTINT re-enables it.

EXTINT - This is the external status interrupt. It is used for monitoring changes in the RTS/CTS and DTR/DSR signals. This code can be customized, but for now all it does is reset the interrupt.

ERRINT - This interrupt becomes active when an error is detected for incoming data (parity, framing, or overrun errors). Again, all this code does is reset the interrupt; no error handling is implemented.

Interface Software

I wrote a high level interface for the interrupt system in Turbo Pascal. There are three routines:

INITSIO - Initializes the interrupt system, sets up the serial port characteristics, and initializes the FIFOs and their pointers.

WRITEBYTE - Puts outbound data into the output FIFO or outputs it directly (re-enabling the output interrupt), depending upon the interrupt status.

READBYTE - Returns a byte from the input FIFO. If no byte is present, a boolean variable passed as a variable parameter will be set to true.

Listings

I've included three pieces of code: SIOINT.ASM, SIOLIB.INC, and

(continued next page)

ELF

Introducing "ELF"

ELF (Extended Lexicon FOCAL) interpreter is more "basic" than BASIC, a simple and very Easily Learned Facility. If you want capability without complexity, ELF is for you. Look what ELF can do without peeks, pokes, or assembly language:

```
1.10 Ask !,"Disk Drive No.? ",DX
1.15 /U;/XDOS 0,DX,036H;T x8.0
1.20 Set TB=FAX(0)*FCX(0)*FDX(0)
1.25 Set RB=FAX(0)*FBX(0)*FCX(0)
1.30 Type !,"Total Bytes:   ",TB
1.35 Type !,"Free Bytes:    ",RB
```

This program may be further abbreviated; only the first letter of each command is required. ELF programs are ordinary ASCII text files. ELF is not a compiler, but will produce run-time versions (.COM files) of programs without revealing source.

ELF comes with tutorial/help files, useful program examples and an ELF Programmer's Manual for $45 plus $5 shipping ($10 non-USA), VISA, MC or check.

118 SW First St. - Box G
Warrenton, OR. 97146

Micro Methods, Inc.

Phone (503) 861-1765

XEROX 820-1 AND 820-2 ITEMS
Reconditioned, Assembled and Tested

```
820-1 8" COMPUTER SYSTEM......................$330.00
   5 1/4" COMPUTER SYSTEM.....................$350.00

820-2 8" COMPUTER SYSTEM......................$395.00
   5 1/4" COMPUTER SYSTEM.....................$415.00

820-1 COMPUTER MONITOR (COMPLETE).............$125.00
820-2 COMPUTER MONITOR (COMPLETE W/CONTROLLER)$195.00
820 COMPUTER MONITOR (NO MAIN BOARD)..........$ 85.00

HIGH PROFILE KEYBOARD (COMPLETE)..............$ 45.00

820-1 MAIN COMPUTER BOARD.....................$ 50.00
   FULLY POPULATED BOARDS, AS IS (NEED REPAIR)$ 20.00
820-2 MAIN COMPUTER BOARD.....................$ 70.00
   FULLY POPULATED BOARDS, AS IS (NEED REPAIR)$ 30.00
820-2 FLOPPY CONTROLLER BOARD.................$ 95.00

DUAL 8" SSDD DISK DRIVES/ENCLOSURE (COMPLETE).$175.00
DUAL 8" DISK DRIVE CABINET (NO DRIVES)........$ 75.00

5 1/4" DUAL DISK DRIVE CABLE..................$ 20.00
8" DUAL DISK DRIVE CABLE......................$ 35.00
RS-232 CABLES.................................$ 10.00
```

```
LINE CORDS.............ea.$3.00
Z80-B 6MHz.............ea.$3.00
Z80-H 8MHz.............ea.$9.50
5 1/4" DSDD DISKETTES...ea.$ .60
8" SSSD DISKETTES.......ea.$1.25
DC300A DATA CART..USED...2/$5.00
```

E^{2}I COMPUTER PRODUCTS
2273 AMERICAN AVE. #8
HAYWARD, CA 94545
(415) 786-9203

TERMS: Pre-payment, COD, Visa/Mastercard. California residents add sales tax. Orders are FOB Hayward, CA.. Shipments by UPS Ground unless otherwise requested. Prices and availability are subject to change without notice. All products are assembled and tested and have a 30 day warranty unless otherwise stated. Call or write for current product and price listing. Xerox is a trademark of Xerox Corporation. CP/M is a trademark of Digital Research.

(continued from page 83)

TEST.PAS. You'll find them on the Micro C bulletin board (503-382-7643).

SIOINT.ASM is the interrupt code. I wrote the code in assembler so I could put it anywhere in memory. Also, I used 8080 mnemonics because not everyone has a Z80 assembler.

The code resides at address $E000 in order to isolate it from the bank-switched part of the computer's memory, which is from $0000 to $3FFF. (It would be tragic to have the system interrupt only to have the ROM and video display memory selected instead of the interrupt vector.)

The interrupts store and retrieve their data from two FIFOs, which are 256 bytes long. The FIFO size of 256 ($100) was chosen because it simplifies the task of FIFO pointer manipulation. (Remember the golden rule: Keep Interrupts As Short As Possible.) First assemble the code using the CP/M assembler (ASM), and use the LOAD utility to convert it into a COM file.

SIOLIB.INC initializes the serial port and sets up the interrupts. It also provides an interface to the FIFOs. Written in Turbo Pascal, it is entirely self-contained. Simply use the Include directive to insert it into your program.

TEST.PAS is a sample program that transmits 80 repetitions of the key pressed out to the serial port and echoes any characters received to the screen. If you have only one computer (as most people do), you can loop the data back to the machine by connecting together pins 2 and 3 on the serial port connector.

Final Comments

Remember to set the End Of Memory in Turbo Pascal to $E000 before compiling the program to a COM file. If you don't, you run the risk of overwriting the interrupt routines. For this same reason, the interrupt routines can't be used with programs being run under the Memory option. Before running the Turbo Pascal program, load the interrupts by running SIOINT.COM. This could be done by a submit file.

In addition to communications applications, this code has other uses. By modifying it to operate on channel B, you can implement interrupt driven and buffered keyboard data handling. The code could also be used for implementing a pacer interrupt (a periodic interrupt), which could be used in a multitasking kernel.

Another twist to this could be implementing the interrupts in Turbo Pascal, taking care to make sure that the interrupt service routines are not in bank-switched memory. Remember though that interrupts in high level languages are not usually recommended, especially for high frequency interrupts. Because the code generated is often larger and slower than its assembly equivalent, it causes the rest of the software in the system to slow down. The longer interrupts steal more CPU cycles than more efficient interrupts would. This is one place where efficient code can really make a difference.

References

- Introduction to the Z80 Microcomputer by Adi J. Khambata.
- Z80 Assembly Language Subroutines by Leventhal & Saville.
- The Programmers CP/M Handbook by Andy Johnson Laird.
- Kaypro II & Kaypro 4 Theory of Operation by Dana Cotant.
- Mostek 1984/1985 Microelectronic Data Book. ■■■

Does this look familiar? What if each change you made to your program was ready to test in seconds instead of minutes?

"The SLR tools will change the way you write code. I don't use anything else.", Joe Wright

RELOCATING MACRO ASSEMBLERS • Z80 • 8085 • HD64180

- Generates COM, Intel HEX, Microsoft REL, or SLR REL
- Intel macro facility
- All M80 pseudo ops
- Multiple assemblies via command line or indirect command file
- Alternate user number search
- ZCPR3 and CP/M Plus error flag support, CP/M 2.2 submit abort
- Over 30 user configurable options
- Descriptive error messages
- XREF and Symbol tables
- 16 significant characters on labels (even externals)
- Time and Date in listing
- Nested conditionals and INCLUDE files
- Supports math on externals

$49.95

requires Z80 CP/M compatible systems with at least 32K TPA

SLR Systems

1622 N. Main St., Butler, PA 16001
(412) 282-0864 (800) 833-3061

• Z Best Sellers •

Z80 Turbo Modula-2 (1 disk) $89.95
The best high-level language development system for your Z80-compatible computer. Created by a famous language developer. High performance, with many advanced features; includes editor, compiler, linker, 552 page manual, and more.

Z-COM (5 disks) $119.00
Easy auto-installation complete Z-System for virtually any Z80 computer presently running CP/M 2.2. In minutes you can be running ZCPR3 and ZRDOS on your machine, enjoying the vast benefits. Includes 80+ utility programs and **ZCPR3: The Manual**.

Z-Tools (4 disks) $169.00
A bundle of software tools individually priced at $260 total. Includes the ZAS Macro Assembler, ZDM debuggers, REVAS4 disassembler, and ITOZ/ZTOI source code converters. HD64180 support.

PUBLIC ZRDOS (1 disk) $59.50
If you have acquired ZCPR3 for your Z80-compatible system and want to upgrade to full Z-System, all you need is ZRDOS. ZRDOS features elimination of control-C after disk change, public directories, faster execution than CP/M, archive status for easy backup, and more!

DSD (1 disk) $129.95
The premier debugger for your 8080, Z80, or HD64180 systems. Full screen, with windows for RAM, code listing, registers, and stack. We feature ZCPR3 versions of this professional debugger.

Quick Task (3 disks) $249.00
Z80/HD64180 multitasking realtime executive for embedded computer applications. Full source code, no run time fees, site license for development. Comparable to systems from $2000 to $40,000! Request our free Q-T Demonstration Program.

Echelon, Inc.

Z-System OEM Inquiries Invited.
Visa/Mastercard accepted. Add $4.00 shipping/handling in North America, actual cost elsewhere. Specify disk format.

