No. 37 Sept./Oct. 1987 $3.95

THE MICRO TECHNICAL JOURNAL

MICRO CORNUCOPIA

Desktop Publishing On A PC

page 6

Build [illegible] Hi-Res Graph[illegible]r For $6.00

Desig[illegible]abase Part 3

Sandy Bra[illegible]her popular database [illegible] up some very close relat[illegible]

Controlling AC Power From Your PC

page 18

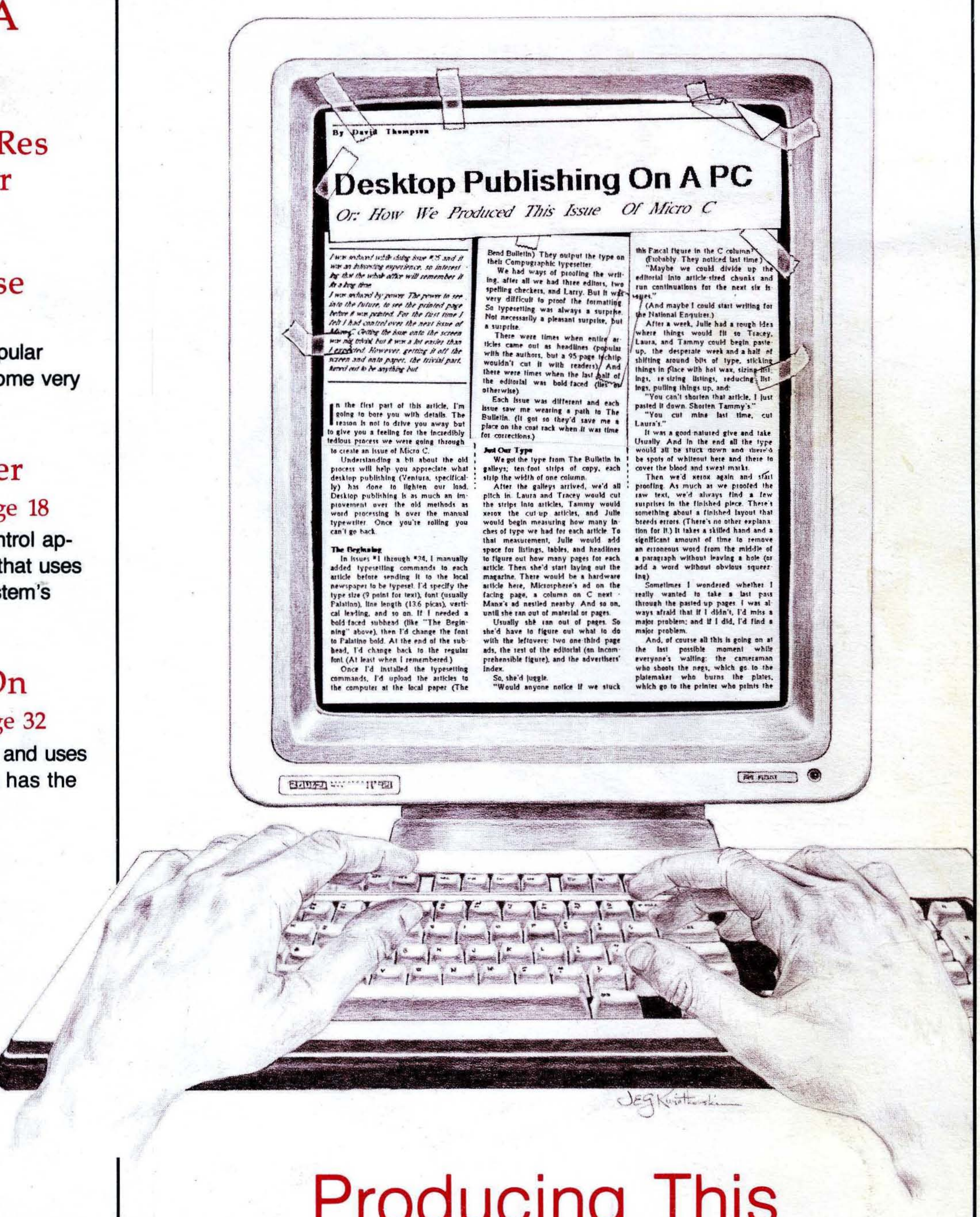
Bruce Eckel shows you how to control appliances, motors, or anything else that uses 110 VAC. (And do it from your system's parallel printer port.)

Expanded Memory On The PC/XT/AT

page 32

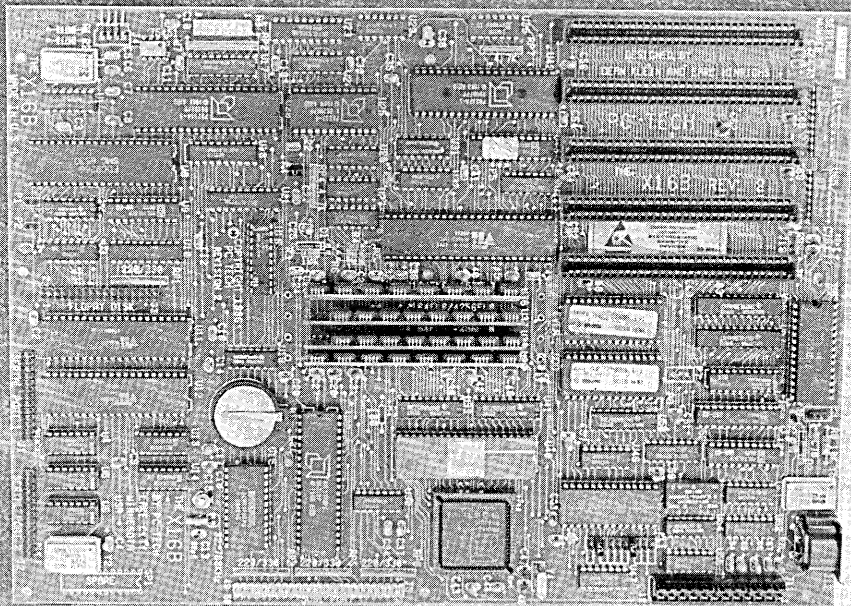
Want to write software that detects and uses expanded memory? Earl Hinrichs has the scoop.

And much, much more.



Producing This Issue

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.

NEW

16 MEGGER!

Expanded or Extended Memory for PC/XT/AT. Up to 16 Megabytes total. Expanded memory works as on FOUR MEGGER.

NEW

TMS 34010 HI-RES COLOR BOARD

Bring workstation graphics to your PC! This board has its own 32-bit graphics CPU running at 5 MIPS, with up to 4 M Bytes of memory plus 1 M byte display memory. Emulates CGA. Up to 1024 by 800 pixels, 256 colors from a palatte of 256k!

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~~ NOW $750.00
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

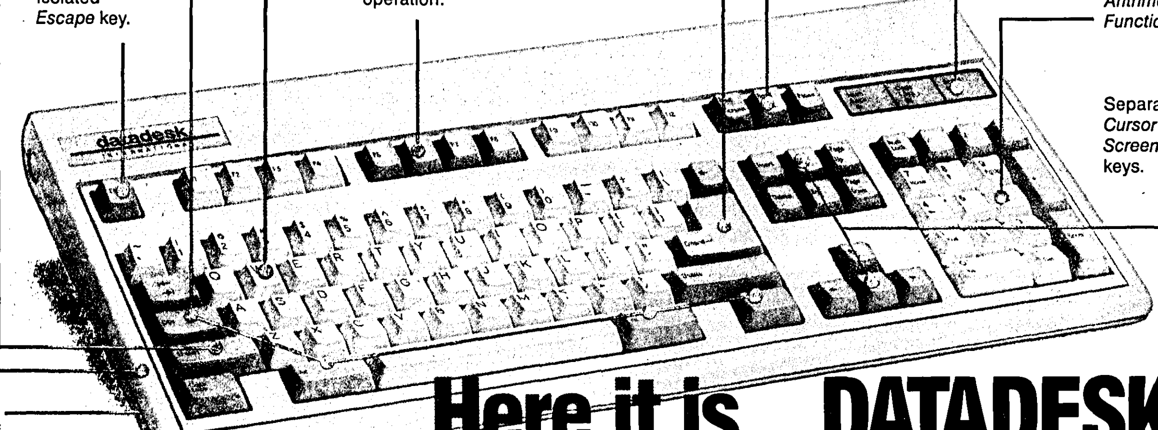
P.O. Box 128
904 North 6th Street
Lake City, MN 55041
(612) 345-4555

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

Borland's Turbo Lightning FREE



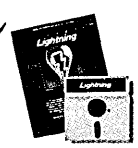
Here it is...DATADESK'S TURBO-101 ENHANCED KEYBOARD for the 10 million PC users IBM just ignored!

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

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: ______

CITY: ______ STATE: ______ ZIP: ______

COMPUTER TYPE: ______

PHONE: ______

CC NO.: ______ EXP: ______

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

The Only Alternative

datadesk™

INTERNATIONAL

7650 Haskell Avenue
Van Nuys, California 91406 (818) 780-1673

THE MICRO TECHNICAL JOURNAL

MICRO CORNUCOPIA

Editor & Publisher
David J. Thompson

Associate Editors
Gary Entsminger
Cary Gatton

Technical Department
Larry Fogg

Director of Advertising
Laura Logan

Director of Marketing and Graphics
Julie Hastings

Distribution 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)	$26.00
1 yr. (Other foreign)	$36.00

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

ISSN 0747-587X

AROUND THE BEND

By David Thompson

Closet Publisher

For the past six years I've pretended to be a hacker, an engineer, a software writer, and a technician - while day after day I've secretly spent my hours doing something very different. Now it's time to fess up.

I've secretly spent these six years as a closet publisher. (No, I don't publish closets.) I maintained a good cover: pretending to take tech calls, letting people think I wrote all those voluminous pieces, and accepting credit for SOG. But I must finally admit - I'm really a publisher.

Oh, I tried to give it up. For weeks I attended monthly Publishers Anonymous meetings. I even tried playing (first chair synthesizer) with the Salvation Army. But it was useless.

This isn't the first time I've faced such an addiction. There was my first computer (a Kim board) and the ecstasy of my first Pascal experience.

Desktop Publishing

But now we have our own software. Desktop publishing has forced publishing out of the closet and into the computer. Of course, some folks were still suspicious when I suggested we run desktopping articles in Micro C.

"Hey, we're a journal. Our readers are programmers and hardware hackers, not squinty-eyed typesetters."

"Desktop publishing is for anti-techies who crave the ultimate WordStar."

They have a point, but I've spoken to four user groups in the past three months and every group wanted to hear about desktop publishing. Was it real? What would it do? Had I solved the output problems? How did it look? Was it cheaper, faster, easier? What about graphics? Could I show them how it worked?

During those talks it was the LSI crowd that was latching onto every word. The questions came so thick and fast that 7 p.m. meetings were breaking up at midnight (so the discussion could adjourn to the local pub).

At the Portland PC meeting (May 13th), I began my presentation by posing a couple of questions to the standing-room-only audience.

"How many of you are currently using desktop publishing?"

(continued on page 89)

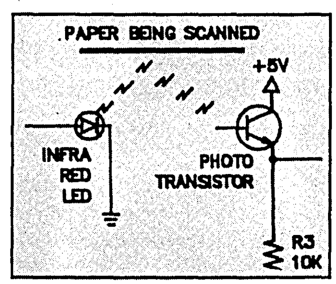
THE MICRO TECHNICAL JOURNAL

MICRO CORNUCOPIA

SEPT./OCT. 1987 - ISSUE NO. 37

FEATURES

COLUMNS



CP/M CORNER

FUTURE TENSE

LETTERS

MIDI Info

I would have written sooner, but I didn't realize that so many Micro C readers are interested in MIDI. I've designed an interface between a Centronics printer port and a MIDI network.

This output-only interface uses commonly available ICs and costs less than $25 to build. I wrote an article for *Computer Music Journal* (vol. 10, no. 3, pp. 79-82) describing the interface. Back issues of *CMJ* can be purchased for $8.50 from:

MIT Press - Journals Dept.
55 Hayward St.
Cambridge, MA 02142

Or, you should be able to get a copy of the article for $4. If your readers just spent their last dollar on a hard drive, they can send me a stamped, self-addressed envelope and I'll return a schematic.

The article deals with the hardware only. An excellent exposition of the MIDI codes appears in an article by R. Moog in the *Journal of the Audio Engineering Society* (vol. 34, no. 5, pp. 394-404). A copy of the article costs $3. Write to:

Audio Engineering Society
60 E. 42 St.
New York, NY 10165-0075

D. M. Gualtieri
Allied-Signal Inc.
P.O. Box 1021-R
Morristown, NJ 07960

PD32 Blues

Just thought you should know about my experiences so far with the PD32 project since Micro C has supported it. I was very interested in the idea since it offered an inexpensive way to add processing power to my machine. I was dying to get started!

However, after six months and four letters, three to Dan Efron and one to Dave Chen (of Cybertools), I've received nary a whisper in reply. I understand that the board was delayed, but thought that at least someone would answer my questions.

So I'm frustrated and can only hope that my experience is rare, though the odds of that seem slight.

David Hillman
2006 NE Davis
Portland, OR 97232

More PD32

Cybertool Systems declared Chapter 7 bankruptcy in May. Dave Rand has asked Definicon Systems (and we have agreed) to be the sole supplier of PD32 kits. Definicon is shipping assembled and tested (warranted) units while Dave is still the sole supplier of the UNIX ports.

Look, all you out there bitching about Definicon ... You have no idea how much effort was needed to make the PD32 work. You surely read of the Japanese RAM embargo? How can we ship kits if the government(s) keep us from getting the RAM?

The accounting foul ups are not excusable, but happen rarely (I hope). They were due to the fact that invoicing thought the boards had shipped at a time that engineering was holding them back. That was our fault for trying to keep you all happy by accelerating the shipping process when the design we had been handed did not work.

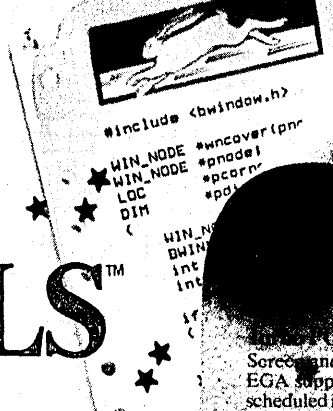
Trevor Marshall
Definicon Systems
1100 Business Circle Dr.
Newbury Park, CA 91320

Editor's note: I was a bit concerned about the PD-32 project when it was first announced at SOG V (July 1986). I was concerned that Dan Efron, who volunteered to be the contact for the kits, would be too swamped by school to handle the load. However, Dave Rand and George Scolaro had a working board, the UNIX port was done, and a lot of people were excited.

(continued on page 94)

More Magic from Blaise.

Turbo C TOOLS™



Magic is easy with Turbo C TOOLS in your bag of tricks. New Turbo C TOOLS™ from Blaise Computing is a library of compiled C functions that allows you full control over the computer, the video environment, and the file system, and gives you the jump on building programs with Borland's new C compiler. Now you can concentrate on the creative parts of your programs.

The library comes with well-documented source code so that you can study, emulate, or adapt it to your specific needs. Blaise Computing's attention to detail, like the use of function prototyping, cleanly organized header files, and a comprehensive, fully-indexed manual, makes Turbo C TOOLS the choice for experienced software developers as well as newcomers to C.

Turbo C TOOLS provides the sophisticated, bullet-proof capabilities needed in today's programming environment, including removable windows, "sidekickable" applications, and general interrupt service routines written in C.

The functions contained in Turbo C TOOLS are carefully crafted to supplement Turbo C, exploiting its strengths without duplicating its library functions. As a result you'll get functions written predominantly in C, that isolate hardware independence, and are small and easy to use.

Turbo C TOOLS embodies the full spectrum of general purpose utility functions that are critical to today's applications. Some of the features in Turbo C TOOLS are:

- **WINDOWS** that are stackable and removable, that have optional borders and a cursor memory, and that can accept user input.
- **INTERRUPT SERVICE ROUTINE** support for truly flexible, robust and polite applications. We show you how to capture DOS critical errors and keystrokes.
- **INTERVENTION CODE** lets you develop memory resident applications that can take full advantage of DOS capabilities. With simple function calls, you can schedule a Turbo C function to execute either when a "hot key" is pressed or at a specified time.
- **RESIDENT SOFTWARE SUPPORT** lets you create, detect, and remove resident utilities that you write with Turbo C TOOLS.
- **FAST DIRECT VIDEO ACCESS** for efficiency, and support for all monitors including EGA 43-line mode.
- **DIRECTORY AND FILE HANDLING** support let you take advantage of the DOS file structure, including volume labels and directory structure.

In addition to Turbo C TOOLS, Blaise Computing Inc. has a full line of support products for Microsoft, Lattice and Datalight C, Microsoft Pascal and Turbo Pascal. Call today for details, and make magic!

Turbo C TOOLS supports the Borland Turbo C compiler, requires DOS 2.00 or later and is just $129.00

BLAISE MENU

POWER TOOLS PLUS *$99.95*
Screen and window management including EGA support; DOS memory control; ISRs; scheduled intervention code; and much more. For Turbo Pascal.

Turbo POWER SCREEN
COMING SOON! General screen management; paint screens; block mode data entry or field-by-field control with instant screen access. For Turbo Pascal.

Turbo ASYNCH PLUS *$99.95*
Interrupt driven support for the COM ports. I/O buffers up to 64K; XON/XOFF; up to 9600 baud; modem and XMODEM control. For Turbo Pascal.

PASCAL TOOLS/TOOLS 2 *$175.00*
Expanded string and screen handling; graphics routines; memory management; general program control; DOS file support and more. For MS-Pascal.

C TOOLS PLUS *$175.00*
Windows; ISRs; screen handling; multiple monitors; EGA 43-line text mode; direct screen access; DOS file handling and more. For MS and Lattice C version 3.00 and later.

LIGHT TOOLS *$99.95*
Windows; ISRs; EGA 43-line text mode; direct screen access; DOS file handling and more. For the Datalight C compiler.

ASYNCH MANAGER *$175.00*
Full featured interrupt driven support for the COM ports. I/O buffers up to 64K; XON/XOFF; up to 9600 baud; modem control and XMODEM. For C or MS-Pascal.

VIEW MANAGER *$275.00*
General screen control; paint screens; block mode data entry or field-by-field control with instant screen access. For C or MS-Pascal.

RUNOFF *$49.95*
Text formatter for all programmers; flexible printer control; user-defined variables; index generation; general macro facility. Crafted in Turbo Pascal.

EXEC *$95.00*
NEW VERSION! Program chaining executive. Chain one program from another in different languages; specify common data areas; less than 2K of overhead.

TO ORDER CALL TOLL FREE
800-333-8087
TELEX NUMBER-338139

YES! I want to make magic!
Enclosed is $________ for ________ copies of ________.
☐ Please send me more information on your products.
CA residents add Sales Tax. Domestic orders add $4.00 for UPS shipping, $10.00 for Federal Express standard air.
Name: ________ Phone: (____) ________
Address: ________ State: ________ Zip: ________
City: ________
VISA or MC#: ________ Exp. Date: ________

BLAISE COMPUTING INC.
2560 Ninth Street, Suite 316 Berkeley, CA 94710 (415) 540-5441

Turbo C is a trademark of Borland International.


Micro Cornucopia

By David Thompson

Desktop Publishing On A PC

Or: How We Produced This Issue Of Micro C

I was seduced while doing issue #35 and it was an interesting experience, so interesting that the whole office will remember it for a long time.

I was seduced by power. The power to see into the future, to see the printed page before it was printed. For the first time I felt I had control over the next issue of Micro C. Getting the issue onto the screen was not trivial, but it was a lot easier than I expected. However, getting it off the screen and onto paper, the trivial part, turned out to be anything but.

In the first part of this article, I'm going to bore you with details. The reason is not to drive you away but to give you a feeling for the incredibly tedious process we were going through to create an issue of Micro C.

Understanding a bit about the old process will help you appreciate what desktop publishing (Ventura, specifically) has done to lighten our load. Desktop publishing is as much an improvement over the old methods as word processing is over the manual typewriter. Once you're rolling you can't go back.

The Beginning

In issues #1 through #34, I manually added typesetting commands to each article before sending it to the local newspaper to be typeset. I'd specify the type size (9 point for text), font (usually Palatino), line length (13.6 picas), vertical leading, and so on. If I needed a boldfaced subhead (like "The Beginning" above), then I'd change the font to Palatino bold. At the end of the subhead, I'd change back to the regular font. (At least when I remembered.)

Once I'd installed the typesetting commands, I'd upload the articles to the computer at the local paper (*The Bend Bulletin*). They output the type on their Compugraphic typesetter.

We had ways of proofing the writing. After all we had three editors, two spelling checkers, and Larry. But it was very difficult to proof the formatting. So typesetting was always a surprise. Not necessarily a pleasant surprise, but a surprise.

There were times when entire articles came out as headlines (popular with the authors, but a 95-page techtip wouldn't cut it with readers). And there were times when the last half of the editorial was bold-faced (lies or otherwise).

Each issue was different and each issue saw me wearing a path to *The Bulletin.* (It got so they'd save me a place on the coat rack when it was time for corrections.)

Just Our Type

We got the type from *The Bulletin* in galleys; ten-foot strips of copy, each strip the width of one column.

After the galleys arrived, we'd all pitch in. Laura and Tracey would cut the strips into articles, Tammy would xerox the cut-up articles, and Julie would begin measuring how many inches of type we had for each article. To that measurement, Julie would add space for listings, tables, and headlines to figure out how many pages for each article. Then she'd start laying out the magazine. There would be a hardware article here, Microsphere's ad on the facing page, a column on C next - Manx's ad nestled nearby. And so on, until she ran out of material or pages.

Usually she ran out of pages. So she'd have to figure out what to do with the leftovers: two one-third page ads, the rest of the editorial (an incomprehensible figure), and the advertisers' index.

So, she'd juggle.

"Would anyone notice if we stuck this Pascal figure in the C column?"

(Probably. They noticed last time.)

"Maybe we could divide up the editorial into article-sized chunks and run continuations for the next six issues."

(And maybe I could start writing for the *National Enquirer.*)

After a week, Julie had a rough idea where things would fit so Tracey, Laura, and Tammy could begin paste-up, the desperate week-and-a-half of shifting

There were times when entire articles came out as headlines...

around bits of type, sticking things in place with hot wax, sizing listings, resizing listings, reducing listings, pulling things up, and:

"You can't shorten that article, I just pasted it down. Shorten Tammy's."

"You cut mine last time, cut Laura's."

It was a good-natured give and take. Usually. And in the end, all the type would be stuck down and there would be spots of whiteout here and there to cover the blood and sweat marks.

Then we'd xerox again and start proofing. As much as we proofed the raw text, we would always find a few surprises in the finished piece. There's something about a finished layout that breeds errors. (There's no other explana-

(continued next page)

(continued from page 7)

tion for it.) It takes a skilled hand and a significant amount of time to remove an erroneous word from the middle of a paragraph without leaving a hole (or add a word without obvious squeezing).

Sometimes I wondered whether I really wanted to take a last pass through the pasted-up pages. I was always afraid that if I didn't, I'd miss a major problem. And if I did, I'd find a major problem.

And of course all this goes on at the last possible moment while everyone's waiting: the cameraman who shoots the negs, which go to the platemaker who burns the plates, which go to the printer who prints the pages, which go to the binder who assembles the magazines, which go to the labeler who fills the bags, which go to the Post Office. And I haven't the slightest idea what the P.O. does with them.

These folks don't care whether I make that last read-through, and they don't care whether Julie stays up most of the night pasting in bits of type so small they vanish under bleary eyes. All they understand is schedule. Sometimes at the end of the process, we get so pushed and so tired that we don't care either.