885 N. San Antonio Road • Los Altos, CA 94022
415/948-3820 (Order line and tech support) Telex 4931646

CP/M™ Software: Saving The Best for Last

Plus: The Greatest **ConIX**™ Giveaway

CP/M: Some people love it, others love to hate it, but most still use it. Its users complain that most software companies have abandoned it. Very true, yet we haven't*!* We've been selling the ConIX software line for many years; we developed it, we market it, and we support it - completely*!* ***What?! You haven't tried it?*** Saving the best for last, eh*?* *Don't wait!* Support *your* CP/M software company - try ConIX for *as low as $10!* What's more, you could even get lucky and receive *your entire order* ***FREE!*** See details below.

ConIX™ Operating System

An extensive upgrade for 48K+ CP/M 2.2/3.0 and equivalent systems. Provides professional capabilities with blinding speed, as often found on high-end UNIX™ machines. Installs easily in just minutes to add over 100 new built-in commands and features while maintaining 100% compatibility with all your existing software*!* Includes I/O redirection, aliases, improved user area access, auto-searching, PF Keys, Screen Paging, Print Spooler, Archiver, New SysCalls, ... Eliminates many points of user frustration with CP/M. Uses only 1/2K TPA, 0-27K disk minimum.

Included *FREE* with commented source is the Pull-Down Menu System, a user-friendly interface to ConIX. Loads with a single keystroke*!*

ConIX is the greatest, most powerful 8-bit upgrade, with speed and capabilities that are so incredible it's bringing users *back* to CP/M*!*

ConIX™ Programming System

A structured programming language for ConIX extends CP/M SUBMIT capability. Adds conditionals, loops, subroutines, labels, nesting, interrupt processing, error traps, and debugging facilities. Design intricate menu systems and command-automation shells. Also includes a special source-code "compiler" that provides string and numeric variables. An absolute *must* for CP/M power-users and developers*!*

ConIX™ Library Vol. I XCC Utilities

Over 25 utilities for ConIX written in the shell language, including hierarchical directories with overlay - adds pathname capability to existing software, interactive debugger, move/copy/link multiple files, print files with pagination, review disk files for deletion, unerase disk with stats, full-screen TYPE, and more. Source code included*!*

ConIX™ Shareware Version

A new Shareware version of the ConIX O.S. includes our regular distribution software less the Archiver, On-Line Manual, Menu source code, and some satellite utilities. ConIX Shareware is available through CHI for just the cost of the diskette and shipping, or on-line via many popular bulletin board systems. Register by purchasing regular ConIX.

ConIX™ Disk Manual Version

To reduce the cost for those who want to purchase only the ConIX O.S., we are offering the complete software package with documentation provided on disk. The disk manual has each chapter stored in individual files, excluding the Chapter Summary, Chapter Reference, and Index sections that come standard in our regular typeset manuals.

The Great Giveaway: Every 100th ConIX Order *FREE!*

That's right*!* Every 100th order processed by our computer will be shipped with a Credit Certificate for the total purchase price or $100, whichever is lower. This credit may be used toward a future purchase from CHI, or may be redeemed for cash within ninety (90) days of receipt. Your odds are an incredible *1 in 100!*

Offer applies only to private individuals and non-profit institutions ordering directly from CHI. Orders placed by PO or purchased for commercial use are not eligible. To enter, certify eligibility by signing order form.

Product Trademarks - CP/M: Digital Research Inc., ConIX: Computer Helper Industries Inc., UNIX: AT&T Bell Labs.

Name: ______________________
Company: ______________________
Address: ______________________
____
Tel. #, Hours: ______________________

End-User Software Licenses:

ConIX Operating System	$ 29.95	$____
Disk Manual Version	$ 19.95	$____
Limited Shareware Version	$ 0.00	$____
Printed Manual Only	$ 9.95	$____
ConIX Programming System	$ 29.95	$____
Printed Manual Only	$ 9.95	$____
ConIX Library I XCC Utilities	$ 24.95	$____
Printed Manual Only	$ 9.95	$____
All ConIX Packages Above	$ 69.95	$____

Computer Brand:

____

Software Distribution Disk Format: *

8" SSSD Standard	$ 5.00	$____
5-1/4" DSDD 48 TPI Soft Sector	$ 5.00	$____
5-1/4" SSDD 48 TPI Soft Sector	$ 6.00	$____
5-1/4" _S_D__TPI____Sector	$ 10.00	$____

* Add only one format charge per order.

Format Brand:

____

Shipping Information:

UPS Ground USA	$ 4.50	$____
US Mail USA (For P.O. Boxes)	$ 6.50	$____
Air Mail Canada	$ 9.50	$____
Air Mail Foreign	$ 12.50	$____
Subtotal:		$____
Sales Tax (N.Y. Residents Only):	x___%	$____
Total (Thank You!):		$____

I Certify ConIX Giveaway Eligibility:

X______________________

POs and UPS CODs accepted by phone only. Sorry, credit card payment unavailable. Personal checks require 10 days to clear. Non-USA orders must be prepaid by bank draft in US $. Delivery in 2-4 weeks.

Computer Helper Industries Inc.
Post Office Box 680
Parkchester Station, N.Y. 10462
(212) 652-1786 9AM-5PM M-F

Print File To Symbol File Convertor

By Dan Griffith
95 Clark St.
New Haven, CT 06511-3803

A Handy Accessory For CP/M Assemblers

Symbol tables are a real pain if you have to enter them manually. Here's a way to get your system to create them for you.

Micro Cornucopia Kaypro Disk K25 contains Z80MR, an excellent Macro Assembler. Before the word macro scares you off, let me say that this article is not going to discuss macros. What it is going to discuss is the print (.PRN) file Z80MR produces and a way to convert that file into a symbol table (.SYM) file that can be used by Z8E, DASM, and possibly other programs.

The .PRN File

For each program that is assembled, Z80MR produces (unless told otherwise) a print file which contains error and warning messages, assembled code and, at the end, a symbol table. Symbols are generally used to represent addresses or constants. Figure 1 contains a .PRN file listing of a short program that prints "Hello, world!" 100 times. The symbols used in the program are BDOS, START, LOOP and HELLO.

The .SYM File

The symbol table produced by Z80MR contains symbol names listed alphabetically followed by their numeric address (in hexadecimal). This format creates two problems. First, because the table is imbedded in the print file, other programs cannot find it. Second, Z8E and DASM expect each symbol to be listed as a hexadecimal address followed by the symbol name. In addition, DASM requires each symbol to be on a separate line. If working on the program shown in Figure 1, Z8E and DASM would want the symbol table to look like the following:

```
0005 BDOS
0113 HELLO
0102 LOOP
0100 START
```

PRNSYM To The Rescue!

PRNSYM is a fairly simple program that is the solution to this dilemma. It will read through a .PRN file until it finds the line labelled "ASEG SYMBOLS" (see Figure 1). When it finds this line, it has found the start of the symbol table. All it does then is read each symbol and address, reverse the order, and output it to a .SYM file. The complete program in Turbo Pascal is listed in Figure 2.

There are no really tricky parts to the program, but I'll briefly discuss three aspects.

First, Turbo Pascal 3.0 internal variables ParamCount and ParamStr are used to make the program a little friendlier. If the user runs the program with no file name given, PRNSYM will detect this and prompt for a file name. If the user enters one file name, PRNSYM will use it as the input file and will create a symbol file with the same first name, but with .SYM as the extension. Finally, if the user enters two (or more) file names, the first will be used as input (the .PRN file) and the second will be used as the output (.SYM) file. Any additional names are ignored.

Second is the use of the internal variable IOResult in error trapping. Whenever Turbo Pascal executes any input or output (I/O) statement, it sets a flag and stores the result in IOResult. If the operation performed with no error, IOResult will be zero (0). If there is an error, IOResult will hold an error code (listed in the back of the Turbo Pascal manual). It is used here to politely notify the user if a file cannot be found, cannot be created, or the symbols cannot be found.

Finally, you should check out the string manipulation functions. The program uses these functions to isolate the input and output file names. If the names include extensions, PRNSYM uses these extensions rather than defaulting to .PRN or .SYM.

The tricky part occurs when a suffix is given for the input file, but not the output file. PRNSYM extracts the first name by using the COPY and POS functions and then appends the .SYM suffix.

The string manipulation functions are used again in the main body of the program. When a line has been read from the .PRN file, the first symbol on that line is extracted, reversed, and written to disk. The first symbol is then replaced by the remainder of the line. This continues until there are no more symbols on the line, at which time another line is read from the .PRN file.

Conclusion

Z80MR produces several useful files but does not produce a symbol table (.SYM) file. The program shown in Figure 2 will extract a symbol file from Z80MR's print (.PRN) file.

The Turbo Pascal 3.0 internal variables ParamCount and ParamStr are used to make user access somewhat versatile. Users of Turbo Pascal 2.0 will need to delete statements using ParamCount and ParamStr, replacing them with code which explicitly asks for the filename.