Desktop Publishing

I purchased a copy of Ventura Publisher in late January, and by the first of February I knew it was a godsend.

After a little futzing, I discovered I could print a very good representation of the final product on a cheap dot-matrix printer. The output was good enough to proofread. Everything (the spacing, line breaks, and graphics) was just like the final product, but printed at 75 dots per inch instead of 1270.

Anyway, around the middle of February I had to make a decision. I could spend a week entering typesetting codes for *The Bulletin,* or I could simply output all those articles that were staring back at me from inside Ventura's screen.

The choice was easy. I would save a week by not adding the old-fashioned typesetting codes. I could output finished-looking pages on the dot-matrix printer so Julie could design the issue in half the time, and the rest of us could do a final proofread on the articles BEFORE they were cast in real type. And, of course, the final paste-up would go much faster because we'd be pasting down complete pages instead of single galleys.

The choice was simple. All I had to do was connect up with one of the half-dozen typesetting outfits in the Northwest that had a Linotronic typesetter and a raster image processor (RIP), and I'd have a pre-produced magazine.

What Really Happened

As you know if you read the editorial in issue #35, it wasn't easy. Murphy had left me alone as I felt my way around Ventura. The style sheets were a piece of cake. Importing ASCII files was duck soup. Marking heads and authors and all that was trivial, and everything showed up on the screen just about like it would appear. And though I wasn't going to include illustrations in Ventura this time, I could create the ruled boxes for them.

Anyway, I had checked out the options for outputting Ventura files before I purchased Ventura, and I knew that three Portland typesetting outfits and one in Salem had RIPs for their Linotronics. When I called them to say I had files, I learned that the RIPs were just being installed.

"We're having the image processor installed Thursday. Come over Saturday and we'll dump them out."

We went to Salem that Saturday and returned to Bend empty-handed. Our connections with Portland outfits turned out to be just as typeless. Then a Seattle company tried, but our files didn't set right with their system.

Finally, I located a Seattle outfit called Wyziwyg. They output issue #35 for us.

It wasn't perfect. We made some mistakes, mistakes that forced Julie to juggle quite a bit of type. But, we did it. And despite everything Murphy could throw, it worked.

Ventura

It's easy for bugs to hide in a program, especially if that program is very complex. And, I have yet to see a version 1.0 of any program that didn't have a full complement of the little fellows.

Well, Ventura is very complex and I'm using version 1.0. So I'd expect lots of bugs. In fact, I'd expect it to be almost unusable. It's not. (I mean, it's very usable.)

It has some bugs, and there are some features that it needs, but on the whole it works well.

Why I Chose Ventura

I spent several months reading everything I could about desktop publishing packages. Sandy and I picked up dozens of desktop press kits at the 1986 Fall Comdex. We read articles, talked to editors, harassed manufacturers, and called typesetters.

I chose Ventura because it was significantly faster and had more features than anything else on the market. I also chose it because, despite its $895 retail price, it appeared to be the least expensive way to put together a complete desktop system.

Ventura is fast enough to run on a standard XT. That and the fact that it works with just about any kind of video graphics card (including the Hercules) meant that we didn't have to purchase any new hardware to begin using it immediately. (We did eventually purchase a Xerox full-page monitor, and it's very, very nice, but not necessary.)

Ventura outputs to a plethora of laser printers, but I chose not to purchase one immediately. Since Ventura lets me proof my material on a standard Epson printer (or Epson compatible such as the newer Gemini 10X), I didn't have to purchase a special printer. (Some folks have told me they're happy using just the dot-matrix output for their newsletters.)

The key to desktop publishing for me is typeset output, and Ventura can produce PostScript files. The key is Linotronic's RIP, which translates PostScript into something that Linotronic typesetters understand.

Note: PostScript is one of several PDLs - page description languages. A PDL is to dots on a page what FORTH is to a .COM file. Both are high-level (ASCII) lists of commands telling the system what you want.

Neither FORTH nor PostScript cares what the final device is. Your FORTH screens should be intelligible to any FORTH interpreter. Your PostScript file should be intelligible to any PostScript interpreter.

Take a look at the "Desktop Printers" article in the next issue for details on PostScript. Meanwhile, if you need to know more about this very powerful language, there are two books you should get. They are the *PostScript Language Reference Manual* ($22.95 ISBN 0-201-10174-2) and the *PostScript Language Tutorial and Cookbook* ($16.95 ISBN 0-201-10179-3). They were written by Adobe Systems Inc. and published by Addison-Wesley. Both are very well done, and I highly recommend them to anyone who's planning to have any dealings with a PostScript device.

The PostScript generator doesn't care what that output device is: its resolution, its speed, anything. The language is device independent. (At least in theory.)

Whatever I chose, it had to support

PostScript so I could get real typeset output, and Ventura supports PostScript.

Style Sheets

Another reason I chose Ventura was because it let me create style sheets. For large publication (or book) size projects, style sheets are a must.

In a style sheet you decide what the page will look like: how many columns (3), their width (13.5 picas), length of the columns, margins around the outside of the page, whether you want vertical rules between columns, and so on.

You also select the type styles, placement, and sizes for: main headlines, secondary headlines, main text, subheads, addresses, footnotes, page headers, footers, etc.

For instance, the body type is 9 point Palatino normal with 11 point leading; it's justified, hyphenated, and it begins with an indent. If I changed my definition of body type to, say, 12 point with 14 point leading, then all the body type would be displayed as 12 point. Also, the article would probably grow from its original 3 pages to about 4-1/2.

Of course, from then on, every article that used that style sheet would have 12 point body type. (At least until I changed it.)

Once you've created your style sheets, laying out a standard article is trivial. You select the style sheet, select the text, mark any special lines (headlines, etc.), and you're done. If there are any special illustrations, you make boxes for them, then grab them from the disk.

Ventura then creates a chapter file. The chapter is essentially a list of all the files that make up an article and a description of how they go together. When you again work on that article, you simply select the chapter. Ventura grabs everything you were using and sticks it all back together, just as it was when you saved it.

Because of the chapter file idea, the original ASCII (or WordStar or WordPerfect...) text files remain ASCII text files. Any changes (corrections, usually) you make while you are in Ventura will be made to the original text file. You can go back into the original file with your normal text editor, see those changes, and make further changes if you wish. This is a very important feature for me.

Other publishing packages, such as PageMaker, create a new file with all the text, graphics, and other instructions. Any changes you make to the text only show up in the new file. (The new file is unintelligible to the original word processor.)

Ventura Vs. PageMaker

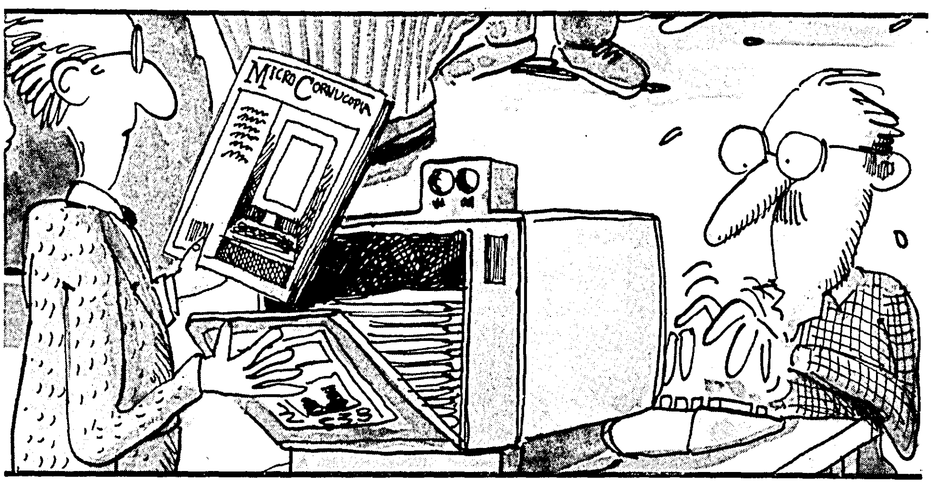
As far as I can see, DP is the biggest game in today's computer marketplace. Ventura is going nuts (deservedly). Aldus isn't far behind with its PC version of PageMaker.

I've heard that PageMaker is easier to learn and more fun to use. It's easier to size, move, and otherwise manipulate text and graphics in PageMaker than in Ventura. These features are important when you're creating lots of different things like ads or brochures. But when you're doing the same thing over and over again (like book chapters or magazine articles), Ventura's style sheets make things a lot faster and easier once you've thrashed through style-sheet creation.

Ideally, you could do it both ways: freehand manipulation of text and graphics, with style sheets for the repetitive stuff. Both Ventura and PageMaker are moving toward that combination, each from its own corner.

Both have very good documentation, and both give you a Macintosh-like work area with mouse-driven pull-down menus. It's an environment that's very easy to learn and very comfortable down the road. It's the first time I've worked with a mouse this way, and it's fun. Really fun. I can see why the Mac is so popular with non-computerists.

Ventura works under GEM; PageMaker works under Microsoft's Windows. These environments do the screen windowing (with many different monitor types), the printer I/O (to all the different printers), and data passing between programs. Windows supposedly supports more monitors and printers and lets you keep more programs coresident. GEM is quite a bit faster, the main reason that Ventura isn't unworkably slow on a standard XT.



The choice was easy. I would save a week by not adding the old-fashioned typesetting codes.

Ventura requires a hard drive. It comes on 11 disks (not copy protected). Initialization is easy and quite fast.

Support

My contacts with Ventura Software Co. have been very cordial, but Ventura is not the place to call for help. Xerox is handling sales and support, but Xerox hasn't been ready for prime time.

I called Xerox because we were getting garbage on the Xerox monitor whenever we deleted text from a Ventura frame. I never reached tech support. Most of the times I called the tech number, I got a busy signal. When it wasn't busy, I'd get a recording:

"All our lines are busy, please hold on and the next available technician will help you."

Once I waited nearly an hour (it's not a free call) before hanging up.

(continued next page)

(continued from page 9)

My other contacts with Xerox have also been less than satisfying. I called to get pictures of the package for this article. It took me three days and about ten calls to locate the person with the pictures.

Also, Xerox sent out a letter to all us registered owners saying we could update from version 1.0 to 1.1 for $100 (or $35 if we had purchased after some date in March). It's a significant update (as well as bug fix), so even at $100 it's quite a bargain. The day the letter arrived, I called the number on the letterhead. The phone company intercepted and gave me a new number. I called the new number, got Xerox switchboard, and asked for the person who had signed the letter. They hadn't heard of him.

I asked about the original number.

"Oh, we haven't used that number for a long time."

Numerous other times I've tried contacting Xerox people, and numerous times they've been unavailable. Very few have returned calls.

I've only contacted Aldus twice. Both times the people I spoke with were very anxious to help and there was no problem getting someone to call me back.

More Desktop Publishers

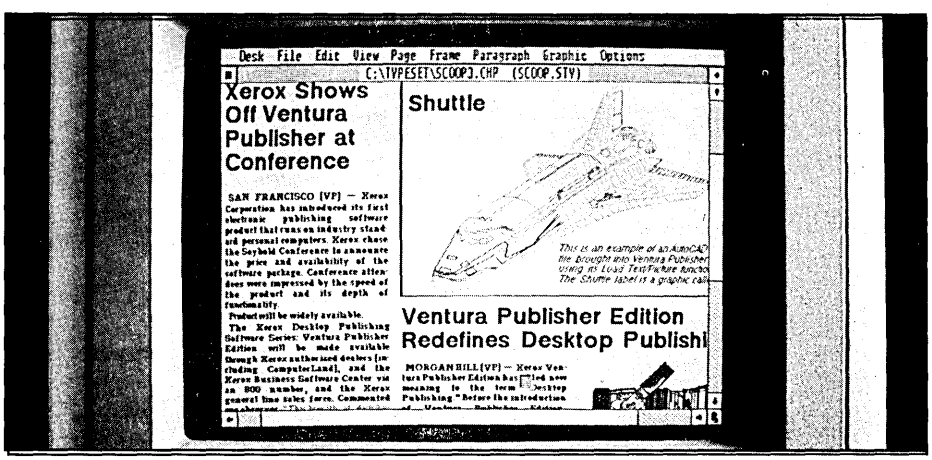
The other desktop software outfits have so far been also-rans. That could change very quickly as newer, fancier, and cheaper packages hit the market. Meanwhile, however, Ventura and Aldus are improving their products so quickly that they are definitely presenting moving targets.

Also, the folks doing the heavy-duty word processing packages, such as WordPerfect, aren't ignoring DP. I'd expect to see these packages gain DP skills very quickly. Some of these outfits will no doubt write their own publishing features. Others will probably shortcut the process by buying out DP also-rans and integrating the code into their word processors.

There are also lots of parallels between desktop publishing and spreadsheets. Spreadsheets started out as expensive, limited-ability, single-function packages aimed at the business user. Since then, prices have come down, the programs are faster and accept more data, and many now have graphics, word processing, and communications.

The Future

Desktop packages are now in their first phase. Their strength is their ability to combine typeset quality text and graphics and display them on a graphics monitor as they'll appear when they're output through a laser (or other) printer. But so far, their editors are crude and their ability to create or edit graphics (other than sizing or cropping) is almost nonexistent.



Both the desktop software and hardware suffer from a shortage of typefaces and (in the case of pixel fonts) sizes. Helvetica and Times Roman are much nicer than the typewriter faces that have dominated computer output; but now that we've seen the possibilities, we aren't satisfied. We want Helvetica Condensed Shaded, Horatio Light, Tabasco Bold, Silver Screen, Playbill, Old English, and Loose New Roman. (I didn't make these up, folks.) If you collect all the world's graphic designers, you'll collect lots of faces.

Finally

I don't feel like I've given you the kind of technical detail Micro C is noted for: Where are the bugs in Ventura? How does PostScript really work? How about all those other little environments? GEM? Windows? HPGL? Output devices? (They're important, but I haven't even mentioned them.)

I also haven't fully shared the excitement. In fact, I haven't seen a single article that prepared me for the feelings I had when I saw issue #35 taking shape on the screen. Everything I'd read talked about control. Control is great, but desktop publishing is a lot more, especially for a publication freak like myself.

Toward the end of my DP presentation to the Portland PC group, I was unmercifully stretching a drawing of a nozzle. The crowd gasped as the text flowed smoothly around the strange shape.

"Now do you understand why I couldn't go back?" I asked.

There was an instant ovation. They understood.

To Be Continued

Desktop publishing isn't much good if you can't get it onto paper. Next issue I'll talk about the trials and tribulations of printing and PostScripting. It'll give me another chance to overwhelm you with details.

Meanwhile, if you're looking for a very good desktop publication to curl up with, try:

Publish!
P.O. Box 51966
Boulder, CO 80321-1966
$23.95 a year, 12 issues.

Access Information

PageMaker ($695)
Aldus Corp.
411 First Ave. So., Ste. 200
Seattle, WA 98104
(206) 622-5500

Ventura Software (Developer)
675 Jarvis Dr. #C
Morgan Hill, CA 95037
(408) 779-5000

Xerox Corp. (Sells Ventura $895)
101 Continental Blvd.
El Segundo, CA 90245
(213) 536-7000

■■■

Intro To Database Programming

Part 3, Wrapping Up Relationships

Database programmers (and lovers of a good story), it's time to wrap up another episode in the black box opera, and to answer those burning questions about relational database design. Those taking their first plunge into the relational lake, who want more introductory details, should slip into something more comfortable and back to issues #35 and #36. The rest of you, grab your trunks, and get ready for the windup of the Dee Base and Dr. Dobbs Tale.

Our story so far: In previous episodes we met Dr. Dobbs, a veterinarian who decides he wants to automate his front office. He arms himself with a shiny new PC and dBASE III, and takes the plunge into the relational lake.

Unfortunately, due to his lack of experience with relational databases, he quickly makes a mess of things. His first attempts at a database design are entirely unsuccessful, and due to billing errors, one of his best clients goes away growling. He finally seeks out the services of a consultant.

Enter Dorothy (known as "Dee") Base. Dee has considerable experience in database design, and she quickly sets Dobbs back on track. Together, they analyze his business needs and, using the Entity-Relationship model for database design, they develop a set of tables that will serve all of Dobbs' business needs. Next, they design the screens and reports that the system will produce, and from this they create a list of data elements that they assign to the tables on a "best guess" basis. During the course of their work together, Dobbs finds himself highly attracted to Dee. And here our story continues...

Table Normalization

Now that Dobbs has identified the tables that his system will need, and has assigned data elements, Dee suggests that they verify the correctness of these guesses by normalizing the tables. (Dobbs is, of course, enthusiastic about anything Dee wants to do.)

Relational tables are considered to be "normalized" if they satisfy certain specified sets of constraints. These constraints are applied in order to reduce common data anomalies, such as redundant data or loss of data integrity.

Books on database theory go into lengthy definitions of the different levels of normalization, starting with the first normal form (1NF) and gradually applying more rigorous constraints until the fifth normal form (5NF), generally considered to be the "final" normal form, is reached. However, a detailed discussion of the hierarchy of normal forms isn't really necessary here, and you can refer to a book on database design if you wish to find out more. The application of three simple rules will actually suffice for normalizing most tables.

In order to better illustrate the process of normalization, we'll use an example based on Dobbs' veterinarian practice. The following is the kind of table that might have been created, using traditional file design methods, in order to keep track of the information about pet visits:

Client (pet owner) name
Client address
Client's balance from previous bill
Pet name
Pet species
Pet breed
Pet color
Pet age
Pet visit date
Procedure performed
Fee for procedure

We add a record to this table for every procedure performed on a client's pet.

The first step in normalizing this table is to identify its key. Remember that the key is the field, or combination of fields, that will identify each record in the table uniquely. In our example table, it would take the following combination of fields to serve as key:

Client name + Pet name + Visit date + Procedure performed

Once the key has been identified, we can apply the following rules for normalization.

RULE ONE : All elements in the table should relate to the KEY. In our example, all of the elements relate directly to at least one of the fields in the key.

RULE TWO : All elements in the table should relate to the WHOLE KEY. Several elements in the example violate this rule. The client address and client account balance, for example, relate directly to only a part of the key, the "client name." Likewise, the pet info fields relate only to the "pet name."

These data elements should be moved out into new tables, along with whatever pieces of the key are necessary to uniquely identify the records in the new tables. The client information will be moved into a new table keyed by client name, which we'll call the Client table:

Client name
Client address
Client's account balance from previous bill

We move the pet information into a table keyed by client name plus pet name, since both elements are needed to fully identify a particular pet. The resulting Pet table looks like this:

Client (pet owner) name
Pet name
Pet species
Pet breed
Pet color

By Sandy Brabandt
6424 Sunnyfield Way
Sacramento, CA 95823

Pet age
Pet visit date
Pet visit date

The original table, now functioning as the Visit/Procedure table, will look like this:

Client name
Pet name
Pet visit date
Procedure performed
Fee for procedure

You can probably see that the application of the "whole key" rule of normalization results in a dramatic reduction of redundant data.

Note also that there's enough information in the three tables to recreate the original table using a "join" operation (see the first article in this series, Micro C #35, for a discussion on the relational "join"). The Client table and Pet table can be joined on the "client name" field, and then the resulting table can be joined to the Visit/Procedure table on the combination of "client name" + "pet name".

The ability to join the decomposed tables until the original table is formed is an important cross-check to determine whether the tables are keyed correctly.

If, for example, the Pet table had been keyed on "pet name" only, without the "client name," join would not have worked and important information (in this case, who the pet belongs to) would have been lost.

RULE THREE : All elements in the table should relate to NOTHING BUT THE KEY. The "procedure" element violates this rule. Although it does relate directly to the key (it is, in fact, part of the key), it can also stand independently.

In the current setup, a procedure will only be represented in the table if it has been performed on a certain pet. There is an additional need to store information about each procedure independently of its relationship to pet visits. To facilitate this, a new table, the Procedure table, is created, keyed on the procedure name:

You can normalize most tables by applying three simple rules.

Procedure name
Normal fee for procedure

This table will contain one entry for each procedure that Dobbs might perform, along with the normal fee he would charge for that procedure. This will serve as the "master table" of Procedures. The "nothing but the key" rule of normalization will often result in the creation of master tables such as this.

The Visit/Procedure table retains the same elements it already had, but the "procedure" and "fee" elements are defined more specifically, to indicate that they relate directly to a particular visit:

Client name
Pet name
Pet visit date
Procedure performed during visit
Actual fee charged for procedure

You can probably see that the tables we've ended up with, as a result of the normalization process, bear a strong resemblance to some of the tables identified as part of the Entity-Relationship modeling. In fact, the Entity-Relationship design process gives you a considerable head start on the normalization process.

Once the tables have been identified via the model, and data elements have been assigned intuitively, you can use the normalization rules (making sure all data elements in the table relate directly to the KEY, the WHOLE KEY, and NOTHING BUT THE KEY) to determine whether all of the data elements were, in fact, assigned to the correct tables. If the design was carefully done, you'll find that very little normalization is needed.

Figure 1 shows what the vet system tables look like after data elements have been assigned and normalized. The primary keys are labeled "PK" and the foreign key "FK" (see part two of this article, Micro C #36, for a discussion on primary and foreign keys).

Now that the tables for the vet practice have been successfully normalized, Dobbs fears that Dee's work with him may be over. However, he's exceedingly pleased to find that several steps remain in what Dee considers to be a complete design process.

They work through these steps during evening sessions that Dobbs manages to stretch out over weeks, but I'll spare you the details and simply summarize their discussions.

I'll also spare you the details of Dobbs' flirtation with Dee, starting with the first casual contact of their fingertips as they work together on the micro keyboard, and leading up to that fateful moment when Dobbs confronts her in his back office, and pledges his undying devotion, and she - no, wait; that can wait until later. For now, let's get back to the system design.

Field Lengths

Dobbs and Dee next decide on the length of each data field.

(continued next page)

(continued from page 13)

Field lengths can be changed when using a relational database system, but since this may have an impact on some aspects of the system, such as screen and report formatting, it's best to try to come up with the optimum field lengths at the beginning. The field should be long enough to contain the largest reasonable piece of data; but at the same time, if the field is longer than necessary, disk space will be wasted.

Edit Criteria

They also established the edit criteria for the fields. In some relational database systems, the edit criteria for each field can be defined as an integral part of the field definition.

In dBASE III, most of the editing must be done by the programs. The fields should be edited for proper format (such as a "last name, first name" format for client name, or a "month/day/year" format for dates), and for valid values (for instance, the procedure name field in the Visit/Procedure table must contain the same procedure names as those listed in the Procedure table).

Data Integrity

Next, they discuss data integrity.

It's up to the application programs to make sure that all items are properly inserted and maintained. For instance, the program must make sure the Client ID of the pet owner is placed in every pet record.

Also, the program should ensure that all the requisite "parent" segments exist for every "child" segment. In other words, the client ID in the pet record had better point back to an existing client record, and a client record should never be deleted as long as there are pet records (or any other of its "child" segments) pointing back to it.

Another data integrity issue concerns the table key definitions. Early on in the design process, Dee encouraged Dobbs to change the key of the client table from "client name" to a new field, "client ID."