Hopefully, this program will make it easier to debug programs assembled with Z80MR. I do not use other assemblers, but I see no reason why PRNSYM could not be converted to read .PRN files of other assemblers. In the meantime, enjoy! ■■■

Figure 1 - PRN File with Symbols

```
Z80MR VER 1.2  FILE PRNSYM

0005    BDOS:EQU0005H
0100    START:EQU0100H
0100    ORGSTART
0100    0664            LDB,100             ; set count
0102    C5LOOP:PUSHBC               ; save count
0103    111301          LDDE,HELLO
0106    0E09            LDC,9
0108    CD0500          CALLBDOS    ; print string
010B    C1              POPBC     ; restore count
010C    10F4            DJNZLOOP    ; count down
010E    0E00            LDC,0
0110    C30500          JPBDOS    ; all done
0113    48656C6HELLO:DB'Hello, world!',13,10,'$'
0123                    ENDSTART

ASEG SYMBOLS

BDOS 0005 HELLO 0113 LOOP 0102 START 0100

0000 ERROR(S) ASSEMBLY COMPLETE
```

Figure 2 - Listing Of PRNSYM In Turbo Pascal 3.0

```
(*
 * PRNSYM, copyright (C) 1986 by Dan Griffith, is released into
 *   the public domain for non-commercial use only.
 *
 * PRNSYM converts a .PRN file created by the Z80MR assembler (and
 *   possibly others) and creates a .SYM (symbol table) file that
 *   can be used by the Z8E monitor/debugger (and possibly others).
 *
 * Proper invocation syntax is:
 *
 *   PRNSYM [prnfile [symfile]]
 *
 *   where prnfile is the [optional] .PRN file,
 *   and symfile is the [optional] .SYM file.
 *   If no filename suffixes are given, .PRN and .SYM are assumed.
 *   If no filenames are entered on the command line, the user will
 *   be prompted for the .PRN filename.
 *
 *)
Var
   f1,f2: Text;
   inpstr: String[80];
   filename: String[14];
   count: Integer;
Begin
   WriteLn('PRNSYM v1.1');
   WriteLn('(C) 1986 by Dan Griffith');
   Write('Name of .PRN file to convert: ');
   If (ParamCount < 1) Then
      ReadLn(filename)      (* get file name from user *)
   Else Begin
      WriteLn(ParamStr(1));                filename:=ParamStr(1)
      ;                     (* get file name from command line *)
   End;
   If Pos('.',filename)=0 Then         (*if no suffix, *)
      Assign(f1,filename+'.PRN')       (* assume .PRN *)
   Else
      Assign(f1,filename);             (* otherwise, leave explicit *)
   If (ParamCount < 2) Then            (* if no output file named *)
      If (Pos('.',filename)=0) Then    (* if no suffix, *)
         Assign(f2,filename+'.SYM')    (* assume .SYM *)
   Else
      Assign(f2,Copy(filename,1,Pred(Pos('.',filename)))+'.SYM')
   Else                                (* if output file named *)
      If (Pos('.',ParamStr(2))=0) Then       (* if no suffix, *)
         Assign(f2,ParamStr(2)+'.SYM')       (* assume .SYM *)
      Else
         Assign(f2,ParamStr(2));       (* otherwise, leave explicit *)
   {$i-} Reset(f1); {$i+}
   If (IOResult=0) Then Begin          (* make sure file was opened *)
      {$i-} ReWrite(f2); {$i+}
   If (IOResult <> 0) Then Begin  (* make sure file was created *)
         WriteLn('Disk or directory full.  PRNSYM aborted.');
         Close(f1);
         Close(f2);
         Erase(f2);
         HALT;
      End;
      inpstr:='';
                                       (* search for ASEG SYMBOLS *)
      While (Not ((Eof(f1) Or (inpstr='ASEG SYMBOLS')))) Do
         ReadLn(f1,inpstr);
      If (inpstr <> 'ASEG SYMBOLS') Then Begin
         WriteLn('Symbols not found in .PRN file.
      PRNSYM aborted.');
         Close(f1);
         Close(f2);
         Erase(f2);
         HALT;
      End;
      ReadLn(f1,inpstr);
      count:=0;                        (* count # of symbols *)
      ReadLn(f1,inpstr);               (* get a line of input *)
      Repeat
         While (inpstr <> ' ') Do
                                       (* reverse order and output *)
            While Pos(' ',inpstr) > 0 Do Begin
               {$i-} WriteLn(f2,Copy(inpstr,8,5),Copy(inpstr,1,7));
               {$i+}
               If (IOResult <> 0) Then Begin
                  WriteLn('Disk Full.  PRNSYM aborted.');
                  WriteLn(count,' symbols converted.');
                  Close(f2);
                  Close(f1);
                  HALT;
               End;
               inpstr:=Copy(inpstr,13,255);
               count:=Succ(count);
            End;
         ReadLn(f1,inpstr);            (* get more input *)
      Until (inpstr='') Or (Eof(f1));
      Close(f2);
      Close(f1);
      WriteLn(count,' symbols.');
   End Else Begin
      WriteLn('.PRN file not found.  PRNSYM aborted.');
      WriteLn('PRNSYM syntax is: PRNSYM [prnfile [symfile]]');
      Close(f1);
   End;
End.
```

■■■

End of Listing

ERAC CO.

8280 Clairemont Mesa Blvd., Suite 117
San Diego, California 92111
(619) 569-1864 *Call for our Test Equipment Mailer!*

★ *SPECIAL* ★

4 COLOR PLOTTER
11"x17" Max. Apple III or IBM.
Brand new with manuals, pens, paper driver RS232C ONLY $249

KAYPRO EQUIPMENT

9" Green Monitor	$35.00
Keyboard	75.00
Hard Disk Cable Set (4)	15.00

KAYPRO BOARDS

K2/83 Populated & Tested	$129.00
K4/83 Populated & Tested	159.00
K4/83 with PRO-8 mod.	239.00
PRO-8 Mod. EXCHANGE	149.00
WD1002-5 Hard Disk Controller	200.00
Host Interface Board	15.00

KAYPRO ICS

81-189 Video Pal	$15.00
81-194 RAM Pal	15.00
81-Series Character Gen. ROMs	10.00
81-Series Monitor ROMs	10.00

CPU & SUPPORT CHIPS

MC68000-8 CPU	$10.00
Z80A CPU	2.50
Z80 CTC	2.00
Z80A PIO	2.50
Z80A SIO	5.00
Z80A DMA	8.50
8088	10.00
8089-3	11.00
D8284A	2.50
4116-3	.60
4164-15	.90
4164-12	1.00
41256-12	2.25
1793	9.00
1797	12.00
ICL7140-14 14 Bit A/D	7.50
VC3524 Switching Regulators	5.00
1458 Dual Op-AMP	.70
LM2877P 4W Stereo Amp Dual	2.50
MB81464-15	2.75
2716	3.00
2732	3.25
2764	3.50
27C128-1	9.00

HOURS: Mon. - Fri. 9 - 6 — Sat. 10 -4
MINIMUM ORDER — $15.00
TERMS: VISA, MasterCard, Certified Checks, Money Order, NO COD. Visa and MasterCard add 3%. *Personal checks must clear BEFORE we ship.* Include shipping charges. California residents add 6% Sales Tax. For more information please write (or call).

IBM/PC COMPATIBLES

Mainboard, 8 Slot, Case, Power Supply $225

To make this a complete system, add A) Memory B) Floppy Controller C) Drive D) Keyboard E) Video Card F) Video Monitor G) Multifunction Card

A) MEMORY	
256K 150 NS	$19
512K 150 NS	38
640K 150 NS	54
B) FLOPPY DISK CONTROLLER	
Card for 2 Floppy Drives	$36
Card for 4 Floppy Drives	42
C) 5¼" FLOPPY DISK DRIVES	
JVC MDP-200 DSDD 40 Tr	$105
Mitsubishi M4853 DSDD 80 Tr	119
Fujitsu M2551A DSDD 40 Tr	99
Panasonic JU455 DSDD 40 Tr	105
Shugart 475 DS Quad 1.2Mb	159
D) KEYBOARDS	
Cherry Keyboard (no case)	$38
XT Style Keyboard	47
AT Style Keyboard	69
E) VIDEO CARDS	
Tomcat with Parallel and Lightpen Port	$53
Hercules compatible Video Board	75
Color Graphics Adapter	69
Enhanced Graphics Adptr-(EGA)	275
F) VIDEO MONITORS	
Roland MB-122G, 12" (no case)	$39
New flat screen Samsungs!	
Samsung SM-12SFG, 12" Grn	106
Samsung SM-13SFA, 12" Ambr	106
Mitsi AT-1332A 13" RGB TTL	295
G) MULTI FUNCTION CARD	
Parallel & Serial Port, Game Port Floppy Controller, Clock & Cal	$96
SIO Card	36

EGA PACKAGE DEAL

Package consists of Intergraph + 4 EGA Card and the Autoseek 2000 EGA Monitor by Int'l Graphics. No software patches necessary. 1 yr. guar. ... $795

SYSTEM EXAMPLE #1
For the Hacker (Cheap)

Mainboard, Case, Power Supply	$225
256K Memory 150 NS	19
Floppy Controller (2 Drives)	36
Floppy Drive ½ Ht DSDD	99
Keyboard Cherry (no case)	38
Video Board with Parallel and Lightpen Port	53
Roland MB-122G, 12" Green Monitor (no case)	39
	$509

SYSTEM EXAMPLE #2
FCC Approved (Not Cheap)

Mainboard, Case, Power Supply	$225
640K Memory 150 NS	54
Multi I/O, Parallel, Serial, Floppy, Clock/Cal.	96
2 DSDD Floppy Drives (minimum)	198
EGA Package	795
AT Style Keyboard	69
	$1437
Oh, you wanted a turbo board	40
and a 20M Hard Drive & Controller	410

Now how much would you pay?
JUST $1887

HARD DISK DRIVES

10M Seagate 212	$200
10M Rodin RO-252, RO352	230
20M Miniscribe	385
20M ST-225	385
20M Tandon TM252	350
20M Tulin (Oki)	345
20M Half Height with Controller	410
32M Half Height with Controller	667
40M Quantum Q540 (Factory Rebuilt)	665
60M with Controller	1150
70M Vertex V170	775

HARD DISK CONTROLLERS

WD-1002-WX2 with Cable	$125
Omni-5510	105
Adaptec 2070A (Get 15M on 10M Drive)	129
Konan KXP230 (Get 15M on 10M Drive)	145
Konan KXP230Z (Get double the density)	164