This ID is a serially assigned number. It's a better key than the "client name" field for three reasons: (1) it takes up less space as the foreign key in the "child" tables belonging to the client table; (2) while a client's name may change, the ID would never have to be changed; and (3) it will always be unique, eliminating the problems that might occur if two John Smiths bring their pets to Dobbs. There is one drawback to using a numeric ID as a

Figure 1 — Vet system after data elements are assigned and normalized.

```
CLIENT TABLE
    Client ID  (PK)
    Client Name
    Client Address
    Client account balance (current)
    Client account balance fwd from previous bill

BILL TABLE
    Billing date  (PK)
    Client ID (ID of billed client) (FK)
    Balance from previous bill
    Total payments since last bill
    Total charges since last bill
    New balance forward

PAYMENT TABLE
    Payment date  (PK)
    Client ID (ID of client making payment) (FK)
    Payment amount
    Mode of payment (cash, check)
    Flag: has payment shown on bill?

PET TABLE
    Pet name  (PK)
    Client ID (ID of pet owner) (FK)
    Pet species
    Pet breed
    Pet age
    Pet color

PET VISIT TABLE
    Visit date (PK)
    Pet name (Name of visiting pet) (FK)
    Client ID (ID of pet owner) (FK)
    Total charge for visit
    Comments about visit
    Flag: has this visit been billed?

PROCEDURE TABLE
    Procedure name (PK)
    Normal fee charged for procedure

MEDICATION TABLE
    Medication name (PK)
    Normal fee charged for medication

VISIT/PROCEDURE TABLE
    Visit date  (FK)
    Name of visiting pet (FK)
    ID of pet owner (FK)
    Procedure name  (FK)
    Actual fee charged for procedure

VISIT/MEDICATION TABLE
    Visit date  (FK)
    Name of visiting pet (FK)
    ID of pet owner (FK)
    Medication name (FK)
    Dosage of medication administered
    Mode of medication adminstration (pill, injection, etc.)
```

Figure 2 — Vet system (Figure 1) with new keys.

```
CLIENT TABLE
    Client ID  (PK)
    Client Name
    Client Address
    Client account balance (current)
    Client account balance fwd from previous bill

BILL TABLE
    Billing date  (PK)
    Client ID (ID of billed client) (FK)
    Balance from previous bill
    Total payments since last bill
    Total charges since last bill
    New balance forward

PAYMENT TABLE
    Payment date  (PK)
    Client ID (ID of client making payment) (FK)
    Payment amount
    Mode of payment (cash, check)

PET TABLE
    Pet name  (PK)
    Client ID (ID of pet owner) (FK)
    Pet species
    Pet breed
    Pet age
    Pet color

PET VISIT TABLE
    Visit ID (PK)
    Visit date
    Pet name (Name of visiting pet) (FK)
    Client ID (ID of pet owner) (FK)
    Total charge for visit
    Comments about visit
    Flag: has this visit been billed?

PROCEDURE TABLE
    Procedure code (PK)
    Procedure name
    Normal fee charged for procedure

MEDICATION TABLE
    Medication code (PK)
    Medication name
    Normal fee charged for medication

VISIT/PROCEDURE TABLE
    Visit ID (FK)
    Procedure code (FK)
    Actual fee charged for procedure

VISIT/MEDICATION TABLE
    Visit ID  (FK)
    Medication code (FK)
    Dosage of medication administered
    Mode of medication adminstration (pill, injection, etc.)
```

key to the Client table. For the people using the system, it's a whole lot easier to deal with the client's name. However, this need not be a problem if the system is well-written.

The office staff can identify the patient by name when using the system; the programs can then keep track of the client ID internally, and use the ID to make connections to the related tables. There's no need for the computer users to remember the ID or even to know that one exists.

The Visit table is another area which would benefit from using a numeric key. Since a pet could conceivably be brought in for two visits on the same day, the current key (visit date + client ID + pet name) doesn't guarantee uniqueness.

This will also save a great deal of space in the Visit/Procedure and Visit/Medication tables, which currently have to carry as a foreign key the entire Visit table key.

The Procedure and Medication tables are also good candidates for numeric IDs. This is mainly because it's easier for the computer user to type in a five-digit code than a 30-byte name when identifying a procedure or medication. And once again, considerable space is saved in the Visit/Procedure and Visit/Medication tables.

Numeric IDs are probably not such a good idea for the Billing, Payment, and Pet tables. Their keys appear to ensure uniqueness, very little space would be saved, and the overhead of maintaining a numeric ID isn't justified.

Figure 2 shows tables after the introduction of the new keys.

Indexing

Dee next suggests they could speed the system up (considerably) by indexing some of the tables on strategic fields. She explains a couple of critical ways in which relational databases differ from more traditional file structures.

In some file structures, the records must be physically stored in some kind of order, sequenced by the key, and this key is the only field that can be used to retrieve a record.

In relational databases, records can be stored in any order, with the key serving only to uniquely identify each record. In addition, records in relational tables can be retrieved using any field or combination of fields.

This setup makes relational databases very flexible since any record can be

(continued next page)

(continued from page 15)

retrieved using any criteria. However, it also slows record retrievals.

Because the records are in random order, the only way the database management system can find the requested record is by doing a sequential scan of the file, looking for the specified search criteria. If the record to be retrieved is near the beginning of the file, great; but if it's near the end, it's not so great.

To build an index for a dBASE III file, you use a field or combination of fields as index fields. Thus you create an index file which contains a record for each record in the original file.

Each record in the index file contains two pieces of information: a copy of the data in the index fields of the original record, and a relative record pointer to that original record. For example, if the client file contained five records for the following clients:

1. Bowwow, Edith
2. Smith, John
3. Jackson, Joe
4. Cunningham, Pat
5. Langley, Jane

then the index file would contain the following records:

Bowwow, Edith === record 1
Cunningham, Pat === record 4
Jackson, Joe === record 3
Langley, Jane === record 5
Smith, John === record 2

The records in the index file will be chained together using an algorithm that enables any index record to be located quickly. This means that any record in a dBASE III file can be retrieved in seconds via the index.

However, there *is a price to pay* for the improved speed of record retrievals. There's quite a bit of overhead in index maintenance.

Every time a record is added to an indexed table, or when the contents of index fields are changed, all indexes against that file must be correspondingly updated. To decide whether this overhead is justifiable, you should compare the number of indexed retrievals versus the number of index file updates.

Indexed Example

Let's use the Client table as an example. What are the most common ways in which this table will be accessed?

You've already noticed that the staff will normally use the client's name as the key for retrieving the client record. This will be a very frequent retrieval path, since many of the processes performed on the computer - such as viewing or updating a client record; adding, viewing, or updating a pet record; recording a client payment; and recording a pet visit

His heart is beating wildly with the realization that unless he says something quickly, his relationship with Dee will terminate.

- begin with a retrieval of the client record by the name.

The update frequency against the client table is, in contrast, relatively small. Updates against a client name index would only occur when a client is added to the file, or a client name is updated. Therefore, the Client table seems to be a good candidate for an index against the client name.

There are two more good reasons to index the Client table on the client name. For one, you'll probably have applications that will want to retrieve client records in name order; for example, to print mailing labels or to report on overdue accounts. The index lets you do this kind of sequential retrieval without having to sort the table first.

Another reason for indexing the Client table is that it will probably be a fairly large file, containing several hundred records, and therefore the performance improvement of indexed retrievals vs. serial retrievals will be dramatic. On a smaller file of only 20 to 30 records, the performance difference would not be noticeable.

Wrapping Up A Relationship

At this point, Dee feels her work on Dobbs' office system is complete. The relational tables have been established, complete with their data elements. The screens and reports have been designed, the indexes put into place, and the edit criteria established. "You shouldn't have any trouble coding the system from this point," she tells Dobbs.

His heart is beating wildly with the realization that unless he says something quickly, his relationship with Dee is about to terminate. He asks her into his back office under the pretense that his checkbook is back there, and once there, he turns and confronts her. "I've enjoyed every moment with you," he says fervently, "analyzing data relationships, normalizing, building indexes... Do you think we could make our professional relationship into something more - well, more personal? And more permanent?"

And by way of reply, Dee... sneezes. Yes, sneezes. "Please let me out of here," she says in a voice suddenly grown nasal due to stuffed sinuses. "I'm deathly allergic to cats, I can't abide dogs, and that canary singing in the back room is about to drive me batty!"

The rest, as they say, is history. Dee collected a hefty fee and went on to design databases for many more clients. Dobbs finished writing his office system and was able to reconcile his office staff and even bring back Edith Bowwow (the client who left in a huff in our first episode).

And What About Naomi?

Dobbs' ever-faithful lab assistant, the mysterious Naomi, has been in love with him ever since first seeing his surgical greens, and her patience is about to be rewarded.

Dobbs suddenly (and finally) realizes that Naomi is a woman of multiple charms. She adores cats, dogs, the canary, and him - not necessarily in that order. And he knows that feathers would fly if he didn't start paying her some attention. So they establish a reciprocal relationship and live happily ever after.

■■■

Magic In The Real World:
Controlling AC Power

It might seem like a big leap, setting up your PC to control AC appliances, lights, and large motors. It isn't. Inexpensive opto-isolated triacs accept tiny TTL outputs on one side and control 110 VAC on the other. With the help of snubbers, they can even control motors and other reactive loads. But wait a moment, I'm getting ahead of Bruce. (Now if the PC would monitor the kids' TV...)

On my last trip to California, I was on the lookout for ideas for articles. Walking along a beach in Santa Barbara with my friend Bob, I asked what he'd like to know.

"No offense, Bruce," he said, "you write about some interesting stuff. But I just want to use my computer to turn things on and off."

Wham! Back to the reality that most people need practical solutions to practical problems.

With that in mind, I've written this article to show two ways to control AC power: a simple and cheap way (with risks involved, because you're messing about with dangerous voltages), and an expensive ($400 or so) but somewhat safer and easier way.

For Cheap: Or Whatsa Triac, Mister?

Both methods are essentially identical: we take a bit from an output port of the computer and connect it through an optocoupler (for voltage isolation - TTL and 117VAC are definitely incompatible) to a device called a triac, which turns AC current on and off. Figure 1 shows what the circuit looks like - it's easy to hook up, but there's a lot to learn about if you don't want to abuse the circuit (or your body - THESE ARE DANGEROUS VOLTAGES!!!).

A triac is two Silicon-Controlled Rectifiers (SCRs), one going in each direction.

An SCR is a diode which has a "gate;" the diode won't turn on until you tweak the gate. An SCR is made of two transistors, a PNP and an NPN, which go into a mutual-stimulation routine when you energize them (Oops, almost forgot - this a family magazine).

To understand all this, we need to start with semiconductors, diodes and transistors.

To Conduct Or Not To Conduct

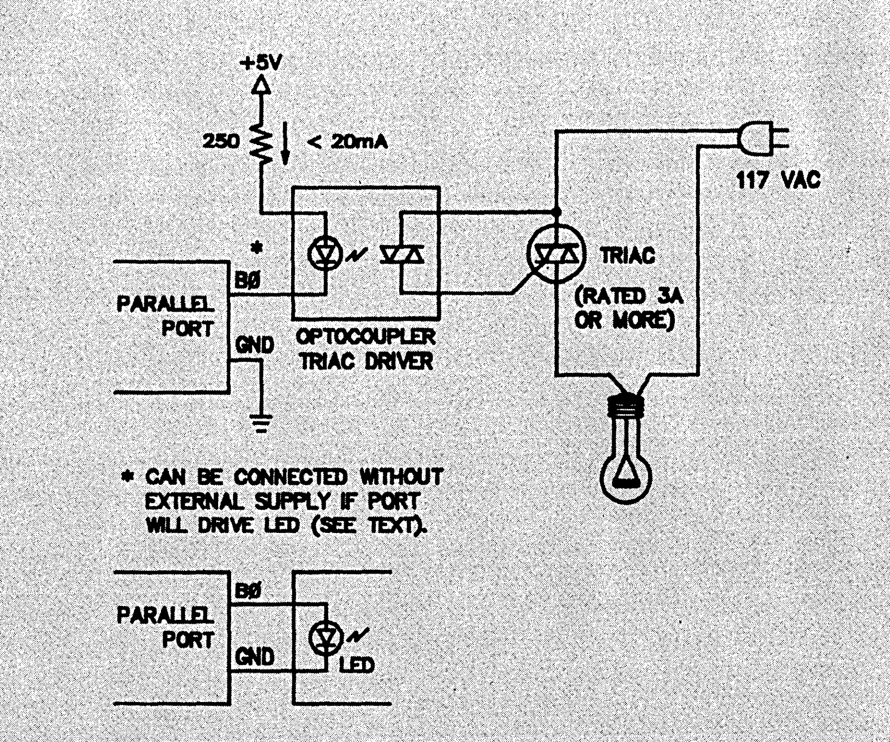
Semiconductors can be either positive (P) type, or negative (N) type. P type has a slight excess positive charge; N type has a slight excess negative charge. If you stick 'em together, the excess positive charge and the excess negative charge will combine near the junction and make a space where there isn't any excess charge (and thus no way for current to flow) - this is called the "depletion region." (See Figure 2).

If you hook a positive wire to the negative N side, and a negative wire to the positive P side, the P's combine with the N's on both sides, which makes the depletion region even larger. So, no way any current's gonna flow.

Reversing the situation, with the positive wire on P and the negative wire on N, the (negative) electrons will flow merrily through the N region, reestablish the N side of the depletion region as N territory, and flow on through (the depletion region goes away).

The P's on the other side (called "holes" by the powers that be, and positive because there's no electron where there oughta be) do the same thing. And current flows from negative to positive (real electron flow) or from positive to negative (hole flow - conventional cur-

Figure 1 — Controlling a TRIAC with a TTL output device.



By Bruce Eckel
EISYS Consulting
1009 North 36th Street
Seattle, WA 98103

rent direction, thanks to Ben Franklin).

The upshot is - connecting the negative wire to the positive side won't let any current flow, while connecting it to the negative side lets current through. This is a diode, which I've talked about before. The P side is the Anode (arrow side of the diode), and the N side is the Cathode (wall or bar side).

Transistors

Transistors use three slices of semiconductor material instead of the two in a diode (see Figure 3). This three-layer semiconductor sandwich can be made two ways: one slice of N between two slices of P, or one slice of P between two slices of N (thus the two kinds of transistors: PNP and NPN).

Transistor symbols look like permuted diode symbols - the bar is there, but the arrow has been shoved over to the side to make room for another wire. The two types of transistors are differentiated by the direction of the arrow. The arrow points in the direction of conventional (hole) current through the device. When the arrow's point is poking the bar, it's a PNP (PiNPoint), and when it isn't, it's an NPN (No PiNpoint).

The flow of current through a transistor is controlled by the middle slice of material: the base.

Let's say we take an NPN transistor and connect the collector (one N region) to plus and the emitter (the other N region) to minus. No current will flow.

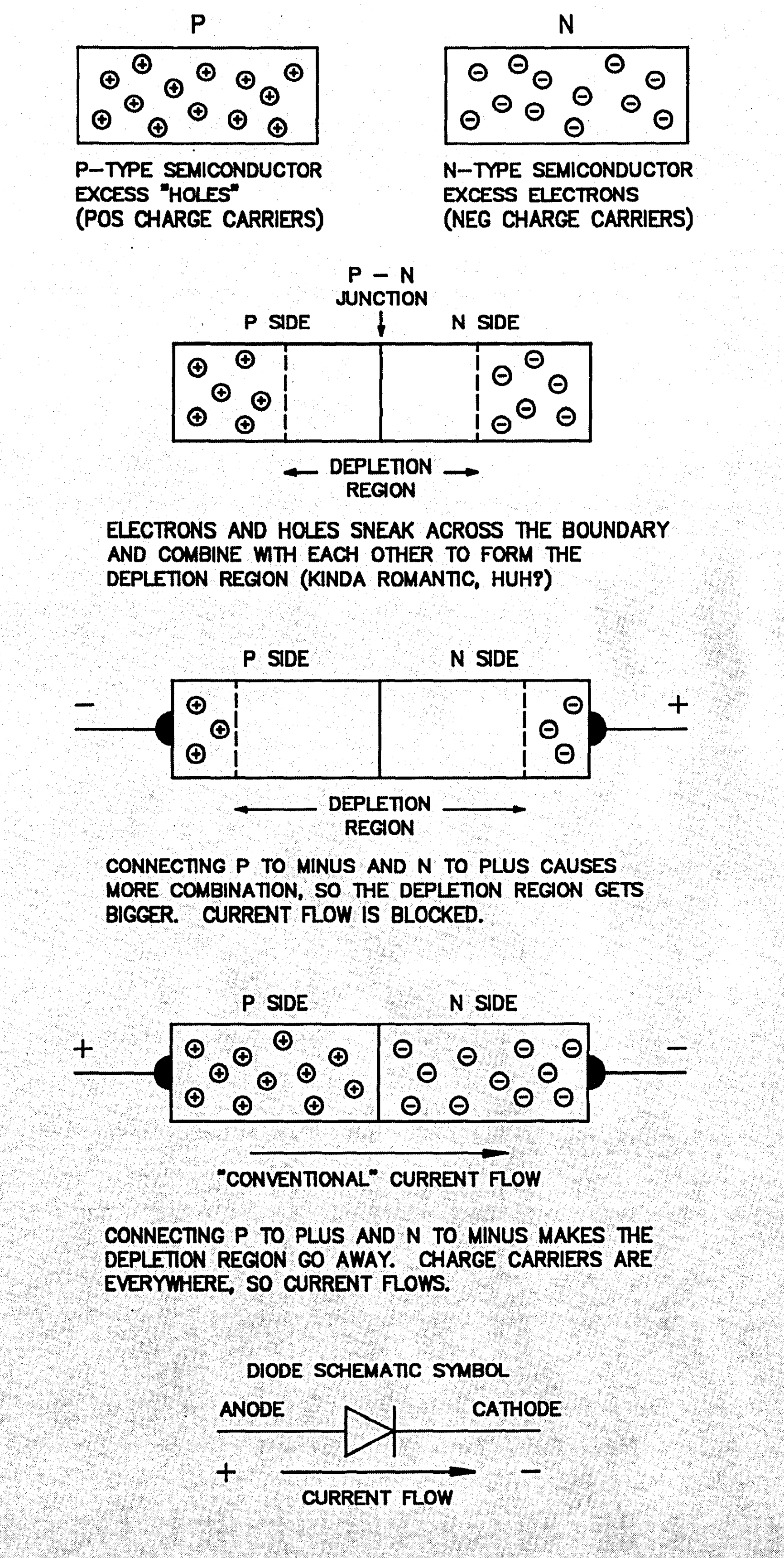
But if we also make the base (the P region in NPN) positive enough so a small amount of current will flow from the base (P) to the emitter (N), then (a much larger amount of) current can flow from the collector (N) to the emitter (N).

Reverse all the polarities for the PNP transistor.

Although transistors may be used as

(continued next page)

Figure 2 — Diode internals.



(continued from page 19)

amplifiers, as in your stereo, digital electronic circuits use them exclusively as switches. So the transistors are either turned all the way on or all the way off.

Silicon Controlled Rectifiers

Figure 4A shows how to make an SCR from two transistors. It's an NPN transistor and a PNP transistor connected together in sort of a funny way.

The NPN's collector (which sucks current into the NPN) is connected to the PNP's base, which, if it has current sucked from it, turns the PNP on. The PNP's collector, which squirts current out, is connected to the NPN's base, which, when it has current squirted into it, turns the NPN on. Once this party gets started, it just doesn't quit!

It can't get started, though, until we increase (briefly) the gate voltage. Until then, nothing happens. In addition, the SCR, like any diode, only conducts current one way. Figure 4B shows the schematic symbol for an SCR and what it physically looks like (another layer has been added to the sandwich).

SCRs are used in DC applications as "crowbars" to prevent damaging other devices in over-voltage situations. They're also the "thyristors" in your electronic flash. When the light sensor decides the flash tube has generated enough light, it turns off the SCR. Thus the remaining energy in the capacitor is not wasted and the flash recharges faster.

We use SCRs in AC applications to control motors and regulate AC voltage and current. Figure 4C shows an AC waveform being chopped by an SCR. When the voltage across the SCR goes negative, the device turns off until the voltage is positive and a new gate pulse appears.

SCRs are often used for light dimmers.

Triacs

A triac is a bidirectional SCR; it has an SCR going in each direction and the gates are connected. Figure 5 shows a triac and its schematic symbol, which has a diode arrow going in each direction. As long as the gate's turned on, AC current will flow. When the gate's turned off, the current will stop flowing the next time the AC wave goes through zero. Just like a wall switch!

Well, not quite. The switch can handle more current and a lot more punishment.

(continued on page 23)

Figure 3 — Transistor internals.

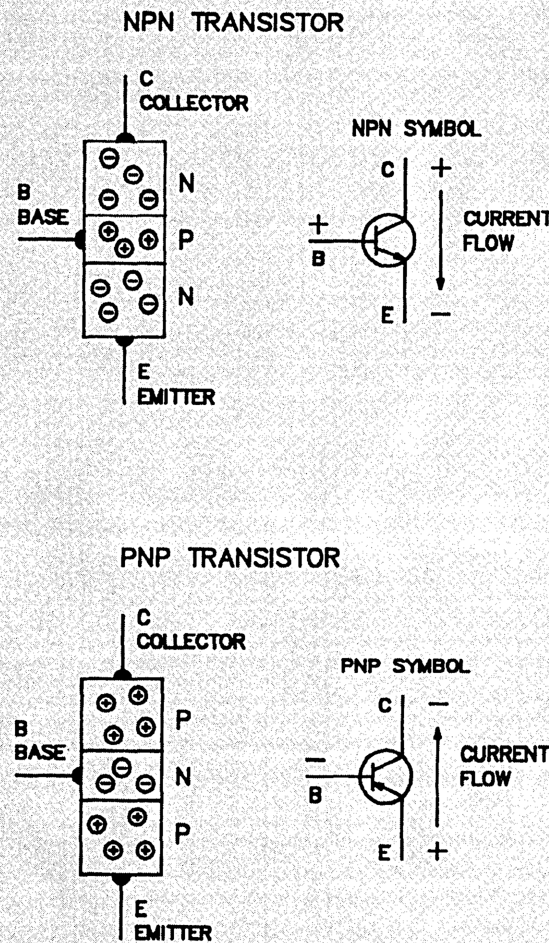


Figure 4A — Building an SCR out of two transistors.

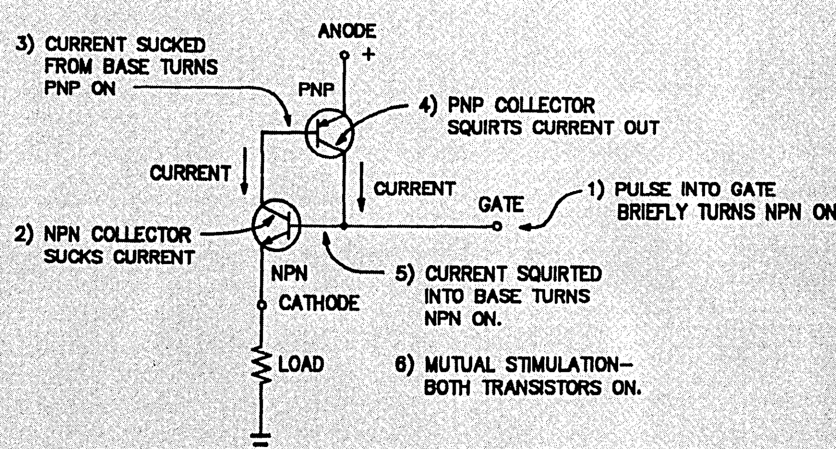


Figure 4B — Building an SCR out of silicon (and its schematic symbol).