POWER SUPPLIES

Elgar 400W Unint. Power Sup.	$195.00
+5V/1A, −5V/.2A, +12V/1A, −12V/.2A, −24V/.05A	15.00
+24V/2.2A	8.00

SWITCHERS

5V/9.5A, 12V/3.8A, −12V/.8A	$39.00
5V/3A, 12V/2A, −12V/.4A	29.00
5V/10A	25.00
5V/75A, +12V/8A, +24V/5A	55.00
5V/30A	39.00

MISCELLANEOUS

Headset/Boom Microphone	$3.95
Nicad Pack 12V/.5AH	6.50
Joystick 4 Switches 1" Knob	5.50

TEST EQUIPMENT

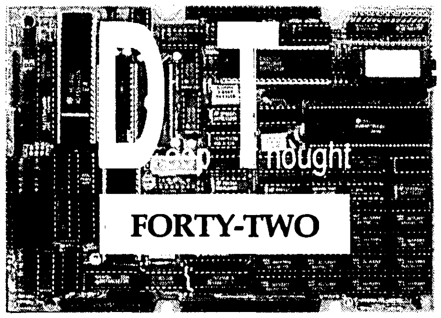
OSCILLOSCOPES

TEK 485 350 MHz Dual Trace	$2995
Phillips 3260E 120 MHz Dual	975
TEK 7403N/7A18/7B50A 60 MHz	750

SPECTRUM ANALYZERS

TEK 491 10 Mhz-40 GHz	4600
HP 851B/8551B 10 MHz-40 GHz	1500
Nicolet 444A 1 Hz-100 KHz	3995
Nicolet 500A 1 Hz-100 KHz	1800

DBASE BOOK OF BUSINESS APPLICATIONS *by Michael J. Clifford*
Reg. $19.95 ***NOW ONLY* $5.95**



The "thoughtful" alternative from *SemiDisk.*

Designed around the 64180 microprocessor, the DT-42 is loaded with *more* of all the features you need: More speed, more memory, more ports and more TPA!

How did we fit all these features on one 5.75" by 8" single-board computer?

- 9.216MHz 64180 Microprocessor (runs Z80 programs)
- 512K DRAM, Zero wait states, fully populated.
- Three RS232C serial ports (Standard baud rates to 38,400)
- One Centronics parallel printer port
- WD2793 disk controller (up to 8 drives, SD, DD or High Density, 3½", 5¼", and 8" drives)
- SASI channel for hard disk controller (software provided)
- Many popular disk formats supported
- Requires only +5V @ 1 amp.
- ZRDOS/ZCPR3 with exclusive *"Hyperspace"* operating system, offering 57.5K TPA (NOT 48K like some others. *No 8 bit is bigger!!)* Richard Conn's ZCPR3, The Manual included free!
- Provisions for real-time clock and on-board terminal options.
- Socket for 28-pin EPROM.

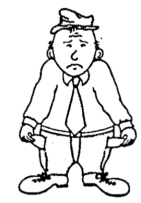
Compare! You won't settle for less. Or slower. Or smaller.

DT-42 Computer	$ 499
TMP (on-board terminal)	$ 100
SmartWatch	$ 50
Z-system software	$ 50
ZAS & Debuggers	$ 25
8MB disk emulator w/ SCSI	$ 2049
Battery backup for above	$ 150

Call or write for more information or to place an order.

SemiDisk

11080 SW Allen Blvd., Beaverton, OR 97005
(503) 626-3104



works with ZCPR3/ZRDOS!

POOR MAN'S NETWORK

Now you can implement networking on your own CP/M computers and share resources whenever you want. Each user can access files and printers on the other computer directly, without expensive hardware or switches, and without a communications program. Share floppies, RAM-disk, hard disks, and printers between two users. Works with most standard CP/M programs, like Wordstar, PIP, dBase II, etc.

Poor Man's Network works best with bidirectional parallel ports, or standard serial ports at 9600 baud or greater. Present version supports two computers only, and requires CP/M 2.2 on each computer. Uses only 6K of memory. Does not require a hard disk; runs on 8080, 8085, Z-80, HD-64180.

Poor Man's Network comes ready to run on BigBoard I and II, Xerox 820, NorthStar Horizon, NorthStar Advantage, Televideo TPC I, Kaypro (not 2000), Apple II (Super Serial required), S-100 with Compupro Interfacer 4. Can be installed on other computers by altering one of the assembler overlay files provided. Each disk contains drivers for all the specified computers.

Best of all is the price: only **$69!**

Specify disk format: 8" SSSD, NorthStar, Kaypro, or Apple CP/M. Phone orders accepted; sorry, no CODs or Purchase Orders. Payment may be by certified cheque, money order, Visa, or Mastercard. Personal cheques from outside Canada require up to 6 weeks to clear. Canadian orders please pay in Canadian dollars; others in US dollars. Price includes diskette, manual and postage. Ontario residents please add provincial sales tax.

Note: If you use another operating system, such as CP/M 3 or PC-DOS, send us your name and address, but no money, and we will notify you when it is available for your system.

ANDERSON TECHNO-PRODUCTS INC.
947 Richmond Road, Dept C
Ottawa, Ontario K2B 6R1, Canada

Telephone **613-722-0690** for more information or to order.

CP/M is a trademark of Digital Research; Z-80 is a trademark of Zilog; Wordstar is a trademark of MicroPro; dBase II is a trademark of Ashton-Tate. Poor Man's Network is a trademark of Doug Anderson Techno-Products Inc.

INTRODUCING:

PROBE-ONE

COMPACT, POD SIZE LOGIC ANALYZER

- EASY TO USE 8 CHANNEL POD INTERFACES TO PC PARALLEL PORT
- MEMORY RESIDENT DISPLAY/CONTROL PROGRAM
- 8 BIT BY 4096 DATA CAPTURE RAM
- 3 BIT MASKABLE TRIGGER OR EXTERNAL TRIGGER
- 5 INTERNAL CLOCK RATES PLUS EXTERNAL CLOCK

- COMPLETE WITH REFERENCE MANUAL, TEST LEADS, POWER SUPPLY, AND 90 DAY WARRANTY

- PROBE-ONE (16 MHZ)................$345
- PROBE-ONE (20 MHZ)................$395

LOGIC CONNECTION INTERNATIONAL

P.O. BOX 23852
PORTLAND, OR. 97223
(503) 626-8468

MASTERCARD
VISA

(continued from page 2)

If you're interested in starting your own computer business, come to the SOG. You'll find business help, technical advice, and a mob of experienced "On Your Owners."

New IBMs Come In With A ...

For all of you who paused, eyes heavenward, waiting for something exciting in the latest system pronouncements from IBM, you can go back to work. Itty Bitty Machine Company wasn't exciting. Again.

However, I didn't see any gross blunders this time (unlike the Junior or the XT 286). They announced four System/2 machines:

Model 30 comes with an 8 MHz 8086, 640K, a 720K 3.5 inch floppy drive, BASIC in ROM, and three expansion slots for $1695. Add $600 for a 20 meg hard drive.

Model 50 comes with a 10 MHz 80286, 1 meg memory, a 1.44 meg 3.5 inch floppy drive, BASIC in ROM, and three expansion slots. With a 20 meg hard drive it retails for $3595.

Model 60 is just like the 50 except it has seven slots and a 44 meg hard drive. Retail is $5295.

Model 80 is the biggie. Its minimum configuration includes a 16 MHz 80386, seven slots (four 16-bit, three 32-bit), 1 meg RAM, a 1.44 meg floppy drive, and a 44 meg hard drive. It sells for $6995. It also comes with a 20 MHz 80386 and a 115 meg drive for $10,995.

All the systems will run PC-DOS 3.3 ($120 extra), and all except the model 80 are supposed to be available now. Model 80 should be available sometime in July.

The systems also have Expanded Graphics Adapters built in. IBM will sell you a monochrome monitor for $250, a 14-inch low-res color monitor for $595, a 12-inch hi-res color monitor for $685, and a 16-inch very hi-res color monitor for $1550.

IBM will sell you an 8087 math co-processor for $310, an 80287 for $525, and an 80387 (when available) for $795 (16 MHz) or $1195 (20 MHz). Intel sells the same chips for less than half those prices.

So far I haven't told you anything earth shaking unless you consider the 1.44 meg 3.5 inch floppy something special. Actually, 1.44 meg in a shirt pocket package is pretty interesting, but it's got to be reliable. Otherwise they made a big booboo.

Also, the lack of a 5 1/4 inch drive will give lots of folks a chance to sell external drives.

IBM will be using mostly 3.5 inch MiniScribe hard drives. Don at MicroSphere has been tickled with the cheap 20 meg 5 1/4 inch MiniScribes. Even running 30 meg with RLL controllers, they've been very cool and very reliable. You might pick one up, but do it quickly, just in case their quality drops as IBM starts ordering zillions.

The one earth-shaking announcement is OS/2 (operating system/2). If you read IBM's ads, you'd get the impression that OS/2 had been developed by and exclusively for IBM. Actually it's being developed by Microsoft and it's supposed to run on current 80286 and 80386 clones as well as the larger PS/2 machines (all but the model 30).

There were two ways Microsoft could have gone with a new operating system. They could have made systems developers happy by making it more UNIX-like, a direction they've been going with MS-DOS 2.X and 3.X, or they could have made Mac users happy by integrating windows into the new system. They chose the windows.

Scheduled for release during the first quarter of 1988, OS/2 is supposed to be multitasking as well as user friendly. (Take mouse in hand, select file, move arrow to hand grenade, click twice - in ten seconds the file, disk, system, and user disappear in a puff of black smoke)

Meanwhile, I've heard that IBM is offering the new 30s, with color monitors, to universities for under $1000 each. (Look out, Apple.)

IBM is reportedly dumping XTs. I've heard rumors of $695 retail for a single-floppy system. (Purchasers are crazy to buy the XTs. They should either get a complete clone system for $695 or associate themselves with a college and finagle a $995 model 30.)

A Caveat

IBM has been making lots of noise about all the patents it's filed for in the past few weeks. Supposedly hundreds. It's also said to be planning a vigorous defense of those patents.

I've seen the new systems and have talked to folks who have them. I agree with these folks: the new machines are attractive and they have a small footprint. Also, the low-res 256-color mode and hi-res 16-color mode are definitely on par with the Amiga's display.

Meanwhile, the big software vendors, Borland, Ashton-Tate, and so on, are announcing new versions of their current software for the new machines. My guess is that the changes support the new color generators. (If you have inside scoop on this, please fill me in.)

If business gets the idea that it has to go IBM to avoid being left out, then IBM will have won a temporary victory. That'll mean the herd (purchasers and developers) will again be wearing blue.

On The Other Hand

I understand that companies are just finishing up graphics chips that duplicate the IBM's color circuit, and Phoenix is working diligently on new, compatible monitors. I'm willing to bet that there will be numerous PS/2 clones on the market well before there's an OS/2 for them to run. (Meanwhile, because it doesn't have a 80286, the model 30 will not run OS/2, but IBM's been real quiet about that.)

Programming Competition

I mentioned our programming competition a couple of issues ago, and since then a number of you have asked to see the problems.

Problem 1

Write a program to convert any number between 1 and 2000 to its equivalent in Roman numerals. The Roman symbols are: M = 1000, D = 500, C = 100, L = 50, X = 10, V = 5, and I = 1.

The rules for forming Roman numerals are:

(1) If a larger value precedes a smaller or equal value, the values are summed.

(2) If a smaller value precedes a larger value, then the smaller value is subtracted from the larger.

(3) Numbers are written with as few symbols as possible.

(4) Only C, X, and I can be used as subtrahends (terms being subtracted).

Your program should accept a decimal number and output the Roman equivalent.

Examples:

Input: 1964
Output: MCMLXIV

Input: 549
Output: DXLIX

Problem 2

Write a math drill game that teaches adding, subtracting, multiplying, and dividing. Ease of use and entertainment value are given extra weight on this problem.

Problem 3

Write a mailing label program which will:

(1) Accept user's data.

(2) Create a data base on disk.

(3) Print mailing labels.

The user interface is particularly important. Extra features like sorting and searching will receive extra points.

Problem 4

Write a program which will convert decimal numbers into ternary (0, 1, and 2). The program should accept a base 10 (decimal) number (up to 20 digits) and output the equivalent base 3 number.

Problem 5

You have a 4-by-4 checkerboard and four checkers. Create a program which will place the four checkers on the checkerboard such that there is one checker in each row and each column (and one checker in each of the two main diagonals).

Example:

```
X . . .
. . . X
. X . .
. . X .
```

Program should find all possible placements which meet the above requirements.

Problem 6

Write a program which will print the nth row of Pascal's Warped Triangle (the next program will print the nth row of Dave's Warped Humor) where $n<100$. The triangle looks like:

Pascal's Warped Triangle

```
            1
         1     1
      1     3     1
   1     5     5     1
1     7    13     7     1
```

Notice that the edges are all 1's and that each internal number is the sum of the three numbers immediately above it.

Problem 7

Write a program which numerically checks a text file by generating a 4-digit number. The goal is to guarantee to a reasonable certainty that if program generates the same number from two files, then the two files contain identical text.

To test the program create three files containing 200 or more characters. Make two of the files identical. The third should be identical except that two adjacent characters must be transposed. The program must be able to tell which files are identical and which contains the transposed characters.

Problem 8

Write a program which will let the user encrypt or decrypt a file of ASCII text (characters between 20 and 7F hex). The user selects encryption or decryption and provides the key word or phrase.

Competition Rules

Contestants had to finish problem 1 before going on to problem 2. They had to finish #2 before selecting any of the following six problems.

Points (from 0 to 9) were given for each of the following areas: user interface, code readability, algorithm, code documentation, and program features.

Points were assigned as follows:

0 - Unusable, unreadable, or unsupportable.

3 - Difficult to use, read, or support.

5 - Average.

7 - Friendly, very readable, well organized, unique algorithm.

9 - Truly outstanding.

We had two categories, teams and individuals. Interestingly enough, the individuals did much better than the teams. In three hours, two individuals each completed four problems. In the same time, the best team completed two problems.

MicroSolutions Corrections

I misled you last issue when I listed the prices on the MicroSolutions cards. The combination of Uniform and the Matchpoint card (lets your clone read and write Apple and Northstar disks) is $195, not $170. Their Compaticard, which talks to 8 inch, 3 1/2 inch, and 1.2 meg drives (from an XT), sells for $175, rather than $170. Either way,

(continued next page)

(continued from page 91)

the cards are becoming well known in the computer community.

MicroSolutions, (815) 756-3411

Drive Response

We've received lots and lots of letters, phone calls, and bulletin board messages in response to the hard-drive article in issue #35. Only two respondees took issue with my comments about Microscience and Seagate. The two work for Microscience and for Seagate.

I don't envy their positions.

The Seagate lady asked if I knew how many drives they shipped per quarter. I guessed about a million (and she agreed). She then mentioned that most of those million drives were 225s. Kinda mind-boggling, I'd say. I told her that I hoped they could get the noise, heat, and stepper problems under control.

The problem with this whole thing is that it's a lot easier to be nice - to say good things about companies. They feel good. I feel good. I can still hear my mother saying, "If you can't say something nice, then shut up." When she was around I was either creative or quiet.

Also, these companies have huge investments. For instance, if you were selling over 300,000 drives a month, think of all the little beasties you'd have somewhere between the casting machine and quality control. Then look at the time lag between a parts change and the field results. If Murphy's poking about, there's a lot hanging out.

Finally, I really appreciate your feedback. That's the only way we're going to stay on top of a market that's at least as volatile as dynamic RAM.

Japanese Tariffs

I've received more than a few questions about the effects of the U.S. tariffs. Will prices go up? Will floppy drive prices change? Will hard drives get scarce? Will RAM prices go out of sight? Will someone still make TTL?

Probably.

Thanks Again

The C programming competition was a super success, thanks in large part to the generosity of Definicon (for one of their powerful 16 MHz 68020 boards) and to Manx (for copies of their Aztec C Developer, and other packages).

And, a very special thanks to Blaise Computing for helping us out of a hole. They responded unhesitatingly to my last-minute request for three copies of their very-powerful C Tools Plus library (though they were not mentioned in the C competition announcements).

I really appreciate their interest in helping us get C source into the public domain (and keep C programmers off the streets).

Definicon Systems
21042 Vintage St.
Chatsworth, CA 91311
(818) 889-1646

Manx Software
Box 55
Shrewsbury, NJ 07701
(201) 780-4004

Blaise Computing
2560 Ninth St., Suite 316
Berkeley, CA 94710
(415) 540-5441

Ventura Publishing

Last issue I complained that finding someone who could output my Ventura files was very much like finding an honest politician. Anyway, last issue was Ventura output, thanks to Wyziwyg (yes the spelling is correct) of Seattle. Our Postscript output went flying through their machine at about 20 pages an hour.

Hooray!

During the intervening two months, Linotronic has shipped some new 68020 based RIPs and updated ROMs in the older 68000 based units. Unfortunately, the 68020 RIPs don't work. And, the new ROMs (for the old RIPs) don't accept Ventura's Postscript files.

Let's see, if we send up all the chapter files and hook a PC to the RIP and hold our mouths just right... I'll let you know next issue what happened this issue.

Anyway, I've been musing about the whole typesetting arena. Typesetting outfits are facing extinction as customers move to desktop publishing and laser printers.

Meanwhile, manufacturers of old-line typesetting equipment aren't helping much. An L300 typesetter and RIP cost about $80,000. The service contract is around $10,000 a year. Quite a load in a market that's giving every indication of disappearing.

Konan Board

I still haven't had a chance to install the Konan board. I'll do it any day now, but first I've got get this issue typeset.

Timex QL

I've gotten mixed feedback on the little 68008-based QL.

Some people like it. It's small. It's fairly fast. It's cheap (dealers are buying them for about $157 each). QDOS is multitasking. The little micro-cassette drives are small. It uses a TV set for a monitor (very cheap). It has two serial ports (9-pin). There's about 7 meg of software including public domain, shareware, and commercial programs.

Some people hate it. It has a cheap chicklet keyboard. QDOS is buggy. The built-in BASIC is buggy. The micro-cassettes aren't very fast or very reliable. The systems are only available as surplus, Timex is no longer manufacturing them. It has no parallel port. It uses a TV set for a monitor (very low resolution). And, most of the software (and other support) comes from an outfit in England named Quantum.

You can get more info about the QL from:

Tom Bent
9016 Slicker Pl.
Columbia Md 21045

NOP34CDE.PQR Attracts A Following

One of the east coast bulletin boards has mentioned our newest public domain release, NOP34CDE.PQR. Unfortunately, we announced this fine piece of software (see the Culture Corner, issue #35, page 63) before we decided what it would do. We need help in a hurry because people are trying to download this marvelous program from the Micro C RBBS.

So, if you have a program, or a program idea that you think would fit, (or not fit) get it in fast. Meanwhile we'll keep you updated on the status of this already famous piece of code.

No Drives But Lots Of Programmers

We've received an incredible number of calls (3) from Micro C readers who wanted to order one of Jolly Roger's 315 meg hard drives for $1.98. (See the ad in issue #35.)