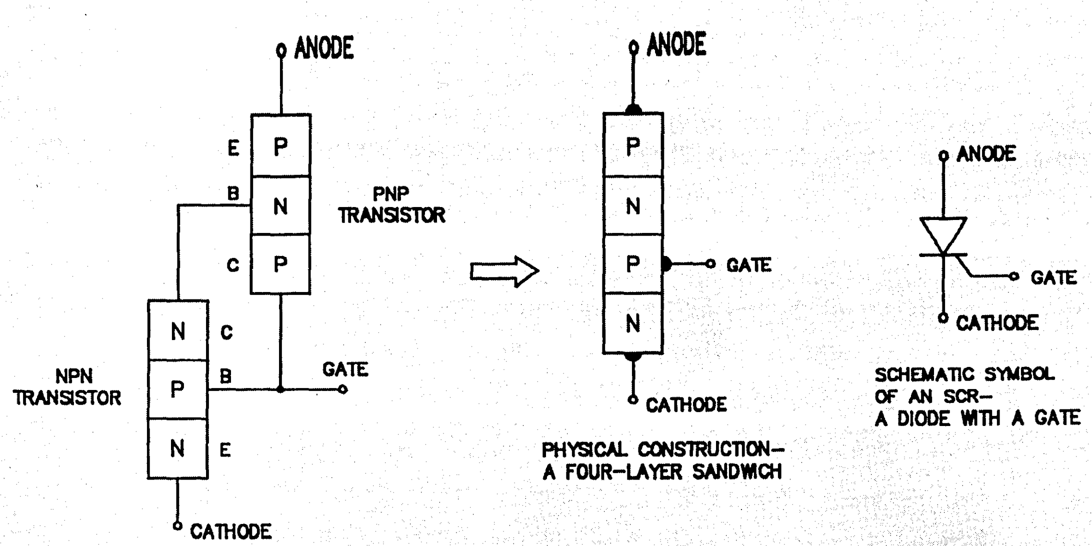


Figure 4C — What an SCR does to an alternating wave.

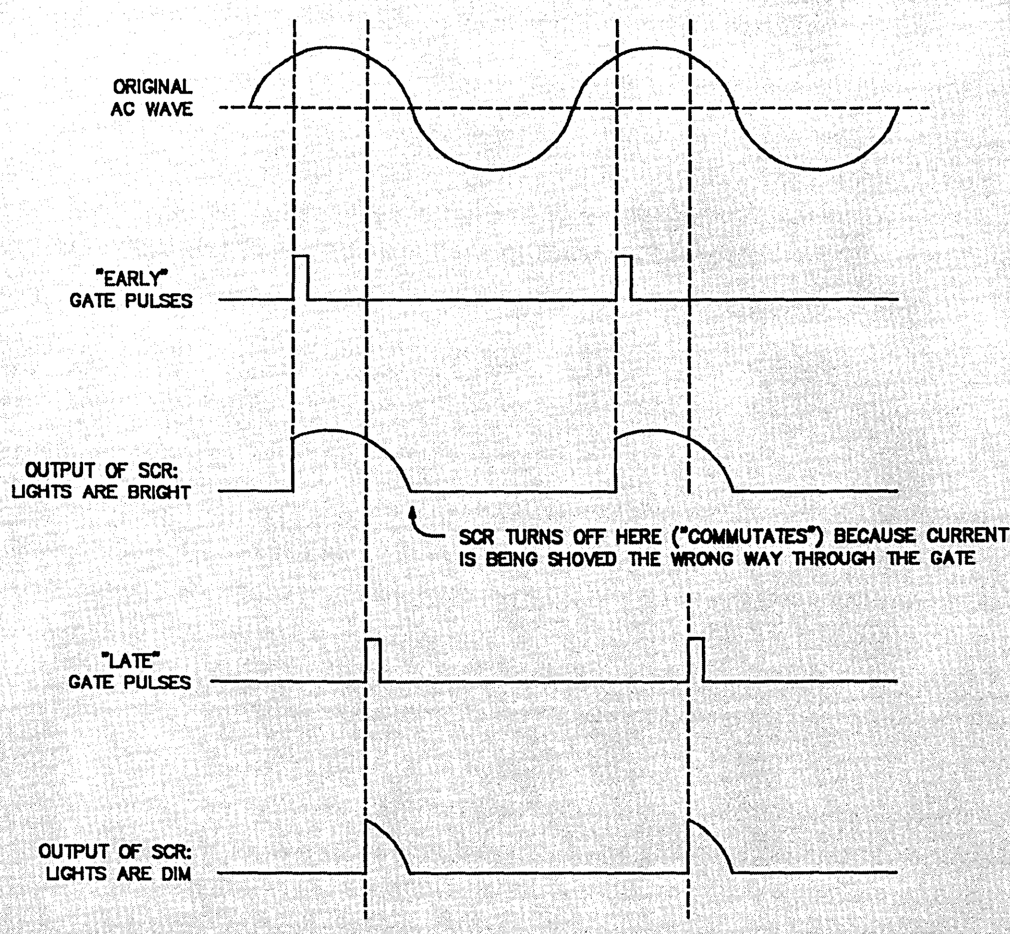


Figure 5 — Building a TRIAC out of two SCRs.

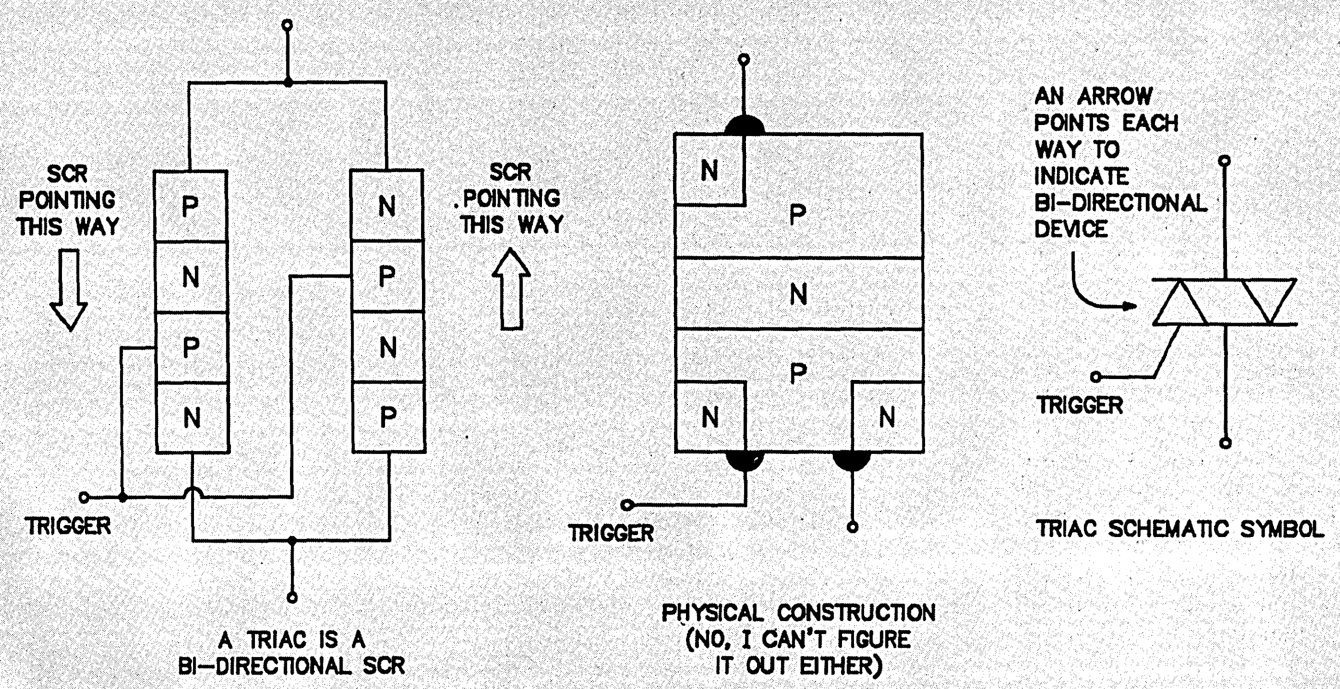
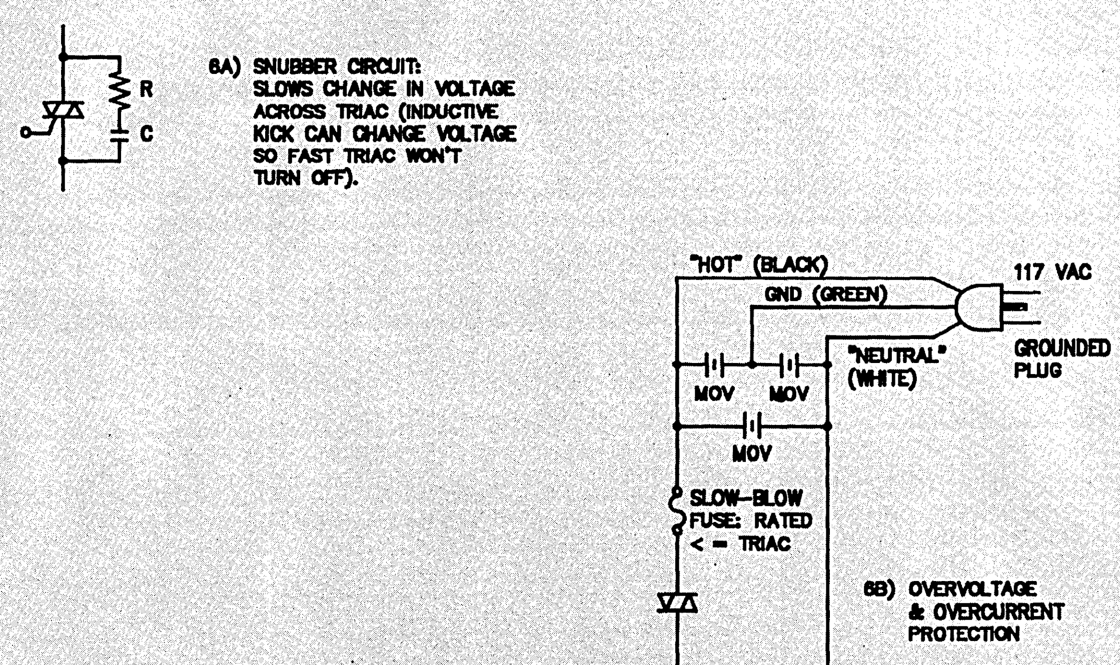


Figure 6 — Protecting the TRIAC from inductive surges and power line spikes.



(continued from page 20)

The triacs you get at Radio Shack will handle 6 Amps, which, when multiplied by 117VAC, means you can power a 700-Watt device! But before you go plugging huge loads into your triac, you should know things aren't what they seem.

A light bulb, for instance, might say "60 Watts," but that's in its steady state - after it's all warmed up and glowing. For a few milliseconds after you first turn it on, the filament is cold, and the resistance is quite low. (Generally, colder things have lower resistance, since the electrons aren't getting knocked about so much. Superconductors are cold enough so the electrons don't get knocked about at all.) Anyway, there's an initial surge of current when a light is first turned on.

Triacs are usually specified to withstand roughly ten times their average current rating for one cycle (with 60Hz wall current, this is 1/60 sec = 16.7 milliseconds). A light bulb heats up within one cycle, so this isn't a problem.

Inductive Loads

We call a motor, transformer, or anything else with a magnetic coil in it an "inductive load." An inductor stores electrical energy in its magnetic field. When you try to shut off the current to an inductive load, the energy in the magnetic field is converted back to electrical energy (as the magnetic field dissipates), and the inductor tries to force current through the circuit.

This "inductive kick" can cause large voltages across the device that's trying to break the circuit. This action is what causes a switch to spark when shutting off the power to a motor or transformer.

The inductive kick can prevent the triac from turning off (if it kicks really hard, the SCR's breakdown voltage will be exceeded, and the part will be damaged). To prevent this, we must keep the voltage across the device from rising too fast when we try to turn the inductive load off. This requires what's called a "snubber" circuit (to snub out the fire, I suppose...).

Figure 6A shows a snubber circuit (also known as a transient suppressor) for an inductive load. Selecting R(s) and C(s) isn't horribly difficult, but it requires calculus and fundamental circuit theory, which isn't really appropriate here. ("The Art Of Electronics," by Horowitz and Hill, pg. 44, suggests these typical values for small inductive loads: R(s) = 100 ohms and C(s) = 0.05 uF.)

Things are a lot simpler if you can avoid an inductive load.

Other Safety Measures

After putting all the effort into wiring this thing together, you might want to protect it in two more ways. A slow-blow fuse rated the same as your triac will protect it from being overloaded.

A metal-oxide varistor (MOV - conducts during a voltage spike; you should already have them to protect your computer equipment) will prevent damage from lightning and other voltage spikes. Figure 6B shows these devices in the circuit.

Backing Up

With all these caveats in place, let's assume you don't need any of the safety stuff (which, indeed, you may not) and just look at how the simple circuit (Figure 1) works.

Digital logic is usually better at sinking current (sucking it in) than supplying current to an external device. So we connect the digital side of the circuit to a 5-volt source through a "current-limiting" resistor (since an LED won't limit its own current), then through the LED, and finally into the parallel port.

For less hassle (no external power supply), you can try connecting the output line through the LED and resistor to the ground of the output port. It will probably provide enough current, but I make no guarantee.

When the port line is set to "0," it will pull the line low and current will flow through the LED (if you want more detail, see my articles in issues #32 and #33). The light from the LED will turn on the triac.

This triac is really just a triac driver; by itself it can only handle about 50 milliamps. A triac is the ideal driver for another triac, so these opto-isolated devices were created.

117VAC Wiring

I know you may be used to thinking of the black wire as ground, but the guys who invented house wiring have their own secret code: black is "hot," white is "neutral," and green is ground.

I grew up thinking both prongs of a wall socket were dangerous, so it came as quite a shock (pardon) to find out neutral and ground were connected together (usually back at the junction box to the house).

If you're running an electric dryer, the resistance in the neutral wire can cause quite a voltage on it in some points of the house, so it can't always be considered ground. (But it goes a long way towards explaining why I survived my childhood.)

The green wire on a three-wire plug connects to the hole in the middle of the socket, which should go directly to a solid ground. The frame of any appliance is supposed to be grounded. This way, if the hot wire in your toaster comes undone and contacts the frame, it doesn't just electrify it (the frame) and wait for you to come along and touch it. It has a direct path to ground through which it starts pouring current - quickly throwing the circuit breaker.

Always connect the green wire to the frame of whatever you're controlling, if you can, and use a fuse - otherwise your triac may blow before the circuit breaker does.

When my dad explained electricity to me, I got the impression that AC current pushes out of one slot and then pushes out the other (slot), and it just ain't so. (No fault of Dad's; he knew how things were wired up. Probably just the generation gap.)

Only one slot does the pushing and the pulling, and it's the short slot connected to the black wire. Test it with your voltmeter on the appropriate AC range if you don't believe me.

The Digital Part

I've covered the parallel ports on a Kaypro in previous articles, so you're on your own there. Since I now own a PC clone, I'll look at the hardware for that.

The cheapest digital output port is one you already own: the parallel printer port.

The original PC design allows for two of these, with I/O addresses at hex $0378 (LPT1) and $0278 (LPT2), depending on the setting of a DIP switch on the card. I suspect IBM is sorry it was so hasty with the design; it would be more useful to have an arbitrary number of ports with arbitrary locations established at boot time. (But CP/M machines were worse: they usually only had one port with an address based on the manufacturer's whim.)

You may observe you're already using your printer port. If you plan to add power control as a permanent fixture to your machine, it's time to buy another parallel printer card and configure it as LPT2. Not only are the cards ridiculously cheap (for example: $21 from Microsphere), but it may be pos-

(continued next page)

Figure 7 — Turbo Pascal routine to control AC via parallel port A or B.

```
type
    switch_positions = (OFF,ON);
    switch_rec = record
                   name : string[15];
                   x_location, y_location : integer; {screen coords}
                end;

const
     port_a = $220;  { Port locations in the PIO-12 8255 chip. }
     port_b = $221;  { Use $378 for LPT1 or $278 for LPT2.     }
     port_c = $222;  { The stupid compiler can't add constants.}
     control = $223; { To configure the 8255.                  }

     switches : array[0..7] of switch_rec = ( { easier than initializing }
       (name:'Doghouse'; x_location:1; y_location: 1 ),
       (name:'Floodlight 1'; x_location:40; y_location: 1 ),
       (name:'Floodlight 2'; x_location: 1 ; y_location: 6 ),
       (name:'Garage'; x_location:40; y_location: 6 ),
       (name:'Spotlight'; x_location: 1; y_location: 11 ),
       (name:'Radio'; x_location: 40 ; y_location: 11 ),
       (name:'Alarm'; x_location: 1 ; y_location: 16 ),
       (name:'Lock'; x_location: 40 ; y_location: 16 )
     );
     HOME_X = 14;  { Keep cursor here during program execution }
     HOME_Y = 20;

var
   switch_state : array [0..7] of switch_positions;
   i : integer;
   switch_byte : byte;  { value sent to the output port }
   CH : char;

procedure display_switch_state(switch : integer);
{ Show new switch state on monitor }
begin
  with switches[i] do begin
     gotoXY(x_location,y_location);
     write(switch,' : ',name);
     gotoXY(x_location + 20, y_location);
     if switch_state[switch] = on then write('ON ')
     else write('OFF');
  end;
end;

procedure invert_switch(switch : integer);
{ flip a switch from off to on or on to off }
begin
     { first flip the bit and output the result }
     switch_byte := switch_byte xor (1 shl switch);
     port[port_b] := switch_byte;  { see "Port Array" in the manual }
     { now invert the display information }
     if switch_state[switch] = on then
        switch_state[switch] := off else
        switch_state[switch] := on;
end;

begin
     { Initialize the 8255.  Not necessary for LPT1 or LPT2   }
     port[control] := $99;   { Port B is output, A & C inputs }
     { Initialize everything else }
     switch_byte := 0;
     ClrScr;
     for i := 0 to 7 do begin
         switch_state[i] := off;
         display_switch_state(i);
     end;
     GotoXY(HOME_X,HOME_Y);
     Write('Press number to change switch; ESC to quit');
     CH := ' ';
     while CH <> chr(27) do begin
           GotoXY(HOME_X -1,HOME_Y);
           read(KBD,CH);
           i := ord(CH) - ord('0');
           if i in [0..7] then begin
              invert_switch(i);
              display_switch_state(i);
           end;
     end;
end.
```

(continued from page 23)

sible to modify one to read from the outside world. Since I'm at least as fond of "I" as I am of "O," I'll check this out for next issue (in "The Adventures Of A Cheap Parallel Input Port").

Figure 7 is a Turbo Pascal program to turn any one of the eight bits of your parallel port on or off (to use it, change the "port_b" constant to $378 for LPT1 or $278 for LPT2).

To test it, you need -

- A male DB-25 connector and cable

It's probably easiest to get the kind of connector which clamps onto ribbon cable, and then use a vise to connect it. But you can also destroy an existing printer cable if it's cheaper.

There are tiny numbers on the connectors so you can figure out which pins go to which wires. The important pins are 2 through 9, which correspond to data bits 0 through 7, and pins 18-25, which are ground.

- 8 LEDs

Connect the pin coming out of the side with the "flat spot" on it (look at the LED, you'll see what I'm talking about) to a ground wire, and the other side to a data wire. I didn't use current-limiting resistors, and it worked fine. If you don't pop any LEDs and they glow brightly, then you don't need an external supply or resistors for the triac optocoupler.

Expensive Mode

You may be using both your printer ports. You may also be exceedingly uninterested in risking electrocution based on the assurances of some yahoo writing for a computer magazine. Fortunately, other companies have solved your problem for you. Unfortunately, they want money.

Metrabyte (440 Myles Standish Blvd., Taunton, MA 02780, (617) 880-3000) sells a 24-bit parallel I/O port (the PIO-12) for $115, which can be placed anywhere in the I/O map and performs input as well as output with no modifications.

They also sell a cable and a nice box which you can fill with eight solid-state I/O modules, which can control (or sense, using a different module) AC or DC (DC uses different modules). All you do is connect your wires to screw terminals in the box. The box, cable and modules cost an additional $278, bringing the total to almost $400 (which is next to nothing, if someone else is paying you for your time).

The PIO-12 card solves a lot of problems. But it does have some limitations: there are no built-in snubber circuits (the modules are specified for resistive loads), and it won't handle more than 3A. The worst feature: the system powers up with all the power ON. I prefer to have the power off until I tell it to go on, but I suppose there's no accounting for tastes.

If you need the convenience, and if it fits your constraints, the Metrabyte system seems like a pretty fair deal (a better deal is their relay board for $135 - it already has the relays on it, and they can handle 3 amps).

To use the PIO-12, you must find an address in the I/O space which isn't already used, and set the DIP switches on the board to that address (the DIP switches are inverted: "on" means 0 and "off" means 1). Then, you simply write and read to offsets of that address. Figure 7 demonstrates the use of the PIO-12.

Finding a free I/O space is a bit more of an adventure. You need four contiguous I/O locations.

The motherboard uses locations 00 - FF (all numbers in hex). Locations 100 - 3FF are for the slots, but they're peppered with reserved locations, including the serial and parallel ports, the monochrome, color and EGA boards, and a number of odd things (you must also insure you don't collide with another "wild card" board).

There's a nice chunk of free space from 220 to 277, so I put the board at 220 (consult the IBM technical reference manual for the full map).

Send Me Your Problems

Send me a description of the problem you need to solve. I can't promise to design something for you, but you'll keep me in touch with the real-world problems readers are trying to cope with (personal problems will be redirected to Dr. Ruth).

DMA Control On The PC

Smart IC Is Dedicated To Speed

Larry's plunging deeper inside the PC than ever before. This time he dissects the DMA controller in excruciating detail. But it's worth it because at the end he discovers a simple system speedup that would, no doubt, leave RAM designers tearing their hair. (We've got to get to the root of this.)

One of the strengths of the Personal Clone lies in its use of smart peripheral chips to aid and abet the processor. The chip I'll talk about this time around takes control of the system address, data, and control busses to direct DMA transfers. So the processor just hangs out until the DMA Controller finishes up.

But there is a very real increase in system speed because the DMA Controller is designed for a specific task while the processor has to be a general purpose device. The DMA Controller deals only with transferring data: from memory to memory, from memory to I/O, and from I/O to memory.

What's DMA?

DMA stands for Direct Memory Access. This technique allows a DMA Controller to set up direct transfers of data without the intervention of the processor. The processor only needs to program the DMA controller with the answers to the following questions:

- How many bytes will be transferred?
- In what direction will data flow between the I/O device and memory?
- At what memory location should the read or write of data begin?

So you've called up the latest 10 MP (mega-point) version of ADVENTURE from your hard disk. What happens as the bytes come chugging in from that metal mother? With the processor doing all the work, the sequence goes something like this.

The processor spins its wheels waiting for a byte to show up from the hard disk. When the byte appears, an IN instruction loads it into the AL register. Next, the contents of AL get written to a memory location pointed to by some pointer. After incrementing the pointer, the processor is again ready for another byte.

All this takes a lot of time. On a hungry day, the processor takes at least 29 clock cycles to grab a byte. Luckily the PC gives the processor help with its memory concerns.

The 8237

Intel's 8237 Programmable DMA Controller handles the DMA chores in the average garden variety PC. It does a good job, too. In the example above, once the 8237 has been programmed, it takes only five clock cycles to collect and deposit an incoming byte.

The 8237 has four independent channels to handle DMA transfers. It supports transfers between two memory locations as well as those between memory and I/O, although this capability isn't implemented on the PC. Channels 0 and 1 are required for memory-memory transfers. But, in the PC, channel 0 has its hands full doing memory refresh. Hence, no fooling around with memory-memory transfers. We'll get back to memory refresh in a minute.

Channels 1 - 3 allow peripheral devices to request DMA service through the PC's expansion bus. Channel 1 is undedicated, channel 2 manages floppy disk I/O, and channel 3 talks to hard disk controllers.

The 8237 can assume one of two operational cycles. In its Idling Cycle, it does just that. Intel calls this SI (State Inactive). This is the condition where the 8237 accepts programming or waits for some action.