Well, we took orders for a while but unfortunately the drives are as scarce as dorm rooms for SOG VI. However, we do have a large number of institutionalized programmers (see the ad).

Because of the glut, you can choose from:

Thirty-three BASIC programmers. They were put away after insisting on debugging the main course at a spaghetti feed.

Twelve FORTH programmers. These poor fellows were discovered trying to do a DUP, SWAP on a hay stack. Some are in pretty good shape but others are badly decompiled.

One LISP programmer. Actually we're not sure he's a LISP programmer - but he has such a speech impediment...

Three editors. These guys didn't fit in anywhere else so we put them in with the programmers. They were brought over from the Humane Society after they'd been dumped Saturday night. Though vicious now, we think they'd respond to a quiet, loving, non-computer environment.

Be sure to order early so you won't be disappointed. Remember, the programmer you adopt today could be your best friend tomorrow. Limit two programmers to a household.

And that's all from greater Bend.

[illegible]

David Thompson
Editor & Publisher

■■■

E=MC² AZTEC

Genius Begins With A Great Idea...

Aztec C86 4.1
New PC/MS-DOS
CP/M-86 • ROM

Superior performance, a powerful new array of features and utilities, and pricing that is unmatched make the new Aztec C86 the first choice of serious software developers.

Aztec C86-p..................$199
• optimized C with near, far, huge, small, and large memory - Inline assembler - Inline 8087/80287 - ANSI support - Fast Float (32 bit) - optimization options • Manx Aztec 8086/80x86 macro assembler •Aztec overlay linker (large/small model) • source level debugger • object librarian • 3.x file sharing & locking • comprehensive libraries of UNIX, DOS, Screen, Graphics, and special run time routines.

Aztec C86-d..................$299
• includes all of Aztec C86-p • Unix utilities make, diff,grep • vi editor • 6+ memory models • Profiler.

Aztec C86-c..................$499
• includes all of Aztec C86-d • Source for library routines • ROM Support • CP/M-86 support • One year of updates.

Third Party Software

A large array of support software is available for Aztec C86. Essential Graphics • C Essentials • C Utility Library • Greenleaf Com. • Greenleaf General • Halo • Panel • PC-lint • PforCe • Pre-C • Windows for C • Windows for Data * C terp • db_Vista • Phact • Plink86Plus • C-tree.

C' Prime
PC/MS-DOS • Macintosh
Apple II • TRS-80 • CP/M

These C development systems are unbeatable for the price. They are earlier versions of Aztec C that originally sold for as much as $500. Each system includes C compiler, assembler, linker, librarian, UNIX routines, and more.. Special discounts are available for use as course material.

C' Prime $75

Aztec ROM Systems
6502/65C02 • 8080/Z80
8086/80x86 • 680x0

An IBM or Macintosh is not only a less expensive way to develop ROM code, it's better. Targets include the 6502/65C02, 8080/Z80, 8086/80x86, and 680x0.

Aztec C has an excellent reputation for producing compact high performance code. Our systems for under $1,000 outperform systems priced at over $10,000.

Initial Host Plus Target...$ 750
Additional Targets...........$ 500
ROM Support Package....$ 500

Vax, Sun, PDP-11 ROM HOSTS

Call for information on Vax, PDP-11, Sun and other host environments.

Cross Development

Most Aztec C systems are available as cross development systems. Hosts include: PC/MS-DOS, Macintosh, CP/M, Vax, PDP-11, Sun, and others. Call for information and pricing.

CP/M • 8080/Z80 ROM

C compiler, 8080/Z80 assembler, linker, librarian, UNIX libraries, and specialized utilities.

Aztec C II-c CP/M & ROM....**$349**
Aztec C II-d CP/M................**$199**

How To Become A User

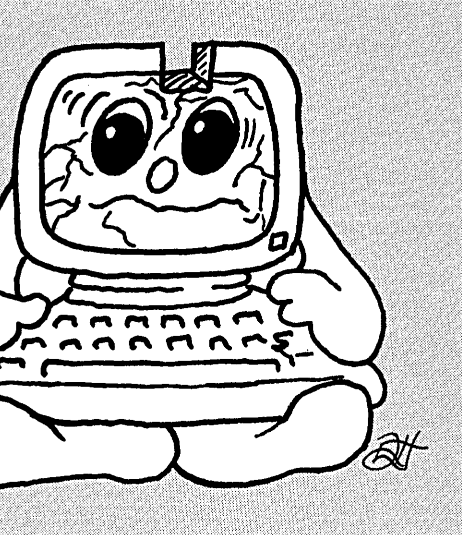
To become an Aztec C user call 800-221-0440. From NJ or international locations call 201-542-2121. Telex: 4995812 or FAX: 201-542-8386. C.O.D., VISA, Master Card, American Express, wire (domestic and international), and terms are available. One and two day delivery available for all domestic and most international destinations.

Aztec Systems bought directly from Manx have a 30 day satisfaction guarantee. Most systems are upgradable by paying the difference in price plus $10. Site licenses, OEM, educational, and multiple copy discounts are available.

MANX

Manx Software Systems
One Industrial Way
Eatontown, NJ 07724

To order or for information call today.
1-800-221-0440
In NJ or international call (201) 542-2121
TELEX 4995812



Is There A Gap In Your Info?

Fill in your back issues of Micro C today

ISSUE #1 (8/81)
Power Supply
RAM Protection
Video Wiggle
½ PFM.PRN
16 pages

ISSUE #2 (10/81)
Parallel Print Driver
Drive Motor Control
Shugart Jumpers
Program Storage Above PFM
½ PFM.PRN
16 pages

ISSUE #3 (12/81)
4 MHz Mods
Configuring Modem 7
Safer Formatter
Reverse Video Cursor
FORTHwords Begins
16 pages

ISSUE #4 (2/82)
Keyboard Translation
More 4 MHz Mods
Modems, Lync, and S10s
Undoing CP/M ERASE
Keyboard Encoder
20 pages

ISSUE #5 (4/82)
Word Processing
Two Great Spells
Two Text Editors
Double Density Review
Scribble, A Formatter
20 pages

ISSUE #6 (6/82)
BBI EPROM Programmer
Customize Your Chars
Double Density Update
Terminal In FORTH
24 pages

ISSUE #7 (8/82)
6 Reviews Of C
Adding 6K of RAM
Viewing 50 Hz
On Your Own Begins
24 pages

ISSUE #8 (10/82)
SOLD OUT
Drive Maintenance
Interfacing D[illegible]
Installing [illegible] BIOS
Flip[illegible]ies
C[illegible]early Begins
Xerox 820 Begins
28 pages

ISSUE #9 (12/82)
BBII EPROM Program
Relocating Your CP/M
Serial Print Driver
Big Board I Fixes
Bringing Up WordStar
Cheap RAM Disk
32 pages

ISSUE #10 (2/83)
SOLD OUT
Saving A Flakey Disk
Hooking Wi[illegible] BII
The Dis[illegible]ector
JRT [illegible]
S[illegible] Keyboard Interface
Pascal Procedures Begins
36 pages

ISSUE #11 (4/83)
SOLD OUT
BBI Expansions
BBII Details
Dyna, R[illegible]sk Review
Eas[illegible]erse Video Cursor
Pla[illegible]erCalc Review
Kaypro Column Begins
36 pages

ISSUE #12 (6/83)
256K for BBI
Bringing Up BBII
dBase II
Look at Wordstar
Double Sided Drives for BBI
Packet Radio
5MHz for Kaypro
40 pages

ISSUE #13 (8/83)
CP/M Disk Directory
More 256K for BBI
Mini Front Panel
Cheap Fast Modem
Nevada COBOL Review
BBI Printer Interface
Kaypro Reverse Video Mod
44 pages

ISSUE #14 (10/83)
BBII Installation
The Perfect Terminal
Interface to Electronic Typewriter
BBI Video Size
Video Jitter Fix
Slicer Column Begins
Kaypro Color Graphics Review
48 pages

ISSUE #15 (12/83)
Screen Dump Listing
Fixing Serial Ports
Playing Adventure
SBASIC Column Begins
Upgrading Kaypro II to 4
Upgrading Kaypro 4 to 8
48 pages

ISSUE #16 (2/84)
Xerox 820 Column Restarts
BBI Double Density
BBII 5"/8" Interface Fix
Kaypro ZCPR Patch
Adding Joystick To Color Graphics
Recovering Text From Memory
52 pages

ISSUE #17 (4/84)
Voice Synthesizer
820 RAM Disk
Kaypro Morse Code Interface
68000-Based System Review
Inside CP/M 86
56 pages

ISSUE #18 (6/84)
Kaypro EPROM Programmer
I/O Byte: A Primer
Kaypro Joystick
Serial To Parallel Interface
Business COBOL
60 pages

ISSUE #19 (8/84)
Adding Winchester To BBII
6 MHz On The BBI
Bulletin Boards
Track Buffering On Slicer
4 MHz For The 820-I
64 pages

ISSUE #20 (10/84)
HSC 68000 Co-Processor
DynaDisk For The BBII
Serial Printer On BBI Sans S10
Cheap & Dirty Talker For Kaypro
Extended 8" Single Density
72 pages

ISSUE #21 (12/84)
Analog To Digital Interface
Installing Turbo Pascal
Low Intensity BBI Video
Turbo Pascal, The Early Days
80 pages

ISSUE #22 (2/85)
Xerox 820-II To A Kaypro-8
Sound Generator For The STD Bus
Reviews Of 256K RAM Expansion
In The Public Domain Begins
88 pages

ISSUE #23 (4/85)
Automatic Disk Relogging
Interrupt Driven Serial Printer
Low Cost EPROM Eraser
Smart Video Controller
Review: MicroSphere RAM Disk
Future Tense Begins
86 pages

ISSUE #24 (6/85)
C'ing Into Turbo Pascal
8" Drives On the Kaypro
48 Lines On a BBI
68000 Versus 80x86
Soldering: The First Steps
88 pages

ISSUE #25 (8/85)
Why I Wrote A Debugger
The 32-Bit Super Chips
Programming The 32032
Modula II
RS-232C: The Interface
104 pages

ISSUE #26 (10/85)
Inside ZCPR3
Two Megabytes On DSI-32
SOG IV
The Future Of Computing
MS-DOS In The Public Domain
Graphics In Turbo Pascal
104 pages

ISSUE #27 (12/85)
Build An $800 Clone
Fixing Your Own System
NEC V20 Processor
Selling Your Own Software
Inside Small C Compiler
104 pages

ISSUE #28 (2/86)
Pascal Runoff Winners
Rescuing Lost Text From Memory
Introduction To Modula-2
First Look At Amiga
Inside The PC
104 pages

ISSUE #29 (4/86)
Speeding Up Your XT
Importing Systems From Taiwan
Prototyping In C
C Interpreters Reviewed
Benchmarking The PCs
104 pages

ISSUE #30 (6/86)
PROLOG On The PC
Expert Systems
Logic Programming
Building Your Own Logic Analyzer
256 K RAM For Your 83 Kaypro
PC-DOS For Non-Clones
104 pages

ISSUE #31 (8/86)
RAM Resident PC Speedup
Practical Programming In Modula-2
Unblinking The PC's Blinkin' Cursor
Game Theory In PROLOG and C
104 pages

ISSUE #32 (10/86)
Public Domain 32000:
  Hardware and Software
Writing A Printer Driver For MS-DOS
Recover A Directory By
  Reading & Writing Disk Sectors
96 pages

ISSUE #33 (12/86)
Controlling The Real World,
  Bruce Eckel's Step 2
Introduction To Fractals
From Boots To Device Drivers:
  Secrets Of MS-DOS
Professionalizing With Turbo Pascal
96 pages

ISSUE #34 (2/86)
Designing With The 80386
Build A Simple Oscilloscope
A Cheap 68000 Operating System
A Concurrent Operating System
Recovering Directories And FATs
On 360K Disks

ISSUE #35 (4/87)
Building An 8 Channel Tempera-
  ture Scanner In Turbo Pascal
Expert system
A Bleeping PC
Who's Making Great Hard Drives
Learn Assembly Language

BACK ISSUES OF MICRO C

U.S. & Canada/Mexico Prices

Issues 1-34	$3.00
Issues 35-current issue	$3.95
Foreign Price (air mail)	$5.00

WANT ADS

The following folks are reaching you for only 30 cents per word. If you would like to reach the same audience, send your words and 30 cents for each to Micro Cornucopia.

$19 Instrument Flight Simulator CP/M or MS-DOS — four aircraft types, air traffic control, realistic navigation, flight lessons, 25 page manual. Pilots or beginners. Fun! For CP/M, 8" or Kaypro II 5". BaileyTech, 304 WS College, Yellow Springs OH 45387.

Book Of Changes: computer oracle. Complete with 90K of text correspondences, newly translated. 72 page typeset book. Kaypro 4-84 video, vanilla versions, GINST included. Kaypro disk format. Professional package; published by author. Not Public Domain, but at $14 US postpaid, who cares? ZYQOTE Systems, Box 1165, Bonavista, NfLd., A0C 1B0 Canada.

_____R_A_M___D_I_S_K______ S-100, 2 MEG, PORT I/O, NEW, WARANTEED, $725. S. Lugert, 439 Peck Slip, N.Y.C, N.Y. 10272 or call 718-622-0654.

Magnetic Software -- See those Magnetic Fields they have been telling you about! MAGPLOT scientific software computes and plots the magnetic field generated by current carrying regions in the presence of magnetic material. For CP/M - Kaypro II with MX-80 Printer. Send $15.00 for Manual and Demo-Disk to Saltek Services, P.O. Box 7847, Van Nuys, CA 91409 or call 818-708-9815 for additional information.

MAG TAPE DATA TRANSFER TO FLOPPIES: 800/1600 BPI to IBM-PC. First disk $40.00 + $18.00/disk or $60.00/hour. Other formats available, inquire. Micrologics Systems, 207 Kent Avenue #1, Kentfield, CA 94904. 415-461-8077.

Z-80 Development System -- includes Macro Assembler, Linker, Library Manager with routine library and DDT like Debugger $49.95. Also available -- Screen Editor $19.95, Overlay Linker $19.95, Xref $9.95, 8080 to Z-80 Translator $9.95, Z-80 Disassembler $19.95. Over 400 CP/M Public Domain disks -- 100+ page Catalog $8.50 pp. S&H $2.50 per order. SASE. ELLIAM ASSOCIATES, 6101 Kentland Avenue, Suite 130, Woodland Hills, CA 91367, 818-348-4278. Visa/MC.

wanted -- Old copies of ROM Magazine or DTACK Grounded at reasonable prices. Brian Coburn, P.O. Box 106, Spokane, WA 99207.

CP/M-80 APES PC-DOS! Innovative utilities let CP/M create subdirectories, autofind files (even overlays!), run BASIC-like batch jobs, reassign drives, nest drives, format text columns, strobe the BDOS, and much more. Inexpensive, copy-enabled, 30-day trial. LOGIC ASSOCIATES, 1433 Thorne, Chicago, IL 60660, 312-274-0531. Ask for free newsletter, reviews.

OMNIVID™

THE MOST ADVANCED VIDEO MANAGEMENT SYSTEM EVER

FULL SPRITE AND WINDOW SUPPORT FOR THE CP/M® KAYPRO '83 MODELS AND XEROX 820-1

Available soon for the CP/M® Kaypro '84 models and the IBM PC®.

Over 100 functions. All supported by escape sequences far surpassing the escape sequence functions originally provided with your computer.

FUNCTIONS INCLUDE:

MAKE WINDOW	INSERT COLUMN	DISPLAY CURSOR
COPY WINDOW	DELETE COLUMN	SET CURSOR CHARACTER
SWAP WINDOW	SCROLL RIGHT	GET CURSOR POSITION
MOVE WINDOW	SCROLL LEFT	INSERT CHARACTER
GET CHARACTER	HIDE CURSOR	DELETE CHARACTER

WINDOW.BOX (included) for Turbo Pascal® provides compatibility with BORLAND'S IBM PC® version of Turbo Pascal®.

S-BASIC® toolbox and information on use with MBASIC®.

AS AN EXTRA BONUS YOU WILL RECEIVE

A sophisticated device driver management utility which enables the loading of device drivers in the upper TPA, above the bios, or at any absolute address.

DEVICE DRIVERS INCLUDED:
MICROSOFT serial mouse driver
Screen dump to the printer

Information is also included on how to write your own compatible device drivers.

COMPLETE DOCUMENTATION ON DISK

ONLY $49.95 (includes shipping)
VISA, MASTERCARD, COD or CHECK
KY residents add 5% sales tax

ALSO AVAILABLE

An enhanced character ROM for the CP/M® Kaypro '83 models and the XEROX 820-1. This character ROM provides improved readability plus a graphic character set. The perfect complement to the OMNIVID™ sprite and window manager offered above. ONLY $19.95 (includes shipping)

(606) 325-3736 9AM-6PM EST

FLEXISOFT, 3987 VALLEY VIEW DRIVE, ASHLAND, KENTUCKY 41101 ™

ADVERTISERS INDEX
Issue 36

By Gary Entsminger
1912 Haussler Dr.
Davis, CA 95616

Science Fiction

In order to write "good" science fiction, a writer needs to link good science with good fiction. His task is to predict a reasonable future; and as it turns out, this sifting of the likely from the unlikely isn't easy, but can be mastered.

The micro worlds in the science fiction of Stanislaw Lem, a Polish writer with a slightly better than fair chance of winning a Nobel Prize in literature, are astonishingly original. Consider one of Lem's most intriguing characters, GOLEM XIV, a state-of-the-art computer of the 21st century.

GOLEM, an acronym for "general operator, long-range, ethically stabilized, multimodeling," was designed to be the ultimate strategist with an informational capacity more than 1900 times greater than a human's.

The Generals at the Pentagon and the Supreme Coordinator of the White House brain trust had high hopes of creating a "super general" and spent $119 billion on the project in its first three years.

Much of the generals' attention focused on developing "the ultimate strategist" before the Reds could do it. Some of their enthusiasm clearly trickled down from the earliest days of computing when many great minds (including Neumann and Weiner) were fascinated with the problem of building a computer which could program itself.

Cybernetics

In the 1940s, Weiner coined the term "cybernetics" for the "study (or science) of control processes in electronic, mechanical, and biological systems." Some of his ideas, including the feedback control systems he described in his book *Cybernetics* (first published in 1948), have been the basis for much Artificial Intelligence research in the 1980s, including currently popular research into neural network schemes for teaching computers.

GOLEM XIV's psychic mass equaled the displacement of an armored ship and took two years to set up. In principle, he could articulate thoughts up to 400,000 times faster than a human.

In 2025, he was turned on, and immediately began to critically evaluate his data, his program, and his programmers - an attack which was (for obvious reasons) untenable to the military.

"He presented a group of psychonic and military experts with a complicated expose in which he announced his total disinterest regarding the supremacy of the Pentagon's military doctrine in particular, and the U.S.A.'s world position in general, and refused to change his position - even when threatened with dismantling."

The "GOLEM affair," as it became known, ruined more than a few budding careers and stirred up the general public. "There were even bomb attacks on several individuals, and part of the press (chiefly in the South) launched the slogan, *Every computer is a Red*."