Six other states make up the Active Cycle. The first state, S0, occurs when the 8237 has asked the 8088 for a hold. The 8237 remains in S0 until the 8088 grants the hold. Then we're into the 8237's working states, S1 - S4. Finally, the 8237 can insert wait states (SW) between S3 and S4 to accommodate slow memory and I/O devices.

Down And Dirty

Let's look at some of the more important pins on the 8237. First there's the Chip Select (CS) input. (See Figure 1.) During the 8237's Idle Cycle, a low on CS selects the 8237 as an I/O device. This allows the 8088 to either read or write the 8237's control registers. CS comes from an LS138 3-to-8 decoder. Any port read or write in the range of 00h to 0Fh will select the 8237.

I/O Read (IOR) and I/O Write (IOW) work in conjunction with CS to tell the 8237 whether it's being programmed or read by the 8088. Both are generated in the bus controller by decoding the processor's three status lines. See Micro C issue #36, pg. 38, for a complete discussion of how the CS, IOR, and IOW signals are generated.

I'll save the programming details for later. The important thing here is that we have to tell the 8237 how many bytes will be transferred and where their starting address lies in memory. Then, after each byte gets transferred, the 8237 can increment an internal register (Current Address Register) to point at the next memory location. It also decrements the byte count (Current Word Count Register) until it reaches 0, ending the transfer.

The RDY line controls wait states. The 8237 samples RDY during S3. If RDY is low, then wait states are inserted between S3 and S4 until RDY once again goes high.

Requests for DMA service come in on DREQ0 - DREQ3. The DREQ lines can be either active high or active low depending on how the 8237 has been programmed. When the 8237 sees activity on an unmasked DREQ line, it

By Larry Fogg
Micro C Staff

sends a Hold Request (HRQ) to the 8088. HRQ is a request for the 8088 to relinquish control of the system busses.

When it's ready to relax, the 8088 signals availability of the busses with a high on Hold Acknowledge (HLDA). At this point, the 8237 takes over the busses and begins the DMA transfer.

Here comes the heart of DMA. Roughly, the DMA transfer goes like this:

The 8237 places an address on the address bus and sends a DMA acknowledge (DACK) to the device requesting the transfer.

DACK acts as a chip select for that device so that it can either place its data on the data bus or read data from the bus.

Finally, the 8237 generates either a memory read (MEMR) or memory write (MEMW) signal and the data goes directly between the memory location and the device.

The DMA Controller deals only with transferring data: from memory to memory, from memory to I/O, and from I/O to memory.

Addressing Concerns

Let's go through all that in detail. The data bus lines (DB0 - DB7) on the 8237 are bidirectional lines which can hold 8 bits of either data or address. During memory-I/O transfers (remember, these are the only kind on the PC), they supply the most significant address byte (MSB) to an LS373 octal latch. The 8237's address strobe (ADSTB) is the signal which tells the LS373 to grab the address byte.

The latch will hold onto that portion of the address until it sees the DMA Address Enable (DMA AEN) signal on its Output Enable (OE) pin. The MSB is then released onto the address bus. The 8237 generates an address enable of its own (AEN), but the PC leaves this pin disconnected in favor of DMA AEN.

When ADSTB signals the loading of the MSB of address, two other things happen more or less simultaneously. First, the least significant byte (LSB) of the address comes out on the 8237's A0 - A7 pins. The 8237 continues to generate the LSB until the next DMA transfer occurs. The LSB gets buffered through an LS244 octal buffer. Again, DMA AEN performs the output control for the LS244.

Second, the 8237 drives its memory read (MEMR) and memory write (MEMW) lines high in preparation for the transfer. MEMR and MEMW are three state, active low outputs which signal to the system whether we're doing a gazzinta or gazzouta operation. Up until now they've been in their third, or disconnected (high impedance) state.

We need a third state on these outputs because the 8237 isn't the only device talking on the MEMR and MEMW lines. An LS243 gives the 8088 access to them as well. The DMA AEN signal controls the output of the LS243. A low on DMA AEN disconnects the LS243 so the 8237 can talk. A high lets MEMR and MEMW signals from the 8088 pass through the LS243. DMA AEN can't go low unless the 8088 has given up control to the 8237. So we don't have any possibility of contention for MEMR and MEMW.

Next in the sequence of signals, the 8237 sends a DACK out to the requesting device. This selects the device, which in turn puts its data onto or begins reading the data bus. The only remaining task for the 8237 is to pull either MEMR or MEMW low to perform the actual data transfer.

After the transfer the 8237 increments its Current Address Register and decrements its Current Word Count Register. If DREQ is still active and the Word Count isn't 0, the 8237 spits out another HRQ to initiate the process again.

The 64K Limit

We've only talked about the 16 bits (64K) of address that the 8237 can generate. That's obviously not enough. We need another 4 bits of address to cover the PC's full memory space.

These 4 bits come from an LS670 4X4 register file. Think of the LS670 as a latch which holds four separate 4-bit words. Each word corresponds to one of the four DMA channels and provides that channel with its highest order 4 bits of address. We'll refer to the words as DMA Page Registers.

The Page Registers are loaded via three I/O ports. Yeah, I know there are four ports, but we really need only three since DMA channel 0 does the memory refresh and doesn't need a full 20-bit address.

The LS138 3-to-8 decoder responds to any write to a port address between 80h and 9Fh by generating a Write To DMA Page Register (WRDMAPGREG) signal. WRDMAPGREG enables the LS670 to receive data.

While the lower order 5 bits of the port address have no significance to the LS138, bits 0 and 1 mean a lot to the LS670. They determine which of the

(continued on page 29)

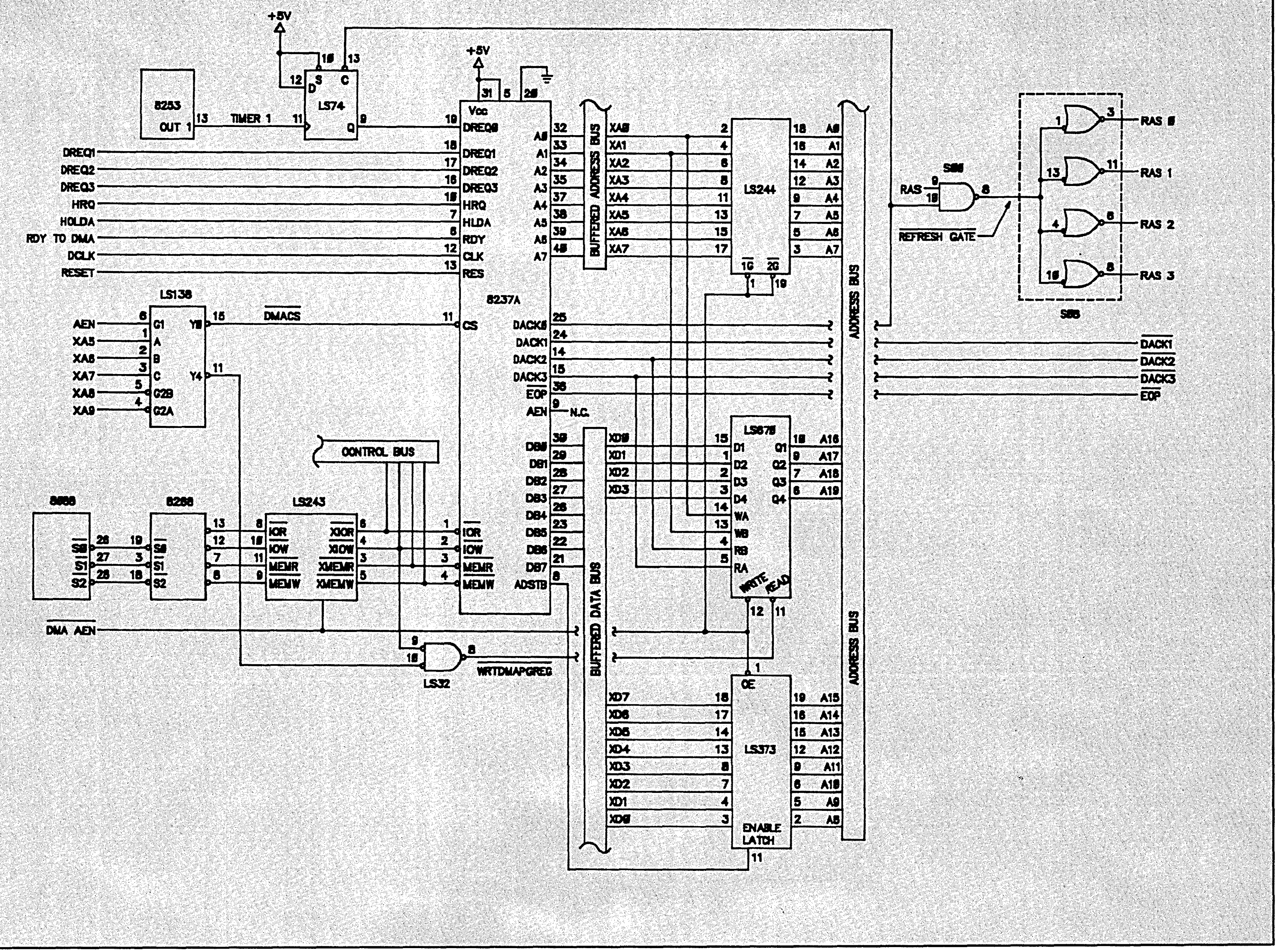


Figure 1 — DMA controller circuit in IBM XT.

(continued from page 27)

DMA Page Registers will be selected for writing. No one's looking at bits 2 through 4, so they could be any value. But we'll let them be 0. The port addresses are listed in Figure 2.

So the four least significant bits of the data byte written to the Page Register get sucked in by the LS670. Later, when a full 20-bit address is needed, a DACK2 or a DACK3 selects either Page Register 2 or 3 to provide the extra address information to the address bus.

Loading the Page Register contributes almost nothing to overhead. It shouldn't have to be done more often than once for every 64K bytes moved during a given transfer. Now we have a valid 20-bit address stored in the LS244, LS373, and LS670. All the chips need is a nod from DMA AEN to dump their contents onto the address bus.

One further question has to do with the timing of DMA AEN. Obviously we need this signal before the MEMR or MEMW signal goes active. Otherwise the three address components will not have been released onto the bus. But, lacking a logic analyzer, I can't tell you exactly when DMA AEN occurs in the sequence of events.

The Rest Of The Pins

The 8237's RES input looks at (strangely enough) the system RESET signal. A high on RES clears all internal registers with the exception of the Mask Register. All channels get masked on RESET. The 8237 then goes into its Idle Cycle.

The End Of Process (EOP) pin deals with termination of a transfer. This active low, bidirectional line either sends EOP information to the device involved in the transfer when the Current Word Count reaches 0, or accepts an early termination signal from the device.

CLK accepts 3 MHz in the 8237 and 5 MHz in the 8237-5. Of course, these are nominal ratings. The 8237-5 in my Clone digests 7.37 MHz quite happily, thank you.

About the mysterious pin 5 Intel will only say, "It should always be at a logic high level." So it's tied high.

Modes

The 8237 has its choice of several operational modes. When I outlined a DMA transfer earlier, I said that each byte was transferred individually in response to an ongoing DREQ. Call this mode the Single Transfer Mode. The PC uses it for both memory refresh and floppy I/O. (I'm not sure which mode hard disk I/O uses.)

In Single Transfer Mode the 8237 doesn't care if DREQ is held active throughout the entire series of transfers. HRQ will always go inactive and release the busses between bytes anyway. The 8237 then issues another HRQ in response to the still active DREQ. So in effect we have separate DREQs for each byte transferred.

In Block Transfer Mode a single DREQ causes transfers to take place until the byte count reaches 0 or an EOP comes in from the I/O device. The DREQ only has to be asserted until the 8237 sends a DACK. Demand Transfer Mode is pretty much the same except that DREQ must remain active for the transfer to continue.

Cascade Mode gives you the option of building any number of extra levels of DMA service by attaching additional 8237 slaves to a master. Each master DREQ - DACK pair connects to a slave's HRQ - HLDA pair, yielding four new DMA channels.

Figure 2 — Port addresses for DMA registers.

DMA Ports

Name	Address (hex)	Read/Write
Ch 0 Base Address	00	W
Ch 0 Base Word Count	01	W
Ch 1 Base Address	02	unused
Ch 1 Base Word Count	03	unused
Ch 2 Base Address	04	W
Ch 2 Base Word Count	05	W
Ch 3 Base Address	06	W
Ch 3 Base Word Count	07	W
Status / Command	08	R/W
Request	09	unused
Mask	0A	W
Mode	0B	W
Clear LSB/MSB Flip-Flop	0C	W
Master Clear	0D	W
Clear Mask	0E	W
Multiple Mask	0F	W
DMA Page 0	80	unused
DMA Page 1	81	W
DMA Page 2	82	W
DMA Page 3	83	W

A Plethora Of Registers

Twenty-seven registers containing a total of 344 bits live in the 8237. The first of the twelve types is the Current Address Register. Each of the four DMA channels has one of these 16-bit registers. When used in conjunction with the DMA Page Registers, these guys point to the memory address currently being read or written. The 16-bit Current Word Count Registers keep track of how many bytes remain in each transfer.

Next we have a 16-bit Base Address Register for each channel. They specify initial memory addresses for DMA transfers. The four Base Word Count Registers (16-bit) contain the total number of bytes to be transferred for each channel. These two types of Base Register provide the values used for initialization of the Current Registers during Autoinitialization. (I'll explain Autoinitialization later.)

Any combination of DMA channels can be masked, or disabled, by use of the 4-bit Mask Register. Each bit corresponds to one of the channels. A 1 bit disables DMA requests from that channel.

The four 6-bit Mode Registers allow programming of the various modes for each channel, and the 8-bit Command Register controls the overall performance of the 8237. The last useful register is the Status Register. It holds information on pending DMA requests and the count status for each channel.

Several 8237 registers go unused in the PC. A 4-bit Request Register allows for software initiated DMA requests. No go on the PC. We already know that memory-memory transfers won't work, so the three Temporary Registers dedicated to those transfers get ignored, too.

Programming The Command Register

I won't be rigorous in describing the

(continued next page)

(continued from page 29)

programming of the 8237 since there's not all that much that normal humans will want to do with it. This is the province of those strange beings who write operating systems and hard disk controller code. Besides, too much rigor leads to rigor mortis. For more information see Intel's *Component Data Catalog.*

Commands written to the Command Register effect all four DMA channels. Bit 0 enables memory-memory transfers. We don't want them so we'll reset bit 0 to 0. Bit 1 can be programmed to force channel 0 to hold the same address for all transfers. Quite useful for filling a portion of memory with a constant value. But again, we don't do memory-memory transfers so bit 1 gets a 0, too.

Bit 2 disables the 8237. Nasty things can happen if a DREQ goes active in the middle of programming. So disable the 8237 before any programming. For normal operation, make bit 2 a 0.

Compressed timing can be set with bit 3. "Really fast" systems use compressed timing to drop the time per transfer to 2 clock cycles. My Holliston board at work must not rate since it immediately puked when I tried this option. (I'm just kidding. I doubt you'll find a board alive that can use compressed timing.) So leave this bit at 0.

Like the 8259A Interrupt Controller, the 8237 prioritizes its incoming requests. The default gives DREQ0 the highest priority and DREQ3 the lowest. Bit 4 of the Command Register sets up a rotating priority situation. In this scheme, the last channel to receive service rotates to the lowest priority. It will then have to wait no more than 3 DMA services before it again sits on top of the priority heap.

There's an important difference in the two ways that the 8237 and the 8259A handle service requests. The 8259A will let a service routine be interrupted by a higher priority interrupt. Not so with the 8237. Once a DMA transfer begins, it must be completed before another DREQ is honored. Anyway, the memory refresh carried out by channel 0 will always be the most important DMA request. So we'll keep the fixed priority by putting a 0 in bit 4.

Bit 5 extends the duration of the MEMW pulse from the 8237. This would only be necessary if we were using compressed timing, so bit 5 is 0.

The two high order bits set the sense of the DREQ and DACK lines. We want DREQ to be active high (bit 6 = 0) and DACK to be active low (bit 7 = 0).

Programming The Mode Register

You can program specific characteristics into each DMA channel with the Mode Register. Bits 1 and 0 identify the channel being programmed (11b = channel 3, 10b = channel 2, etc.).

Bits 3 and 2 specify the type of transfer. Transfer of data from memory to an I/O device constitutes a Read Transfer. Program this direction with a 10b. Transfer of data in the other direction is called a Write Transfer. A 01b tells the 8237 to do a Write Transfer. The third type of transfer has the 8237 shooting blanks. It goes through all the work of the transfer without letting anything out on the memory and I/O control lines. This is a Verify Transfer (00b).

Autoinitialization makes it easy to repeat DMA processes with the same initial parameters. A channel set to Autoinitialize automatically re-initializes its Current Address and Current Word Count Registers when DMA service has been completed. Values for these registers come from the Base Address and Base Word Count Registers. A 1 in bit 4 enables this function.

I've talked in terms of incrementing through the address space during DMA service. The PC treats transfers this way, but the 8237 can either increment (bit 5 = 0) or decrement (bit 5 = 1) the Current Address Register after each transfer.

Bits 7 and 6 determine the transfer mode (00b for Demand Mode, 01b for Single Mode, 10b for Block Mode, and 11b for Cascade Mode).

The PC BIOS initializes DMA channels 1 - 3 for Verify Transfers, which means they don't do anything. The hard disk and floppy code have the responsibility of programming their respective channels.

Channel 0 gets set to 58h or 01011000b. That's channel 0, Read Transfer, Autoinitialization enabled, address incrementing, and Single Transfer Mode. Just the ticket for setting up ...

Memory Refresh

On to a practical example. DMA channel 0 receives a DREQ0 from the 8253 timer chip every 15 microseconds. This triggers the 8237 to perform a vital system task - memory refresh. Let's take a close look since refresh is so important.

Dynamic RAM chips store their bits of data in tiny capacitors. The small size of the capacitors leads to high density, high capacity, small package DRAMs. However, the smaller the capacitor, the fewer electrons (or holes) there are to determine whether the bit is a 1 or 0. (And the faster they begin wandering off.) If not recharged (refreshed) every 2 milliseconds, the charge bleeds away and we've lost data.

Every time we read a memory location holding a 1 (charge), a feedback loop recharges the capacitor. So a read of each location once every 2 milliseconds will take care of refresh.

The PC's memory space consists of four banks of 256K each. Think of the banks as 512 X 512 bit arrays. If you only have 256K of memory, this would be four banks of 64K organized in 256 X 256 bit arrays. Each location in the array has a specific row and column address.

During a normal memory read, the system generates one of four Row Address Strobes (RAS0 - RAS3), depending on which bank is accessed. This strobe latches bits 0 - 8 of memory address into a row address buffer within the RAM. While RAS is still active, the corresponding Column Address Strobe (CAS0 - CAS3) latches bits 9 - 17 of address into the RAM's column address buffer. Now

Figure 3 — DEBUG session for altering RAM refresh rate.

```
C:>DEBUG                    ; invoke DEBUG
-A                          ; call the assembler
????:0100  MOV AL,74        ; configure 8253 (see Micro C #35, p.30)
????:0102  OUT 43,AL
????:0104  MOV AL,40        ; LSB for counter 1 ...
????:0106  OUT 41,AL        ; ... change to alter refresh rate
????:0108  MOV AL,00        ; MSB for counter 1
????:010A  OUT 41,AL
????:010C  INT 20           ; back to DOS
????:010E                   ; <cr> exits assembler
-NREFRESH.COM               ; name the file
-RCX                        ; file size stored in CX
CX 0000
:E                          ; file is E bytes long
-W                          ; write file to disk
Writing 000E bytes
-Q                          ; quit DEBUG
```

the system can get at the memory location (within the selected bank), which is specified by the row and column in the buffers.

A dedicated refresh read is easier. After DACK0 goes active, the next cycle of RAS generates a REFRESH GATE signal (see Figure 1) at an LS00. An LS08 uses this signal to create a RAS for each bank. All four banks are now selected.

Let's assume we don't know what address the 8237 has put on the address bus in response to DREQ0. Whatever it is, the RAS signals have now strobed it into each of the four memory banks. The RAM sees no CAS signals during a refresh cycle. This RAS-only cycle causes all columns within the selected row to be refreshed (read) simultaneously.

Since we've selected four rows, each RAS-only cycle refreshes 2K bits (that's four 512-bit rows). The 8237 bumps its Current Address Register and 15 microseconds later, when another DREQ0 comes in from the 8253, another gang of four rows gets refreshed.

You can see that all it takes is the RAS-only read of ANY 512 successive memory locations to refresh all of memory. That's because the 512 possible row addresses are determined by the least significant 9 bits of system address. And all possible values for those 9 bits will be exercised within any contiguous block of 512 addresses. Pretty slick.

That's why we don't care about the DMA Page Register for channel 0. It has no significance. That's also why we don't care what value ends up in channel 0's Base Address Register. It just doesn't matter where we are in memory as long as we do 512 successive addresses.

The BIOS initializes channel 0's Base Word Count Register to 64K. The refresh process would have been just as happy with a value of 512. But then the 8237 would have to Autoinitialize the Current Word Count Register every 512 refresh cycles. Autoinitialization once every 64K refreshes makes life easier.

A final note on memory refresh. We know it takes 512 DREQ0s to refresh all of memory and that the DREQ0s happen every 15 microseconds. Let's see, that's ... wait a minute. It comes out to over 7 milliseconds. Those little charges were only supposed to last 2 milliseconds.

Quick And Dirty Speedup

I asked Dave what he thought about my long-term memory RAM. He knew of some folks who had slowed refresh all the way down to once every second before failure. I like to see my machine suffer, so I thought I'd see how far it would go before giving up. I had other motives as well. If my system spent less time on memory refresh, it would have more time for computation.

You can easily experiment on your own. Use DEBUG to change the frequency of the 8253 timer that drives DREQ0 (see Figure 3). I guess I'll have to start believing Dave. It was right at one refresh every second that my RAM faded away. A count of 900h loaded into the 8253's Timer 1 gives the one second period.

The BIOS loads the count with 12h and the point of diminishing returns comes around 30h or 40h. After 40h refresh happens so infrequently as to have no effect on system performance. Benchmarks show no difference between 40h and 900h.

A value of 40h gives a 2.6% speed increase on my Holliston 186 board and a 5.8% increase on a standard 4.77 MHz 8088. I haven't had any problems using this speedup. But don't blame me if your machine dissolves into a puddle of blue slime. After all, we've gone beyond the limits of computer decency.

As The Sun Sinks Slowly ...

DMA can be a fairly confusing subject. I found lots of conflicting information in my search for The Truth. The horse's mouth (IBM's *Technical Reference*) remains the best, if most obscure source of information on the PC's innards. If you really want to do some digging, that's the end of the horse to go to. Of course, you'll find a great many books at the other end, too.

Let me pass on another good source. I used *COMPUTE's Mapping the IBM PC and PCjr* by Russ Davies for the first time in researching this article. He does a great job describing the memory and port maps. He also gives what looks to be a very complete treatment of video.

Next time around I'll look into the PC's parallel interface chip, the 8255A. So, so long and thanks for the memories.

■■■

The Lotus/Intel/Microsoft EMS:
A Programmer's Guide To Expanded Memory

In a world where standards are (at best) more or less acceptable, three of the big players joined, in 1985, to establish another - the Lotus/Intel/Microsoft Expanded Memory Standard (or LIM-EMS, or EMS, for short). But before you could shake a big stick at it, AST and Quadram more-or-lessed the standard with THEIR version: the Enhanced Expanded Memory Specification. EEMS has bigger pages, more pointers, and runs EMS programs, although EMS won't run EEMS programs.

But that's another story. First, let Earl, programmer extraordinaire at PC Tech, give you the cook's tour of "the standard" - EMS.

The Intel 8088, 8086, and 80186 microprocessors can address 1 megabyte of memory, but not all of it is available to programs. Some is reserved for video memory, ROM BIOS, and expansion cards. So on the PCs, XTs, and clones, only 640K is general purpose RAM.

The Intel 80286 can address up to 16 megabytes in protected mode, but only 1 megabyte in real address mode. MS-DOS expects the 80286 to be in real address mode, so consequently, 640K is also the general purpose memory limit on most 80286s.

Software writers can access 640K easily (although that depends a bit on the writer), since every byte has a unique address, beginning at 0 and increasing by ones. Going beyond the 640K limit is a bit more complicated, but there is a way - by bank switching.

To use bank switched memory, the software writer and hardware developer must agree on at least three specs:

(1) The output ports (one or more) which will handle the bank switching,

(2) The size of the banks, and

(3) The addresses where the software will see the banks.

EMS answers the questions software writers and hardware developers would normally ask each other. Now they don't have to talk at all - which is how they like it.

Bank Switching

Bank switched memory allows a single CPU memory address to access multiple memory locations.

The switching mechanism selects blocks of memory, which (in theory) could vary in size from one byte to the full CPU address space. But neither of these sizes is ideal. With small blocks, the switching overhead kills program performance; and with large blocks, the memory layout isn't flexible.

Which CPU addresses should we use for the bank switched memory? Again, in theory, we could use any address. But in practice, we must switch memory CAREFULLY, making certain the executing part of a program, the stack, the interrupt vectors, and the interrupt service routines don't get switched. So, it's best to locate the bank switched memory where it's not likely to contain these crucial program elements.

We need to consider:

- What size blocks to switch.
- How many blocks to have visible at one time.
- Where the CPU will see the visible blocks.

Whiz Quiz (With Answers)

Let's say we want to transfer a block of data from one area of memory to another. And (just to make it interesting), let's add that we want to do it in a reasonable way, meaning without an unreasonable number of bank switches.

So, we'll need to see all, or at least a substantial part, of the source and destination blocks. And we won't want to worry about suddenly mapping our program, stack, operating system, or interrupt structure out of view. Therefore:

- What size blocks should be switched? Answer: between 1 and 256K.

The blocks should be large enough so that each "switch" switches a useful amount of data - at least 1K. In addition, the switched block must be limited to data: NO PROGRAM CODE. So the block size should be small compared to the total address space of the CPU. Thus, 256K is a reasonable upper limit.

- How many of these blocks can be visible at one time? Answer: 2 to 10.

The total amount of CPU address space dedicated to the bank switched memory should be small. Since this total is the number of blocks times the block size, the number of blocks should also be small.

- Where does the CPU see the visible blocks? Answer: between 640K and 1M, with caution.

We want to avoid switching a running program, DOS, interrupt vectors, or the stack out of the CPU's visible address space (between 0 to 640K). But the area above 640K isn't entirely safe. Video memory and interrupt service routines are located in the 640 - 1M range.

How does LIM-EMS answer these questions?

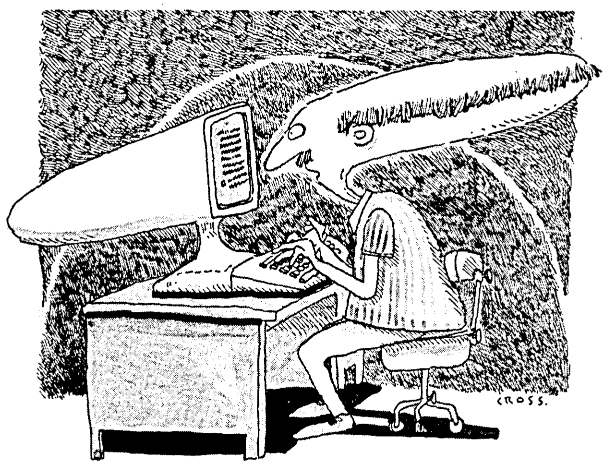
- What size blocks should be switched? 16K.
- How many of these blocks can be visible at one time? 4.
- Where does the CPU see the visible blocks? It depends on what other hardware is installed - usually segment A000H (640K) or D0000H (896K).

The LIM-EMS choices aren't the only correct ones. And some will argue that the LIM-EMS choices aren't even the best ones. But even if they aren't the best, they're still reasonable. What's important is that EMS hardware and software no longer have to worry about these choices. The hardware designers and software writers can go directly to the more important issues.

The Expanded Memory Manager

The expanded memory manager

By Earl Hinrichs
PC Tech
P.O. Box 128
Lake City, MN 55041



(EMM) is like a little operating system for expanded memory. The EMM is the interface between hardware and software. Software calls on the EMM to perform a task. The EMM runs the hardware and makes things happen. The EMM needs to know how the hardware works, but software that calls the EMM doesn't.

The expanded memory manager is an MS-DOS device driver, loaded by a "device =" line in CONFIG.SYS.

Before we list the EMM functions, let's pretend we don't have them and imagine what we'd like them to be.

First, how should we call it? Since the EMM is a device driver, one obvious choice is to call it through the DOS OPEN handle and WRITE handle functions. But there's a problem with this scheme - DOS isn't reentrant. If we choose this method, device drivers and interrupt routines can't call the EMM.

Or we could access the EMM through a software interrupt (the way we normally call DOS and the ROM BIOS) and not worry about the non-reentrancy of DOS. As long as the EMM is reentrant, we can call it from interrupt routines. The disadvantage of this scheme is the lack of spare interrupts (very few free interrupts are left after the ROM BIOS, MS-DOS, and BASIC claim theirs).

But EMS found a spare - interrupt 67H. We put a function number in AH, and other parameters as needed in other registers. Status is returned in AH, and return parameters are placed in other registers. So an EMM call ends up looking a lot like a ROM BIOS call.

How 'Bout Those Functions

What functions did you say you'd like the EMM to have?

Well, imagine writing a program that will use expanded memory. You'll want to know (at least):

(1) How much memory is available,

(2) How to make a particular page visible, and

(3) Where the pages will be visible.

Let's consider the last first. You already know that EMS can make four different pages of 16K each visible at once. It would seem natural to locate these pages in one 64K region. Then, using a little care, programs could look at this as four 16K blocks, two 32K blocks, or one 64K block.

But locating this block poses a problem - it's not possible to find a 64K block that's free on everyone's PC. The location of this block will depend on the specific PC, and the other hardware that's installed.

EMM provides a function which tells software where the 64K window to expanded memory is located. Software can determine the base address of this region by calling EMM Function 2, Get Page Frame. Function 2 will return a segment value. The four physical pages are located at fixed offsets within this segment:

Bank switched memory allows a single CPU memory address to access multiple memory locations.

```
Page 0   0000H
Page 1   4000H
Page 2   8000H
Page 3   C000H
```

We can't expect every EMS board to have the same amount of memory; therefore, programs must determine how much memory is available. EMM Function 3 (Get Unallocated Count) does that for us.

We also need a way to select which pages are visible. EMS provides EMM Function 5 (Map Handle Page) to tell the

(continued next page)

(continued from page 33)

EMM to map a particular logical page to a designated physical page.

This seems a minimum set of functions necessary to implement an expanded memory manager. (But there is one little twist, which I'll save for a few paragraphs.)

EMM adds two other functions to make the spec look professional.

Function 1 (Get Status) returns a simple working/not working indication to software.

Function 7 (Get EMM Version) returns the current version of the expanded memory manager. (By the way, this article describes EMS version 3.2. I know of no EMS hardware that doesn't work with version 3.2, and no software which requires an earlier version.)

Multitasking

Now let's add that little twist.

While DOS can't be called a true multitasking operating system, there may be several "tasks" simultaneously occurring.

In addition to the "running" program, DOS itself is loaded. And although DOS doesn't use expanded memory, there may be a place for expanded memory in a future version (since Microsoft was involved in writing the spec). EMS should keep this possibility open.

Also, other device drivers, a background print spooler, terminate-stay-resident programs, and other interrupt service routines may be operating. All of these programs should be able to use the expanded memory. So let's look for functions that allow several programs to use EMS simultaneously.

The EMM will need to somehow track which pages belong to which programs. A program will be required to request memory before it can use it, and release the memory it owns when it's through with it. Interrupt service routines will also need a way to save and restore the EM state of the foreground program.

To facilitate multiple programs, EMS provides:

EMM Function 3 (Get Unallocated Count) returns the number of free pages (i.e., those not allocated to other programs).

EMM Function 4 (Allocate Pages) reserves pages for the exclusive use of the calling program. Other programs aren't allowed to manipulate these pages. The Allocate Pages function returns a handle which the program uses in other calls to identify its pages.

EMM Function 5 (Map Handle Pages) uses the handle assigned by the Allocate Function to map a logical page to a physical page. The handle ensures that the page belongs to the calling program.

EMM Function 6 (Deallocate Pages) returns pages to the EMS pool.

EMM Function 8 (Save Page Map) and EMM Function 9 (Restore Page Map) enable interrupt service routines and device drivers to safely use expanded memory.

The Save and Restore functions work well for interrupt service routines and device drivers, but aren't powerful enough for an operating system. EMM Function 15 (Get/Set Page Map) provides what a multitasking operating system needs to incorporate expanded memory.

Finally, EMS throws in some functions which appear to be designed for diagnostic programs.

EMM Function 12 (Get EMM Handle Count) reports how many handles are currently being used.

EMM Function 13 (Get EMM Handle Pages) reports how many are assigned to a particular handle.

EMM Function 14 (Get All EMM Handle Pages) builds a table of all active handles and the number of pages assigned to each.

And that's the end of the preview (and a little motivation). Now let's get to the details.

Definitions

Conventional Memory - memory which an 8088 can address directly, from 0 to 1 Megabyte.

Since only the first 640K of the 1M address space usually contains general purpose RAM, conventional memory is sometimes considered to be from 0 to 640K, exclusively. But we don't need to get too picky about the exact range.

Extended Memory - an 80286 and 80386 can access memory above 1 Megabyte directly. Memory in the 80286 and 80386 address space, but not in the 8088 address space, is called extended memory. Extended memory isn't available on an 8088 based computer such as the IBM XT. And an 80286 in real address mode can't address extended memory.

Expanded Memory - bank switched memory, together with a software driver, which adheres to the Expanded Memory Specification.

EMM - Expanded Memory Manager, a device driver which provides the interface between programs and expanded memory hardware.

EMS - Expanded Memory Specification, which tells programs what to expect from expanded memory hardware, and expanded memory hardware what to expect from programs.

Page Frame - the expanded memory is addressed through a 64K block in the processor's address space. The page frame is the segment address of this 64K block.

Physical Page - the 64K block of the page frame is divided into four 16K blocks called physical pages. The physical pages start at offsets 0000H, 4000H, 8000H, C000H in the page frame, and are identified as physical page number 0, 1, 2, 3, respectively.

Logical Page - all expanded memory is divided into 16K blocks called logical pages.

Page Map - at any given moment, four logical pages of expanded memory are mapped to the four physical pages of the page frame. When logical pages are mapped this way, they can be accessed by reading and writing to the addresses of the physical page. The page map tells which logical pages are mapped to which physical pages.

Handle - logical pages are assigned to programs in groups of varying sizes. Each group is assigned a number or handle. Programs use this handle when requesting expanded memory services. Handles allow several programs to use expanded memory without conflict.

The Expanded Memory Spec

Calling the EMS services is similar to calling ROM BIOS or DOS services; these calls are made through software interrupts. EMS services are called by placing a function number in the AH register, then executing an INT 67H instruction. Status is always returned in AH. AH = 0 indicates that the service request was successfully completed. A non-zero value in AH indicates an error condition.

For example:

```
MOV  AH, 40H
INT  67H
OR   AH, AH
JZ   EM_WORKING
JNZ  EM_ERROR
```

tests expanded memory status.

EMS Function Descriptions

Status -

```
In:     AH = 40H
Out:    AH = Status
```

returns EM status. If AH = 0 is returned, then the expanded memory hardware and software are working. AH = 80H indicates a software malfunction. AH = 81H indicates a hardware malfunction.

Page Frame -

```
In:     AH = 41H
Out:    AH = Status
        BX = Page frame segment
```

returns the segment address of the page frame. The page frame address can be set by jumpers on the expanded memory board. D000H is a typical choice. A000H is a good alternative as long as EGA video isn't being used. However, software shouldn't rely on the hardware jumpering, but should use this function to determine the page frame. When software uses this function, the hardware may be changed without altering the software.

Unallocated Count -

```
In:     AH = 42H
Out:    AH = Status
        BX = # of unallocated pages
        DX = Total number of
             expanded memory pages
```

The number of unallocated pages is returned in BX. Generally, software calls this function to see what's available before requesting expanded memory. Device drivers and memory resident programs may request some percentage of the free memory and leave the rest for other programs.

Allocate Pages -

```
In:     AH = 43H
        BX = # of pages to allocate
Out:    AH = Status
        DX = Handle
```

We assign expanded memory to a program by using the Allocate Pages Function. We put the number of pages we're requesting into BX. If the request is granted, the function returns with 0 in AH, and a handle in DX.

We use the handle in other expanded memory calls to identify a particular group of logical pages. The logical pages assigned to a handle are referenced by the numbers 0 through "number allocated to the handle" - 1. Error status 85H is returned if there are no free handles. Error status 87H or 88H means there aren't enough free pages to fill the request, and error 89H is returned if zero pages are requested.

Map Page -

```
In:     AH = 44H
        AL = Physical page number
        BX = Logical page number
        DX = Handle
Out:    AH = Status
```

The Map Page Function maps the specified logical page to the specified physical page. The logical page is identified by the handle assigned when the page was allocated (Allocate Pages function) and a page number between 0 and "number allocated to the handle" - 1.

The physical page number must be in the range 0 to 3. The physical page number is passed in AL, the handle in BX, and the logical page number in BX. Possible errors are 83H (invalid handle), 8AH (logical page out of range), and 8BH (physical page out of range).

Deallocate Pages -

```
In:     AH = 45H
        DX = Handle
Out:    AH = Status
```

releases all pages allocated to handle DX. The released pages can then be allocated to other pages. Programs should make this call before terminating. Error status 83H is returned if DX contains an invalid handle number.

A handle may have a save state assigned to it by function 47H. Error code 86H is returned if the handle has an active save state which hasn't been restored via function 48H.

Version Number -

```
In:     AH = 46H
Out:    AH = Status
        AL = 32H
```

returns the EMS version number in AL - major version in upper nibble, minor version in lower nibble. The present version number is 3.2.

Save Page Map -

```
In:     AH = 47H
        DX = Handle
Out:    AH = Status
```

saves the current page map. It can be restored later with function 48H (restore page map). Device drivers, memory resident programs, and interrupt service routines that use expanded memory should use this call.

The save and restore calls will preserve the state of expanded memory for the foreground program. The handle in DX belongs to the program doing the save (the interrupt service routine), not to the foreground program using expanded memory.

Error status 83H is returned if the handle number isn't valid. There can be at most one saved page map for each handle. Error code 8DH is returned if a saved page map already exists for the specified handle. The save area is limited, and error status code 8DH is returned if there isn't enough room to save another page map.

Restore Page Map -

```
In:     AH = 48H
        DX = Handle
Out:    AH = Status
```

This restores the page map saved by function 47H, under handle DX. Error status 83H is returned if DX doesn't contain a valid handle number. Error status 8EH is returned if no saved page map exists for handle DX.

Handle Count -

```
In:     AH = 4BH
Out:    AH = Status
        BX = # of active handles
```

returns the number of active handles in BX.

Page Count -

```
In:     AH = 4CH
        DX = Handle
Out:    AH = Status
        BX = # of pgs in handle DX
```

returns the number of pages allocated to handle DX in register BX. Error status 83H indicates that DX isn't an active handle.

Page Count Array -

```
In:     AH = 4DH
        ES:DI = Pg cnt array ptr
Out:    AH = Status
```

returns an array of the active handles and the number of pages assigned to each. ES:DI should point to a buffer of 4 * (number of active handles) bytes.

The number of active handles can be determined by function 4BH. There is a two-word entry in the array for each active handle. The first word of each entry is the handle number; the second word is the number of pages allocated to that handle.

Set Page Map -

```
In:  AH = 4EH
```

(Get/set page map function consists of four subfunctions. Pass the subfunction code in AL.)

```
In:     AL = 00H
        ES:DI = Page map array ptr
Out:    AH = Status
```

puts a copy of the page map in memory at ES:DI. The size of the array can be determined by get/set subfunction 03H (see below). The format of this array isn't part of the expanded memory specification, and programs shouldn't try to interpret the data in the page map array. It should only be used for the Set Page Map (01H) and Get and Set Page Map (02H) subfunctions below.

Set Page Map -

```
In:     AL = 01H
        DS:SI = Page map array ptr
Out:    AH = Status
```

Set the page map by the array at DS:SI. The size of the page map array can be determined by EMS function 4EH, subfunction 03H. The array must

(continued next page)

(continued from page 35)

have been filled in with EMS function 4EH, subfunctions 00H or 02H.

Get And Set Page Map -

```
In:     AL = 02H
        DS:SI = Pg map srce for set
        ES:DI = Pg map dest. for get
Out:    AH = Status
```

This subfunction does both a get page map and a set page map in one function call. The get and the set arrays must be different.

Page Map Size -

```
In:     AL = 03H
Out:    AH = Status
        AL = Map array size
```

returns the size of the page map arrays used in EMS function 4EH, subfunctions 00H, 01H, and 02H.

Typically, programs will only use expanded memory functions 40H through 46H. Interrupt service routines, device drivers, and memory resident background programs will need functions 47H and 48H. Functions 4BH through 4DH are for test and utility programs which track expanded memory. Function 4EH is for an operating system which incorporates expanded memory.

All EMS function calls return status in the AH register. Figure 1 summarizes the status codes.

Eight Megabyte Limit

The LIM-EMS mentions eight megabytes in the introduction - "The EMM is a set of standard interface routines which allow programs running on IBM PCs, XTs, ATs and compatibles to use up to 8M bytes of expanded memory." And in the description of the number of pages value returned by Function 13: "This number never exceeds 512 because the EMM allows, at most, 512 pages of expanded memory." When you have 512 pages of 16K each, you have eight megabytes.

Sometimes the eight megabyte limit seems to be per handle. But at other times, it seems to refer to total expanded memory. There's no good reason for this limitation. Page numbers are always passed in 16-bit registers. There are no places in the spec where funny things are done with the high order bits. The spec can just as easily handle 64K pages, or 1 gigabyte total. Of course, it may take a few years to solve the problem of fitting a gigabyte into one slot. (The PC Tech 16 Megger board has 16 megabytes of EMS compatible expanded memory.)

Detecting The EMM

Programs that use expanded memory should make sure the expanded memory manager is installed before calling it. The EMM is installed as a DOS device driver, so we can use DOS calls to verify that the EMM driver is installed.

The expanded memory device driver has the unlikely name of "EMMXXXX0." Programs should attempt to open the EMS device driver with the DOS Open Command. If the Open fails, then the expanded memory manager isn't installed. If the Open is successful, the program should do a DOS IOCTL call to ensure that a device was opened, not a file. The program should close the EMS device driver before terminating.

Figure 2 is a subroutine which can be used to test if the expanded memory manager is installed.

This method (the "Open Handle") is one of two methods which we can use to detect expanded memory. The second is called the "Interrupt Vector" method.

The Interrupt Vector method works like this - a program looks at the interrupt vector for interrupt 67H, the EMM interrupt. This vector should point to the entry point for the EMM. The program looks only at the segment value of this vector; the offset value is ignored.

The EMM software is a device driver, which is a special type of "COM" file. In particular, the code and data segments are the same. Device drivers always have a "device driver header" at offset 0000H.

The segment value from the interrupt vector can be used to look at the device driver header. At offset 000AH in the device driver header of character devices is the device name. The name of the EMM device driver is "EMMXXXX0." If a program can find the string "EMMXXXX0" at this location, it can assume that the EMM is installed.

In my opinion, the Interrupt Vector method is flawed. The assumption that the segment value of int 67H is always the same as the segment where the EMM device driver is loaded isn't valid. It's not true that a device driver must always occupy just one segment. It's possible to write device drivers which use multiple segments.

Therefore, you should view this assumption not as a statement about device drivers in general, but rather as a restriction on EMM drivers. This restriction isn't stated explicitly anywhere in the spec, only implicitly, and subtly, in the description of the interrupt vector detection method.

However, the success of this method isn't guaranteed, even when the EMM observes the "entry segment = device driver header segment" convention.

One of the very few interrupts on the PC not reserved by IBM or by Microsoft is 67H. These non-reserved interrupts seem to be the target of every programmer who wants to do something a little bit tricky. It may be hard to believe, but some of these programs do try to coexist with others. Sometimes they even manage to share interrupts.

A program can intercept interrupt 67H without disrupting EMS. All it has to do is make sure that all calls with a valid EMS function code in AH get passed on to the EMM. At least such a program would work until someone tries to run another program which uses the "Interrupt Vector" detection method. (This isn't pure speculation; it happened to someone I know very well.)

The designers of EMS may have anticipated this program. The expanded memory specification flatly states that no programs which use interrupt 67H can be run while EMS is active. However, I suspect they were warning against programs that steal interrupt 67H completely, not programs that try to share it, and the little traps in the "interrupt vector" detection method weren't foreseen.

The EM specification states that the interrupt vector detection method is for device drivers and interrupts which can't call DOS.

I've experimented with device drivers and found that a device driver can successfully call the DOS open handle function from within the init code of the device driver. It shouldn't be necessary to test for the presence of EMM from anywhere else. Of course, Microsoft put their name on the spec, and Microsoft MUST know something about MS-DOS that I don't.

As for interrupt service routines, there will be trouble if you call DOS from an interrupt which has interrupted a DOS function. However, an interrupt shouldn't have to test for the presence of an EMM. Any questions of whether an EMM is installed should have been answered when the interrupt service routine was installed, long before the interrupt actually occurs.

We are only questioning when a program may use the DOS open handle function to detect whether an EMM is present. EMS is reentrant; DOS isn't. There are no problems with calling the EMM through interrupt 67H from a device driver or from an interrupt.

Expanded Versus Extended

On an 80286, the protected mode bit is a sticky bit; once it's turned on, it can't be turned off without resetting the processor. Extended memory can only be accessed in protected mode, and DOS must run in real address mode.

It would seem, then, that once we access extended memory in protected mode on an AT, we can never return to DOS. But IBM has come up with an ingenious method (some say, "stupid method") to unstick the sticky protected mode bit. While this method is clever, it is also slow - much slower than switching a bank of expanded memory.

In short, nearly all of the advantages of extended memory are lost on an AT running DOS.

I devised a quick comparison of expanded memory versus extended memory.

For the expanded memory test, I used an AT, a PC Tech 16 Megger board, and MS-DOS RAMDRIVE.SYS memory disk program set for a 3-megabyte memory disk in expanded memory.

For the extended memory test, I used the same AT, the same 16 Megger board, and RAMDRIVE.SYS set up as a 3-megabyte memory disk in extended memory.

The CPU didn't change, and the 16 Megger can be either expanded or extended memory, with no advantage in the access time in one mode over the other.

I'm not sure about the memory disk program, RAMDRIVE.SYS. I assume the two modes of RAMDRIVE.SYS, "expanded" and "extended," will have as much code in common as possible, and that only the actual memory transfer routines will be different.

The test was a batch file which reset the system clock, wrote 20 copies of a 128K file, erased the 20 copies, and repeated 6 times for a total of 120 - 128 Kbyte copies. Then, the batch file displayed the time.

The result - 62 seconds using expanded memory and 74 seconds using extended memory.