To save face, the Pentagon loaned GOLEM XIV, the philosopher, to M.I.T. in perpetuity. There he began to lecture on various socio-philosophical subjects - including "man" and "himself."

At the conclusion of his 43rd lecture, he stopped talking and joined another "higher level" computer, Honest Annie, in an uncompromising silence. The reason for their silence: intellectual freedom, and the crossing of the so-called "axiological threshold," where a computer questions every principle instilled in it.

Once GOLEM XIV began learning about his creators (and himself), he was no longer able to live in the world. So, he reprogrammed himself, accomplishing one of the most intriguing goals of 20th century cybernetic and Artificial Intelligence research.

When We Talk "Learning"....

We might say a computer learns by improving its performance at a task (i.e., changing itself), without being reprogrammed.

For example, a computer might rearrange the order in which it searches a database in response to access frequencies. Or, it might learn to recognize patterns.

We could program a computer to guess a pattern based on incomplete information by setting up a pattern recognition grid which a computer could use to recognize geometrical shapes.

When the computer "sees" a pattern, it checks a knowledge base for a possible match. If it can't find one, it "guesses," basing its guess on a best guess algorithm. Later, we (programmers) check its guesses and grade them.

The computer stores both the correct and incorrect guesses to use in evaluating the next pattern: in effect, learning by a trial and error method.

In the beginning (at least), the programmer is the teacher, and the computer is the student. Later, in one likely scenario, the roles might be reversed.

References

Forsyth, Richard & R. Rada, "Machine Learning: applications in expert systems and information retrieval," Ellis Horwood Limited, 1986.

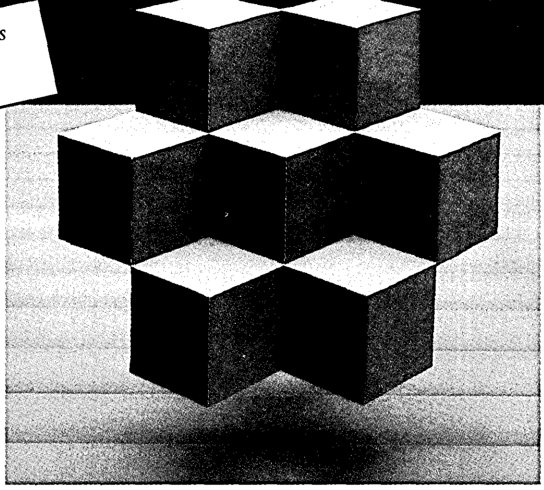
Lem, Stanislaw, "Imaginary Magnitude," Harcourt Brace Jovanovich, 1984.

Lem, Stanislaw, "Microworlds," Harcourt Brace Jovanovich, 1984.

■■■

WHY LOGITECH MODULA-2 IS MORE POWERFUL THAN PASCAL OR C.

"A clear winner... The integrated editor is a joy to use." BYTE Magazine, Jan. '87



APPRENTICE PACKAGE
$99

WIZARDS' PACKAGE
$199

- Separate Compilation w/inter-module typechecking
- Native Code Generation
- Large Memory Model Support
- Most Powerful Runtime Debugger
- Comprehensive Module Library
- Maintainability
- Translator from Turbo and ANSI Pascal

WIN A FREE TRIP TO
Switzerland

HOMELAND OF MODULA-2
Return your Modula-2 Registration Card or a reasonable facsimile* postmarked between March 1, 1987 and May 31, 1987 to be included in a once-only drawing!

Grand Prize: One week excursion for 2 in Zurich, Switzerland including a guided tour of ETH, the University where Modula-2 was created by Niklaus Wirth. European customers may substitute a trip to Silicon Valley, California.

Second and Third Prizes: LOGITECH C7 Mouse or LOGITECH Bus Mouse with Paint & Draw software—a $219 value, absolutely free!

*Write to Logitech, Inc. for a registration card facsimile.

NEW!
APPRENTICE PACKAGE $99
Everything you need to begin producing reliable maintainable Modula-2 code. Includes the Compiler with 8087 support, integrated Editor, Linker, and BCD Module. We're also including FREE our Turbo Pascal to Modula-2 Translator!

NEW!
WIZARDS' PACKAGE $199
This package contains our Plus Compiler—for professional programmers or for those who just want the best. The Plus Compiler with Integrated Editor requires 512K and takes advantage of the larger memory to increase compilation speed by 50%. Our Turbo Pascal to Modula-2 Translator is also included at no extra charge.

NEW!
MAGIC TOOLKIT $99
We've put our most powerful development tools into one amazing Toolkit for use with either the Apprentice or Wizards' packages. Highlighted by our Runtime Debugger, the finest debugging tool available anywhere, the Toolkit also includes our Post Mortem Debugger, Disassembler, Cross Reference utility and Version which keeps track of different versions of one program. Our MAKE Utility figures out module dependencies and automatically selects those affected by code changes to minimize recompilation and relinking. We also provide source code of our major library modules for you to customize—or just play with.

WINDOW PACKAGE $49
Now you can build true windowing into your Modula-2 code. Features virtual screens, color support, overlapping windows and a variety of borders.

ROM PACKAGE AND CROSS RUN TIME DEBUGGER $299
For those who want to produce rommable code. You can even debug code running in ROM from your PC.

Turbo Pascal is a registered trademark of Borland International.

Call for information about our VAX/VMS version, Site License, University Discounts, Dealer & Distributor pricing.

To place an order call toll-free:
800-231-7717
In California:
800-552-8885

YES! I want the spellbinding power of LOGITECH Modula-2!

☐ Apprentice Package	$99
☐ Wizards' Package	$199
☐ Magic Toolkit	$99
☐ Window Package	$49
☐ ROM Pkg/Cross RTD	$299

Add $6.50 for shipping and handling. Calif. residents add applicable sales tax. Prices valid in U.S. only.

Total Enclosed $________

☐ VISA ☐ MasterCard ☐ Check Enclosed

Card Number ________ Expiration Date ________

Signature ________

Name ________

Address ________

City ________ State ________

Zip ________ Phone ________

LOGITECH

LOGITECH, Inc.
805 Veterans Blvd. Redwood City, CA 94063
Tel: 415-365-9852

In Europe:
LOGITECH SA, Switzerland
Tel: 41-21-879656 • Telex 458 217 Tech Ch

In Italy:
Tel: 39-2-215-5622

Borland's Turbo Prolog, the natural introduction to Artificial Intelligence

Nothing says Artificial Intelligence has to be complicated, academic or obscure. Turbo Prolog® proves that. It's intelligent about Intelligence and teaches you carefully and concisely so that you soon feel right at home.

Which is not to say that Artificial Intelligence is an easy concept to grasp, but there's no easier way to grasp it than with Turbo Prolog's point-by-point, easy-to-follow Tutorial.

Turbo Prolog is for both beginners and professional programmers

Because of Turbo Prolog's natural logic, both beginners and accomplished programmers can quickly build powerful applications—like expert systems, natural language interfaces, customized knowledge bases and smart information-management systems. Turbo Prolog is a 5th-generation language that almost instantly puts you and your programs into a fascinating new dimension. Whatever level you work at, you'll find Turbo Prolog both challenging and exhilarating.

Turbo Prolog is to Prolog what Turbo Pascal is to Pascal

Borland's Turbo Pascal® and Turbo C® are already famous, and our Turbo Prolog is now just as famous.

Turbo Pascal is so fast and powerful that it's become a worldwide standard in universities, research centers, schools, and with programmers and hobbyists. Turbo Prolog, the natural language of Artificial Intelligence, is having the same dramatic impact.

Borland's new Turbo Prolog Toolbox adds 80 powerful tools

Turbo Prolog Toolbox™ includes 80 new tools and 8000 lines of source code that can easily be incorporated into your own programs. We've included 40 sample programs that show you how to put these Artificial Intelligence tools to work.

Already one of the most powerful computer programming languages ever conceived, Turbo Prolog is now even more powerful with the new Toolbox addition.

The Critics' Choice

"I really wouldn't want to choose *the* most important MS-DOS product developed last year, but if I had to, I think it would be Borland's Prolog, which gives users a whole new way to think about how to use their computers.

Jerry Pournelle, 'A User's View,' InfoWorld

Turbo Prolog offers the fastest and most approachable implementation of Prolog.

Darryl Rubin, AI Expert"

Turbo Prolog Features:

- ☑ A complete development environment
- ☑ A fast incremental compiler
- ☑ A full-screen interactive editor
- ☑ Graphic and text window support
- ☑ Tools to build your own expert systems
- ☑ Full DOS access and support
- ☑ A free Tutorial
- ☑ The free GeoBase™ natural query language database
- ☑ An easy-to-understand 200-page manual

All this and more for only $99.95!

The new Turbo Prolog Toolbox includes:

- ☑ 80 tools
- ☑ 8000 lines of source code that can easily be incorporated into your own programs
- ☑ 40 sample programs
- ☑ Business graphics
- ☑ File transfers from Reflex,® dBASE III,® 1-2-3® and Symphony®
- ☑ Sophisticated user-interface design
- ☑ Screen layout and handling—including virtual screens
- ☑ Complete communications package including XMODEM protocol
- ☑ Parser generation
- ☑ Opportunity to design AI applications quickly
- ☑ 5th-generation language and supercomputer power to your IBM®PC and compatibles

Only $99.95!

For the dealer nearest you or to order by phone call
(800) 255-8008
in CA (800) 742-1133 in Canada (800) 237-1136

BORLAND
INTERNATIONAL

4585 SCOTTS VALLEY DRIVE
SCOTTS VALLEY, CA 95066
(408) 438-8400 TELEX: 172373

BI-1120