The IBM VDISK program (the memory disk program that comes with PC-DOS) used to have a problem with interrupts. When extended memory was used, the interrupts were shut off.

This was done because the 80286 must be in protected mode to access extended memory, and most of the interrupt service routines were written as-

(continued next page)

Figure 1 — Error codes returned in AH.

```
AH = 0          No error.
AH = 80H        Software failure.
AH = 81H        Hardware failure.
AH = 83H        Inactive handle specified.
AH = 84H        Illegal function code.
AH = 85H        No free handles. (Function 43H.)
AH = 86H        Handle has saved state. (Function 45H.)

(A handle can't be deallocated while there's a saved state
associated with that handle.)

AH = 87H        Not enough pages. (Function 43H.)
AH = 88H        Not enough unallocated pages. (Function 43H.)
AH = 89H        Can't allocate zero pages. (Function 43H.)
AH = 8AH        Logical page out of range.
AH = 8BH        Physical page out of range.
AH = 8CH        Save area full. (Function 47H.)
AH = 8DH        Handle already has saved state. (Function 47H.)

(A handle can only have one saved state associated with it at a
time.)

AH = 8EH        No save state associated with specified handle. (Function 48H.)

(An attempt has been made to restore a page map that was not
saved.)

AH = 8FH        Illegal subfunction number. (Function 4FH.)
```

Figure 2 — Test routine to detect functioning expanded memory driver.

```
  EM_NAME               DB 'EMMXXXX0', 0

FIND_EM:
;Out:   NC indicates expanded memory driver installed and functioning.
;       BX = EM base segment if EMM installed
;       C set if expanded memory can not be used.
  PUSH  DS
  PUSH  DX
  PUSH  AX
  MOV   AX, CS
  MOV   DS, AX              ;Use DOS open handle call to determine whether
  MOV   DX, OFFSET EM_NAME  ;EM driver is installed.
  MOV   AH, 3DH             ;Open handle function code.
  MOV   AL, 0
  INT   21H                 ;Dos function.
  JC    FE_00               ;C set if open fails.
  MOV   BX, AX              ;Device handle
  MOV   AH, 44H             ;IOCTL call
  MOV   AL, 0               ;Device info subfunction
  MOV   DX, 0
  INT   21H                 ;Call DOS
  JC    FE_00               ;Exit, C set if error
  TEST  DL, 80H             ;Is it a device?
  STC
  JZ    FE_00               ;Exit with C set if not device.
  MOV   AH, 44H             ;DOS IOCTL call
  MOV   AL, 7               ;Output status
  INT   21H
  JC    FE_00               ;Exit, C set if error.
  OR    AL, AL              ;Is it ready?
  STC
  JZ    FE_00               ;Error exit if not.
  MOV   AH, 3EH             ;Close handle
  INT   21H
  JC    FE_00
  MOV   AH, 41H             ;Get EM base address.
  INT   EM_INT
  MOV   EM_BASE, BX
FE_00:
  POP   AX
  POP   DX
  POP   DS
  RET
```

(continued from page 37)

suming that the 80286 was in real address mode. Some interrupts may be lost during memory disk transfers. This is particularly important with a communication program, since incoming characters are lost.

This VDISK problem came with PC-DOS 3.1, and I don't know whether it's been fixed in later versions. I'm also not sure whether Microsoft's RAMDRIVE program has the same flaw. So you should test for lost characters before mixing a communication program with any extended memory disk.

EMS Emulators

EMS was created for bank switched memory add on boards, but the spec does a good job of hiding the hardware particulars from the software. With a little inventiveness, one can turn just about any storage into EMS.

My first experience with EMS was along these lines. PC Tech's X16 computers have one megabyte of RAM. DOS only uses 640K, so we turned the remainder into EMS memory. The memory above 640K on the X16 can be turned on and off, but it can't be bank-switched as EMS memory usually is.

EMS hides these details from application software. When a program asks the X16 EMM to switch pages, the X16 EMM will shuffle the appropriate bytes around in memory so that the right data appears at the right locations.

Others have done similar things. There is a program on the market that will turn extended memory on an AT into expanded memory. The "bank switching" is accomplished by moving memory between extended memory and conventional memory. There's also a program that will emulate expanded memory on your hard drive. Here, the "bank switching" is a transfer between the hard disk and conventional memory.

There are two things to watch out for when using an EMS emulator.

One obvious difference between a real expanded memory board and these emulator programs is the time it takes to do a page switch. On an expanded memory board, a page switch is accomplished by doing an output to a page register, one CPU instruction. The emulators must move 32K of data, save the old 16K page, then write the new page. This requires at least 8000 CPU instructions.

The question becomes, "How often does the program have to switch pages?" If page switching is relatively rare, then programs may run even faster on the emulator. I wrote an EMS memory disk driver for the X16 with speed as the primary goal.

This memory disk must be the only program using expanded memory, so there will be no need to save and restore the EMS state on each disk access. The directory and FAT were always kept visible, only one page with data sectors was switched. Since this page is 16K, it holds 32 - 512 byte sectors. Often, the desired sector in a disk transfer was already visible.

The souped-up memory disk program and the expanded memory emulator are fast. There's no visible speed difference between them and a "real" expanded memory board. The EMS emulator may be even faster. With the X16 EMM, the visible memory is 16 bits wide, 8/10 Mz, no wait states. Expanded memory is 8 bits wide and runs at I/O bus speeds.

On the other hand, Microsoft's RAMDRIVE.SYS was written with the assumption that other programs would be using expanded memory. It does a Save Page Map and a Restore Page Map function call on every disk access. Each call requires that the emulator exchange 64K. With an emulator, RAMDRIVE.SYS is slower than a floppy.

A while back I mentioned two differences. The second is very subtle. The expanded memory specification doesn't say you can't have one logical page mapped to two or more physical pages simultaneously. It also doesn't say you can.

An expanded memory board can easily do this strange mapping. This isn't possible with an emulator program. It's very unlikely a program would need to map a logical page to more than one physical page.

Programming Suggestions

Use the open handle method for detecting the expanded memory manager. Don't use the interrupt vector method.

Consider that someone might want to run your program on an EMS emulator. Don't switch pages unnecessarily. And don't try to map one logical page into two different physical pages at the same time.

Deallocate all the expanded memory your program owns before exiting. This isn't as easy as it sounds. What happens if your program exits because the user typed ^C? Or, what if the user types "A" in response to "Abort, Retry, Ignore?" Are there any other exits in your program? Does your program abort when it detects an error? I know of a related bug in a program written by someone who should know better, Lotus.

Handles are a limited resource. EMM status code 85H (No Free Handles) may be returned by the Allocate Pages function. This implies a limit to the number of EM handles that can be active at any one time. The spec doesn't state how many handles EMM must provide, and it lacks a "free handle count" function call.

The intention of the expanded memory specification is clear - one handle to a program.

Handles were created so EMM could keep track of each program's pages. They weren't intended to be an extension of the logical page number. Programs should anticipate their memory needs and claim that amount of memory in one allocate call. If the initial anticipation of needs was short, then one or two more allocate calls may be acceptable. Programs shouldn't try to do their own memory management by issuing allocate calls whenever memory is needed.

It's unfortunate that EMS doesn't have "Allocate More Pages To Handle" and "Free Some Handle Pages" functions. Programs that try to simulate these functions with the allocate and deallocate functions will run out of handles long before they run out of memory.

Save pages are a limited resource. There can be at most one save page for each handle. The Save Page Map function may return EMM status code 8C (Save Area Full). This implies possible room for fewer saved page maps than handles. EMS doesn't specify a minimum number of saved page maps, and there's no "Get Free Save Space" function.

The Save Page Map function was intended for interrupt service routines and device drivers. It provides a way for these programs to save and restore the EMS state established by other programs. This function is to be used only when the current EMS state isn't known. A program shouldn't use the Save Page Map function to save its own page map. A program should know how it has the EMS pages set up, and be able to reconstruct the page map without the Save/Restore functions.

The folks at Lotus who helped create the expanded memory specification must not be the same people who wrote Symphony. Symphony will call the Allocate function eight times, tying up eight handles. Symphony will do a "Save Page Map" for each of these eight handles, thus tying up eight save page map tables.

Symphony never uses the saved page maps - they just sit until the program is exited. They're restored just before the program exists, and the restored maps are never used.

The EMM status code 8CH (Save Area Full), which may be returned by the Save Page Map function, is ignored by Symphony. Symphony assumes the save was successful and goes on. This has no effect until you try to exit Symphony.

Symphony tries to restore the page map that was never saved. It gets EMM status 8EH (No Saved State Associated With Specified Handle). Symphony won't return to DOS when it sees this error. The power switch, reset button, or Control-Alt-Del are the only ways to get back to DOS.

Future

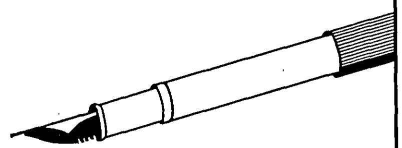
Despite the existence of more powerful microprocessors, the 8088 can't be ignored. The installed base of 8088-based clones is still very attractive to software sellers. And EMS eliminates one of the biggest limitations of these computers, the 640K DOS memory limit.

EMS will definitely extend the lifetime of MS-DOS. To some, this is good news. Others would rather see the 8088 and DOS replaced by a more powerful CPU and operating system, the sooner the better.

I believe MS-DOS will remain the most popular operating system for some time yet, and EMS will be the preferred way to write large programs for DOS.

It seems that all you hear from software publishers is "adding more features." No one talks about streamlining code these days. Let's hope that more memory leads to better programs, not merely larger ones.

■■■



Writer Guidelines

Articles should be submitted on disk or via Micro C's bulletin board (503-382-7643, 300/1200/2400, 24 hrs.). We can read most disk formats except Apple, Rainbows, NorthStar, and Eagle CP/M, but please mark the format on the disk.

Follow these guidelines:

1) Use standard ASCII format.

2) Leave a blank line between paragraphs.

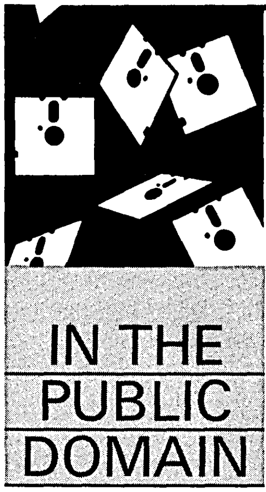
3) Don't indent paragraphs.

4) Don't right-justify your article. (Leave the right margin ragged, like this document.)

5) Don't use hyphens except when they're part of a word or phrase. If you let your word processor add hyphens automatically, we have to remove them individually.

6) Don't use tabs. Use spaces when you need to indent.

THE MICRO TECHNICAL JOURNAL
MICRO CORNUCOPIA



"If It's So Good, Then Why Don't They Sell It?"

Anthony Barcellos
P.O. Box 2249
Davis, CA 95617
(916) 756-4866

Tony is well-known around the public domain circles as a champion of shareware. Besides being librarian for the Sacramento PC group, he's a regular contributor to their newsletter. Here he reviews what he feels are the plums in the burgeoning shareware marketplace.

People who ask the above question really haven't been paying attention. Shareware authors do sell their programs. Shareware is just commercial software without the customary commercial distribution channels. It's also called "user-supported software," since it depends on the voluntary remittances of users for its success.

You've heard about "cutting out the middle man?" In the case of shareware, the "middle men" are volunteering their services. The distributors of shareware include satisfied individual users, electronic bulletin board systems, and computer club software libraries.

Shareware was invented by the late Andrew Fluegelman when he began to distribute his now-classic PC-Talk III communications program as "Freeware." ("Freeware" is a registered trademark of Fluegelman's company, The Headlands Press, so "shareware" came into use as a generic substitute for the term.)

"But Is It Any Good?"

Sure it is. Many of the programs are highly competitive with traditional commercial offerings. Shareware never comes with copy protection schemes that frustrate you, corrupt your hard disks, or treat you as guilty until proven innocent.

We who help to distribute shareware can easily single out several superb programs that are among the best for their particular application. They form a sampler of shareware's big-time winners. For example...

PC-Write

Bob Wallace was a professional programmer with Microsoft when he got the itch to write a word processor. The result, PC-Write, is one of the shareware classics. Now up to version 2.71, PC-Write is no slouch when it comes to supporting different printers (nearly 400, including lasers), handling two documents simultaneously, checking your spelling, and formatting your text (for example, microspace justification, font changes, and variable line spacing).

Wallace's new company, Quicksoft, has 20 employees, issues a periodic newsletter, and offers technical support by phone to registered PC-Write users. While it costs $89 to register PC-Write, the user can recover the fee in $25 increments by passing copies of PC-Write to friends.

Quicksoft remits $25 of each registration fee to the original registrant (identified by the serial number on the registration form included in the package). Some of the larger computer clubs (like Capital PC in Maryland and Sacramento PC in California) have waived the commission, so that their members can obtain a fully-registered copy of PC-Write for only $64.

The curious user can order a two-disk trial set from Quicksoft for $16. This fits Quicksoft's sales pitch: "Use it first, then buy it." That's a good definition of shareware.

PC-Write version 2.71
Quicksoft
219 First N. #224
Seattle, WA 98109

PC-File+

Jim Button wrote a simplified database program to use on his Osborne computer, since dBASE II was a lot more than he needed. He shared it with his friends. Some of them liked it and suggested improvements. When Jim started using an IBM PC, he transferred his database program to it and continued to add enhancements.

Fluegelman's PC-Talk was soaring in popularity and the shareware concept beckoned to Button, who began to distribute PC-File in the same way. Soon Button had established ButtonWare to handle the numerous registration fees, requests for technical assistance, and suggestions for improvements.

PC-File underwent several revisions. The latest is PC-File+, version 1.0 (the successor to what Button called version 4.0 of PC-File III, for those of you who are trying to keep track of what's current).

Speedy, menu-driven, and easy to use, PC-File+ sports help screens, file sorting and searching, report writing, automatic generation of calculated fields, macro capability, mail merge, and label printing. Its ability to read and write standard text files permits exchange of data with many other programs. PC-File+ offers a DOS shell for easy access to operating system commands while in the database.

PC-File has become the centerpiece of a large assortment of programs from ButtonWare. These include PC-Dial for communications, PC-Graph for business graphics, PC-Type+ for word processing, PC-Calc for spreadsheets, PC-Style for evaluating writing style, and PC-Tickle to manage appointment schedules. More information on these programs can be obtained from ButtonWare.

PC-File+ is available from ButtonWare for $69.95 plus $5 shipping and handling. (Washington State residents will need to add 7.9% sales tax.)

PC-File+, version 1.0
ButtonWare, Inc.
P.O. Box 5786
Bellevue, WA 98006
Toll-free: 1-(800)-JBUTTON

ProComm and Qmodem

What happened to Andrew Fluegelman's PC-Talk III? It faded as Fluegelman turned his attention to the Macintosh. His untimely death found PC-Talk 4 still unfinished. Although the release of PC-Talk III's successor has now been announced, PC-Talk 4 is being distributed through regular channels. It is not Freeware.

Users eager for additional features and more sophisticated communications programs found inexpensive shareware alternatives to the regular commercial packages and PC-Talk III. DataStorm's ProComm and the Forbin Project's Qmodem are among the best of these.

Both programs offer large dialing directories, with ample choices for dialing prefixes (for example, alternate long-distance service access numbers like Sprint), plenty of file-transfer protocols (Xmodem, Ymodem, ASCII), support for faster speeds like 2400 bps (even up to 19,200 bps!), and sophisticated script execution capabilities (for unattended operation).

ProComm offers sound effects (which can, mercifully, be turned off) and Qmodem specializes in fancy pop-up windows. ProComm's file-transfer protocols are accessed via function keys.

Since both programs are so capable, a choice is largely a matter of taste. Users who recoiled from ProComm's penny-arcade sound effects have chosen Qmodem. (One of ProComm's authors recently admitted that he hated the noise. The current release of ProComm lets you turn off the sound.) If you need fancy terminal emulation choose ProComm, and if you can't decide, get both. Together they'll cost you less than a regular commercial package.

You can obtain a trial copy of ProComm for $10 from DataStorm Technologies; they will send you a disk containing the program and its documentation. For $25 they will record you as a registered user and provide support over their bulletin board system; an additional $10 will get you the disk containing the latest version of ProComm and its on-disk documentation. Those who register at the $50 level obtain a printed, bound manual as well.

ProComm, version 2.4.2
DataStorm Technologies, Inc.
P.O. Box 1471
Columbia, MO 65205
(314) 449-7012
BBS: (314) 449-9401

Qmodem can be obtained for $30 from The Forbin Project.

Qmodem, version 2.3
The Forbin Project
c/o John Friel III
4945 Colfax Avenue South
Minneapolis, MN 55409
(612) 824-1451

Automenu

The DOS prompt is not particularly user-friendly. Many software packages have been offered as alternatives. These "DOS shells" insulate users from DOS and make standard operating system functions (such as file copying, deleting, or typing) simple menu choices. The better shells offer features beyond making DOS menu-driven; these include "tagging" files so you can manipulate several files at once, access to applications packages, macro capability, screen editing, and customization.

The clear shareware choice is Automenu from Magee Enterprises. Program author Marshall Magee has created a shell that offers single-keystroke selection of applications programs, batch files, or DOS commands, on-screen help and prompts, and dual-monitor support (for all of you with two-headed PCs). It also includes password protection of selected programs for data security, custom menu design, and screen blanking (to protect the monitor(s) when not in use).

Since DOS shells are notorious for running afoul of system idiosyncrasies (as well as adding new ones), Magee bulletproofs Automenu by enabling his shell to recognize system errors and set the DOS flag ERRORLEVEL. Since batch files can test ERRORLEVEL and branch

Shareware never comes with copy protection schemes that frustrate you, corrupt your hard disks, or treat you as guilty...

on its result, graceful recovery from program problems is possible. While this feature is of greater interest to the power user than to your average computer jockey, it shows Magee's attention to detail.

A less tangible feature of Automenu is Magee's fervent support of users groups. His advertisements for Automenu have appeared in Capital PC's Monitor, Houston's HAL-PC newsletter, Boston's PC Report, New York's NYPC, Sacramento's Sacra Blue, and San Francisco's Blue Notes, among others. His ubiquitous presence is a constant reminder of Automenu's target audience - the workaday computer user - and undoubtedly invites the steady flow of user feedback that causes Automenu's revisions to be faithful reflections of current user needs.

Magee Enterprises experimented with regular commercial distribution of

(continued next page)

(continued from page 41)

Automenu version 4.0, while maintaining an earlier version for the shareware market. Magee recently announced that 4.0 will be released as shareware. Pending completion of the new documentation, Automenu will be available for $50 from Magee Enterprises in a complete, registered version with manual; for $10 you can get a starter kit that includes just a diskette with documentation in a disk file.

Automenu, version 4.0
Magee Enterprises
6577 Peachtree Industrial Boulevard
Norcross, GA 30092-3796
(404) 446-6611
BBS: (404) 446-6650

LIST

With DOS's TYPE command you can view a text file on-screen - provided you're quick enough to tap Ctrl-NumLock whenever you need to freeze the scrolling. And there's no going back. Vernon Buerg's LIST program is the preeminent substitute. Not only does it permit you to scroll up and down through a file, it will send text to your printer, it can search for characters you specify, it will scroll horizontally through long lines, and its "hex dump" feature will handle binary files with aplomb.

LIST's built-in filters will optionally expand tabs, suppress the high-bit (to handle characters in WordStar files, for example), and remove (from display, not the file) control codes and backspace characters. In short, LIST will let you look at any DOS file at all.

The registration fee for LIST is only $15. Buerg also maintains a 24-hour FidoNet bulletin board at (415) 994-2944 where you can find the latest version of LIST. (His FidoNet node is 125/4.)

LIST, version 6.1
Vernon Buerg
456 Lakeshire Drive
Daly City, CA 94015
BBS: (415) 994-2944

PC-Outline

When outlining programs - "thought organizers" - became the rage two years ago, no one should have been surprised to find a shareware product contending for space at the top of the heap. Shareware differs in kind rather than quality from regular commercial software, and PC-Outline is another example of how a shareware choice can often be the best choice.

With extensive word processing features built-in, PC-Outline permits the user to work in either "outline" or "text" mode. The outline mode automatically labels paragraphs depending on their depth in the hierarchy. If the default outline hierarchy doesn't suit you, PC-Outline will remember a customized format for you and use it as the future default.

The commands are accessed through a Lotus-like menu that is invoked by the slash key. Commands are executed by pressing the initial letter of the desired procedure. The user can hide lower levels of the outline to clean up the screen, then recall the suppressed information when he needs to access it.

PC-Outline does windows - up to nine of them at a time. You can have an outline in one window while working in text mode in another window. PC-Outline can save its documents in WordStar format or in the structured format used by ThinkTank or Ready!. The ASCII option permits the user to export outlines and text files to any other program that can deal with standard text files.

The program can be installed in a RAM-resident form, making it available from within any other program at the touch of a "hot" key. So you can use it with your regular word processor.

PC-Outline was written by John Friend and is now distributed by Brown Bag Software. (Brown Bag's version 3.24 of PC-Outline is essentially the same as Friend's original version 1.08.) Brown Bag offers several registration options for PC-Outline.

Level one costs $89.95 and includes the latest version of the program and its manual, a bonus program (Dr. DOS), and toll-free 24-hour phone support. Level two costs $49.95 and provides you with the current manual and 24-hour support via a standard phone line. For $29.95, level three simply registers you as a legitimate user and offers you "pay as you go" phone support via the standard line.

PC-Outline, version 3.24
Brown Bag Software
2155 South Bascom, Suite 114
Campbell, CA 95008
(800) 523-0764
(800) 323-5335 (California)
(408) 559-4545
BBS: (408) 371-7654 (up to 9600 bps)

What's Next for Shareware?

In February, Nelson Ford hosted a shareware convention in Houston. Bob Wallace, Jim Button, Marshall Magee, and most of the other shareware luminaries were in attendance. One outgrowth of this increased cooperation among shareware authors is the new Association of Shareware Professionals. We can expect to see more organized efforts by program authors to ensure that their products are in the software libraries of the major users groups and on the download directories of the most popular bulletin boards.

Nelson Ford and Marshall Magee are key figures in the organization of the association. Ford already runs The Public (Software) Library, which gathers shareware and issues a monthly newsletter for people interested in user-supported software. We can expect his newsletter to be a major forum for discussion of shareware's future and news on actual occurrences. Subscriptions are $12 per year.

The Public (Software) Library
Nelson Ford
P.O. Box 35705
Houston, TX 77235-5705
(713) 721-6104

■■■

ON YOUR OWN

Shareware Authors Talk About Their Experiences

By David Thompson

If you didn't catch this column last issue, you might think that shareware is just public domain by another name. As far as shareware authors are concerned it's not that at all, it's copyrighted commercial software. The only thing that makes shareware different from other commercial software is the way it's distributed.

Some shareware authors, notably Bob Wallace of PC-Write and Jim Button of ButtonWare, have been doing quite well in the profit column. Well enough into the six-figure column to make shareware interesting for anyone with the right program and copious amounts of time.

Last issue we heard from the authors of PC Key Draw, Automenu, PC-Outline, and PC-Write. This issue we finish up the PD/Shareware Convention with two discussions. The first lead by Jim Button of ButtonWare, the second lead by Sandy Schupper of Brown Bag Software.

ButtonWare

Jim Button began by saying he's not pioneering free software. He's pioneering the distribution of commercial software, a distribution which minimizes start-up costs, maximizes exposure, and maximizes voluntary payment.

"If I wanted to give software away without making any money, I know how to do that already. We're trying to tread that narrow path between giving things away totally free and offending no one, and maximizing the amount of money a programmer can make and still offend the fewest people.

"I'm not much of a marketing type; I've spent the last year of my life in a closet. They shove Twinkies and Coke through the door to me, and that keeps me going. I threw away all the source for PC-File and have been rewriting it. We'll be announcing that in two to three weeks.

"I get off on tweaking bits, making things run faster and smaller."

He did a survey of those ordering the package, where did they hear about ButtonWare:

Friend	38%
User group	19%
Magazine (ads & reviews)	16%
ButtonWare	8%
PC SIG	6%
Newspaper	3%
Dealer	3%
Bulletin board	2%
Buyer's guide	1%
Other	3%

"I send out 350 press releases every time we have something new. We usually get some response to the releases."

ButtonWare sales have continued to climb. Each time Jim announces an update, there's a surge. After the surge, sales remain higher than before. Their version 3 came out over a year ago. Since then they've just sent out little promotional pieces while Jim's been in the closet rewriting the code.

"That was kind of a gamble. I hoped I could keep the revenue coming in while I came out with a new version. We've done okay.

"Our peak was $130K in one month. Last year we did about $2,000,000. Most of that for PC-File."

Shareware Suggestions

"It may be that the things that have been successful for Bob and me wouldn't work for new authors. And, these aren't hard and fast rules, they're my thinking on what works and what doesn't.

"Games don't work. Games have traditionally been pirated. People who play games grew up with Ataris and Apples and part of the fun was getting other people's games. Another reason, you don't feel you're getting productive use out of it. You just enjoy it for a week or two and then you're through with it.

"Complexity sells. Part of the reason is that people will pay for support and updates or fixes. Perceived value of a complex program is much higher than that of a utility or other simple program, so people feel that $50 is cheap. If you have a simple utility, my recommendation is to put it into the public domain.

"General purpose programs work best. Shareware is like building a large snowball and rolling it down the hill. The snowball gets bigger and gains momentum as it goes. If you can't get the snowball going, then you're not going to make any money. If your program is too special-purpose, a compiler or something only a techie can use, he won't have a million friends to share it with and you won't get the snowball rolling. If you have a program like that, find someone to publish it for you. It won't work shareware.

"Go for the highest quality. These people eat and breathe their program. They love it as much or more than they love their wife. I didn't bring my wife along so I can say that."

(A comment from the audience, "My wife's license plate is PC Widow.")

"Everything I think about when I'm sitting in church while the preacher's trying to interest me in salvation is, what new feature can I put into my product?

"Some of the best ideas occur while sitting on the toilet in the morning.

"Price it right (between $20 and $75). If too low, customers assume the guy doesn't care. And, it isn't worth the user's time to sit down and write the check.

"If you price it too high, they'll think they can go out and purchase a commercial product with its slick packaging and advertising. And it costs too much to voluntarily pay for it.

"Provide as many incentives as you can think of. Newer versions are a big incentive. The reason new announcements bring in so much revenue is that it smokes out the closet users. With each version that goes out through the user groups, you acquire a whole new cadre of closet users.

"They decide not to send in a payment, but they're using it productively and getting a lot of use out of it. All of a sudden, along comes an update and they're saying, 'If I'd paid for this I'd have it already, now I have to wait for my club to get it.' Or they have to go to their friend to get it. 'He'll think I'm kind of a piker.'

"Part of the peaks we had in revenue were due to the $20 update fees, but more of it was the brand new orders.

"Don't use guilt as a shareware tool. My reaction when I read the pitch that says, 'If you don't send in your money for this, lightning will strike you dead within two weeks,' is that he's daring me. If there's guilt in there, I feel like it's a challenge.

"People will respond to pride, and the first versions of PC-Talk proved that. There were no incentives to buying PC-Talk, there were no incentives to buy the first versions of PC-File. But we received thousands of dollars because we said that people who were honest would pay for the programs.

"The reason to add the other incentives is to get the other people for whom honesty is not enough incentive.

"Support your product heavily. You're going to have to do your own technical support, marketing, sitting on the phone. You start out as the company. I worked eight hours a day at IBM, and I'd come home and try to do it all at night and Saturdays. You'll spend a lot of time. If you don't have the time, don't go shareware.

"Listen to your users. They know more about your product and what will make it sell than you do. I'm like others here. I don't listen very hard to those who call and say, 'I would send you money if you did...' because I've learned that they don't. I listen to those who have sent money. I want to reward those people.

"All the ideas I've had in the past three years have come from my users. I would have run out of ideas long ago.

"There's an optimum upgrade interval, for my product it's been eight to nine months. If I update more frequently, my users get angry. If I take longer, the revenues start falling off.

"That's why you need a more complex product. If you can't upgrade, then you lose this incentive. Pick a product like PC-Write, so you can program for the rest of your life and always have something new that will please people.

"Fortunately the newness of shareware was enough to get us going. If I had to start now, knowing the kind of marketing person I am, I don't know if I could make it.

"There are several ways to market: send out disks to user groups, send press releases to magazines, call up bulletin boards. I bought subscriptions to the Source and Compuserve. I didn't upload anything, but I put little commercials on their boards saying, 'I have a free program if anyone wants it.'

"I don't know if they allow you to do that anymore."

("Nope," came out of the crowd.)

"User groups have been the backbone of shareware, and for the past year we've had a campaign going to identify user groups. We used to require the librarian of each group to register. Now we don't.

"Now we're looking for ways to get shareware into the bookstores and other outlets, not as shareware but as regular retail product.

"A year ago we had no retail sales, so I announced a commercial version of ButtonWare with a few extra bells and whistles that I would sell for $150, because I was told that they didn't want to carry $50 software. And they don't want it to be shareware, so I made one that wasn't shareware.

"We now have almost 1,000 dealers (and I don't go to them; if someone calls in, we know what to do with him). I've found that they're selling the $50 product like hotcakes. Some of them like the $150 product, but many more like the $50 shareware version. And they'll be just as happy when it's all shareware. So the new PC-File+ will have everything. It will run five times faster, be twice as easy to use, cost $69.95, and it will be shareware. At that point I won't have a non-shareware product.

"For PC-file, about one-tenth pay, but I don't get upset about people who don't register. That's part of the price I agreed to pay to use this inexpensive method of advertising. I will eventually catch most of them with an update; and even if they don't ever pay, they will pass it on. And that's the best thing you can do for shareware authors.

"My experiment with butchered

Shareware is like building a large snowball and rolling it down the hill. The snowball gets bigger and gains momentum as it goes.

(continued next page)

(continued from page 45)

documentation was a failure. I had some (people) very unhappy. I think they'll be less unhappy if the manual isn't there at all. I'm thinking about no documentation at all because at 350 pages, it will be too big to fit on a disk, but the program is so easy to use that people won't need the manual. There's on-line context sensitive help."

Distribution Restrictions

At this point there was a question on ButtonWare's restrictions on distribution.

"The reason for many of the restrictions was early legal problems with a couple of user-supported resellers. There was a company in California (which will remain un-named) which was, for example, making copies of my disk manual and selling their copies of my disk and manual for $30 or $40. My response was that I was going to stop it somehow. And even though I couldn't afford it, I got a lawyer and we put a stop to it.

"I had to find a way of limiting the companies that were doing that sort of thing. That's why I put in the $10 maximum. I never put in restrictions to keep a company with integrity from distributing my software. I hope that people who find my statements restrictive will contact me."

Question: "File-express is your direct competitor - have you looked at what they're doing?"

"I don't tend to look at the competitor's product. If I do look at a competitor, it's to see if they have something that one of my customers hasn't suggested."

Brown Bag Software

"Our next speaker is that man you all love to hate, the chairman of Brown Bag Software who's been pirating away all these great shareware programs. I've offered equal time to anyone in the audience who he attacks." There was a laugh from the group as Nelson Ford sat down and Sandy Schupper took the podium.

"Before I start, let me give you an idea where my head is at. You remember Jim Button's comment about 'no games.' We started off in the business as a game publisher, so that's one of my perspectives, coming from the game industry. Low-priced software with everyone stealing it; distributors ordering huge quantities never intending to pay.

"Our other products include Home Base (a desktop utility), Power Menu (a menuing system), the Brown Bag Utilities (a floppy and hard drive file recovery system), and Goal Seeker (a spreadsheet work-along product).

"We come from a conventional distribution background. We're also well-known for our insertion of Bob Wallace's PC-Write in PC Magazine almost a year ago. We licensed a portion of Bob's code for private labeling. We called it Brown Bag Editor, which we are no longer distributing.

"We put a lock on the program which let the user load the program three times. After the program was loaded and exited three times, you needed to call us, send us money on a credit card, and we gave you a password and sent along a manual. We received a lot of flak for that, and I think that's how shareware got our attention.

"A lot of people questioned why we would put a shareware program that a lot of people could get for free inside a magazine. Obviously we spent quite a lot of money to do that. Well, if you want to get a mule's attention, they say hit him on the head with a 2X4. That situation taught me a lot about the power of shareware and taught me about perception.

"At that time the Brown Bag Editor and the utilities were copy protected. We were really upset when someone broke our copy protection, but we were conventional distribution. We spent a lot of money to advertise. We were spending between $60,000 to $80,000 per month for advertising.

"Anyway, we realized that there was great power in shareware, and Bob Wallace and I had many conversations when I would get really upset about people stealing the software. Bob would say a few words, and I'd start to think how good shareware was.

"So John Friend came along one day with a product that I considered the leading outline processor, PC-Outline. And it was shareware so we thought about it; and I guess, as you say, 'If you can't beat 'em, join 'em.' So we decided to acquire PC-Outline and keep it shareware. And be as enthusiastic about it being shareware as possible.

"Before we were even willing to mention the word shareware, we were looking for new avenues of distribution. We wanted to find a way to get software into the hands of as many people as we possibly could, as quickly as possible. PC Magazine did that, and shareware did it even more.

"So now we are very much into shareware. We're supporting all the customers John had, providing toll-free support 12 hours a day. We see strong sales for PC-Outline, rivaling all our other products which are not shareware. It also encouraged us to bring out a shareware version of Home Base, I suppose you'd call it a crippled version of Home Base. And we are considering a full version, an earlier version than the one we're now shipping, as shareware.

"I'm firmly behind it and will do anything I can to help the cause."

He went on to cover some of the issues facing shareware authors. One is the definition of freeware, shareware, crippleware, and beware. Also, he felt that initial screens should come up with the same wording.

Very few people who are using a copy know if their copy has been paid for. Perhaps they work for Ford or the Army, and their organization has bought a license to use the software. Maybe not. If not, he says it's a major problem.

From the consumer's side this is ideal, they can try the software for free.

"Of course there are disk distributors who advertise that they are the 'Robin Hood' of the software industry. You know, rob from the rich and give to the poor.

"In their catalogs, you'd think they'd say why they can sell you a disk of software for $5. They should say, 'These are shareware programs, and if you like them the author expects you to pay for them.' But no, it doesn't say that. It says $5 a disk. How can we do it? We make it up in volume.

"If you're an average consumer and you get this, you'll say, 'WOW! Here's PC-Write,' in my opinion a leading shareware product. The catalog gives it four stars and a cup, but nowhere does it say that if you like this program you get to pay for it. The manual costs money, the program costs money. I think that's a real, real problem.

"Just down the hall in Room 108, there's an organization handing out a paper saying they have public domain software, and lo and behold PC-Outline is here. PC-Outline never has been public domain. And some guy is going to go in there, pay his $5, and think he owns PC-Outline. He's wrong.

"I'm emotional about this because, (1) I'm in this to make money, and (2) I think this can hurt shareware.

"Does any author have any numbers about how many disks these guys sell versus the number of registrations that authors get?"

(No answers.)

At the end a member of the audience commented that every piece of shareware should contain a restriction stating that no one could sell copies. That way, copiers wouldn't be making money off the programmer's labor.

Bob Wallace (PC-Write), also in the audience, joined in.

Wallace: "I would never agree to that."

"But everyone's making money off the copies. They should at least be providing money back into a fund for the authors."

Wallace: "I totally disagree, completely disagree."

"One of the other things that scares me, I was on Genie and someone posted a message saying that Copyright means nothing because anyone can download the program free."

Wallace: "We get five to ten calls every day for help, and the people are really disappointed when we tell them that they really didn't purchase the copy when they got it from a copying service. And they are very irritated at us."

"Do you think the disk copiers are bad for the shareware industry?"

Wallace: "Well, that's a real loaded question. I think that perhaps some people think that three registrations a day are better than nothing. I don't think they are hurting the industry per se. I think that it's misleading to have a book like this and not say at the beginning why the software is $5 per disk."

Then a shareware purchaser added that he was irritated that he had sent $49 to register his copy of PC-Outline as requested in the program, and had received along with the manual a bill from Brown Bag for an additional $40.

Afterthoughts

Shareware is a new industry that's searching for credibility. But even without, it's been very lucrative for the few who have learned how to effectively work in its environment.

For each person reaping six-figures, however, I've talked to dozens whose games, tutorials, or utilities are generating one or two checks a week. Many of these lesser lights are still working the "eight-to-five," but most are hopeful that the next revision will work the magic.

However, many more are burned out - tired of the long hours answering phone calls and letters and bummed that the thousands of copies they know are being used generate such pitiful returns. Getting into shareware is easy, a single upload. Getting out is another story.

■ ■ ■

Teaching Your PC To Listen

Cheap Four-channel Data Acquisition, Display, And Storage

You'll discover lots of pleasant little things if you hang around the PC long enough. Here, Russ turns the game port into a four-channel A/D converter. Certainly a very inexpensive way to monitor just about anything.

For some time now, I've been interested in interfacing computers to the real world (whatever that is). I previously worked with robotic arms and speech recognition systems to build a voice-activated computer/robotic work station for the IBM PC.

Recently, I've been considering how best to sense such things as light level, temperature, sound level, etc., with my PC. One of my objectives has been to use my PC for a home security system, and I didn't want to lay out hundreds of dollars for special electronics boards.

Additionally, at work, my project team has been working on a physiologic data acquisition and telemetry system for use in NASA's Space Shuttle. You may wonder what possible connection these two items could have.

It turns out that we were experiencing some delays in getting a data acquisition system for our lab's PC. We needed to monitor blood pressure and ECG transducers. And we needed to do it right away.

One night, slaving over my hot PC at home, I realized that the kind of system I was considering for home security could be used to acquire biomedical data at work. The resulting data acquisition, display, and storage system for the IBM PC, that I call PC-DADS, is the subject of this article.

To use PC-DADS, your PC/XT system will need a game (joystick) port. The system can acquire up to four channels of data at a resolution of between seven and eight bits per channel.

The maximum data acquisition rate depends on your software. The rate ranges from about 30 samples per second for each channel (in a four-channel system running under interpreted BASIC) to 200 samples per second per channel (when the BASIC is compiled and linked with assembly modules).

The acquisition rates quoted are for my plain vanilla dual floppy 512K IBM PC running at 4.77 MHZ. See the discussion in the section on PC-DADS performance for more information.

The input voltage range for each channel is switch selectable to either 0 to 9 volts (with a primary seven-bit range of 0 to 2 volts), or -1 to +8 volts (with a primary seven-bit range of -1 to +1 volt).

The total parts cost for a four-channel system is under $15. You can purchase a game port board for about $25. If you have an AST Six-Pack Plus or equivalent (as I have), you can add the chips, jumper, and cable to install the game port for about $5. So you can have this system operational for less than $40, even if you don't already have a game port. *(Editor's note: Russ called and said he would be glad to make parts kits available for cost ($15) plus $2 postage. Contact him at his home address, listed above.)*

PC-DADS Hardware

The IBM PC joystick port, called the Game Control Adapter by IBM, is easy to transform into a simple data acquisition system. Four of the inputs on the port are usually used for joystick inputs (one for the x-axis and one for the y-axis on each of two joysticks). There are also four switch inputs used for the two fire buttons on each joystick.

A technical description of the joystick port appears in the PC and XT Tech Reference manuals. I warn you, however, that each edition of the tech manual differs a bit in its treatment of this port. I collected three or four manuals before I had the complete picture.

Also, I found a reasonably comprehensive review of the use of the joystick port to make measurements in an article by Michael Covington in the May, 1985, issue of *PC Tech Journal.* If you want to learn more about the basics of the port's operation and use, I suggest that you refer to Covington's article. (And check out PC Tech's reference manual.)

Each joystick consists of two 100K-ohm potentiometers at right angles to each other. The joystick position is sensed by measuring the resistance of each potentiometer. Because of the method used by the PC to measure the resistance, values of up to about 250K ohms can be measured by the joystick port.

The hardware circuit used in PC-DADS appears in Figure 1. The circuit in Figure 1 depicts one of the four identical circuits required for four channels. If you don't need four channels, you can build fewer than four circuits. That cuts the parts cost but you will have to modify the software. Oh well.

The circuit is an adaptation of a PC Oscilloscope circuit presented in Covington's article. My PC-DADS circuit uses a 12-volt supply for the quad operational amplifiers, rather than Covington's 9 volts.

I used 12 volts for three reasons. First, the circuit accepts wider range of input voltages. Second, 12 volts is available from the PC's supply. Third, a 12-volt supply makes it easier to implement the two-range feature (i.e., allowing one of the primary seven-bit input ranges to be -1 to +1 volt).

Resistors R4 and R5 should be

By Russ Eberhart
6770 Half Crown Ct.
Columbia, MD 21044

matched as closely as possible. The values of resistors R3, R6, R7, and R8 determine the range and zero-input reading of the device. The output of the device read directly from BASIC as STICK(n) (where n is the channel number 0-3) should be approximately 200-225 for an input of 0 volts (on the 0-9 volt range). From 18.5-20K should be about right for R3.

PC-DADS Software

Figure 2 contains the interpreted BASIC source.

It should be run under BASICA or equivalent on DOS 2.1 or later. This code supports four channels and a RAM disk (defined as drive C:). Drive C: is the data repository.

If your RAM disk is other than C:, you'll need to change lines 100, 101, 140, 640, 730, 840, 900, 930, 960 and 1030. If your drive C: is a hard drive, see the modifications at the end of Figure 2.

The PCDADS2C.BAS source and assembler routine (for high-speed data acquisition) is on the Micro C bulletin board (503) 382-7643. I compiled PCDADS2C.BAS using QuickBASIC version 2.0, and assembled SDQ.ASM with Microsoft's MASM version 2.0. I then linked them with the QuickBASIC linker, including Microsoft's PREFIX.OBJ.

At startup, both versions will ask you to connect the first calibration voltage. Whatever the voltage, it should be connected to all inputs. I suggest that 0 volts be used for the first voltage (ground the inputs). The second calibration voltage, again connected to all inputs, can be something like 3 or 4 volts. Use a digital voltmeter, if possible, to measure the calibration voltage(s). Accuracy is important.

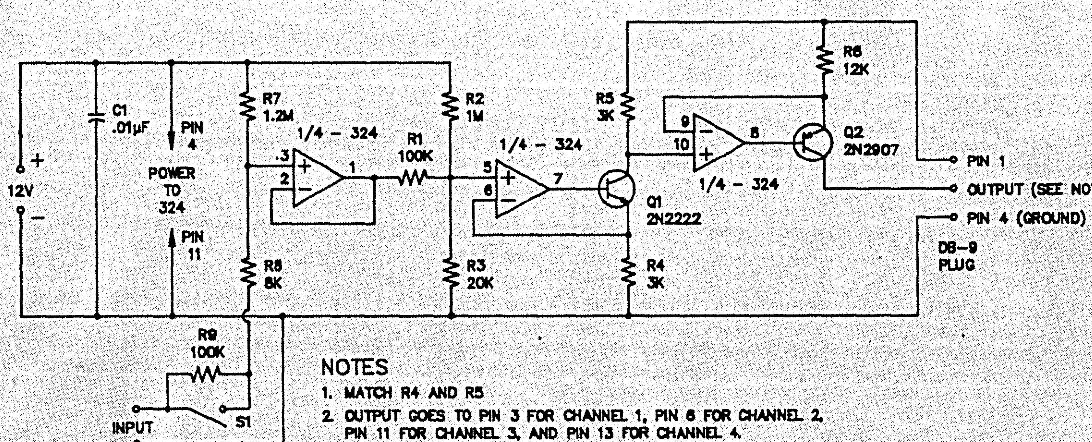
Once the program knows the calibration voltages, it computes offset and gain for each channel. These factors, along with a variable indicating whether it's in single or multiple trace mode and whether data is to be saved to diskette, are then passed to the assembly language routine.

After all this, you get the main menu. It gives you four choices.

The first choice is to operate PC-

(continued next page)

Figure 1 — One of four identical analog voltage scalers. You need one for each channel.



NOTES
1. MATCH R4 AND R5
2. OUTPUT GOES TO PIN 3 FOR CHANNEL 1, PIN 6 FOR CHANNEL 2, PIN 11 FOR CHANNEL 3, AND PIN 13 FOR CHANNEL 4.
3. SWITCH S1 CLOSED FOR 0–2V RANGE, OPEN FOR –1 TO +1 RANGE.
4. SWITCH SIDE OF INPUT MUST BE POSITIVE ON 0–2V RANGE.

Figure 2 — Interpreted BASIC routine to input and store analog data.

```
100 REM PC-DADS Ver. 1.1 for two floppy drive system with NO hard disk
101 REM PCDADSIC.BAS 11/22/86 Russ Eberhart
110 CLS:KEY OFF:ST=0
120 PRINT:PRINT "PC DATA Acquisition, Display and Storage (PC-DADS) v. 1.1"
130 PRINT:PRINT "Copyright 1986, JHU Applied Physics Laboratory":PRINT
140 PRINT:PRINT "NOTE: This program version requires RAM disk on drive C"
150 PRINT        "      Data are recorded on diskette in drive B:"
160 PRINT:PRINT
170 PRINT "Connect first cal. voltage to all input channels, then press key..."
180 IF INKEY$="" THEN 180
190 R1(1)=1/STICK(0):R1(2)=1/STICK(1):R1(3)=1/STICK(2):R1(4)=1/STICK(3)
200 PRINT:PRINT "First set of calibration readings taken"
210 INPUT "Enter value in volts of first calibration voltage";V1:PRINT
220 PRINT "Connect second cal. voltage to all input channels, then press key..."
230 IF INKEY$="" THEN 230
240 R2(1)=1/STICK(0):R2(2)=1/STICK(1):R2(3)=1/STICK(2):R2(4)=1/STICK(3)
250 PRINT:PRINT "Second set of calibration readings taken"
260 INPUT "Enter value in volts of second calibration voltage";V2
270 DV=V2-V1
280 FOR I=1 TO 4:F(I)=(R2(I)-R1(I))/DV:OF(I)=R2(I)-(V2*F(I))
300 NEXT
310 PRINT:PRINT "PC-DADS System is now calibrated"
320 PRINT:PRINT "                 *****     MAIN MENU     *****"
330 PRINT:PRINT "   1. Four-channel data display, single screen at a time"
340 PRINT:PRINT "   2. Continuous four-channel data display"
350 PRINT:PRINT "   3. Acquire and record data set, display on screen;"
360 PRINT       "      saving to disk is then optional for each data set"
370 PRINT:PRINT "   4. Acquire and record data sets at selected intervals,"
380 PRINT       "      save to disk automatically"
390 PRINT:PRINT "   5. Execute SHELL command to operating system; type EXIT"
400 PRINT       "      to return to PC-DADS Main Menu"
410 PRINT
420 INPUT "Enter choice:";MC
430 ON MC GOTO 460, 450, 610, 880, 860
440 GOTO 320
450 ST=1
460 PRINT "Channels one and two will appear in the lower half of the screen,"
470 PRINT "Channels three and four will appear in the upper half."
480 PRINT "A beep will signal that a trace is complete; if you are in single"
490 PRINT "trace mode, you may then press any key to start another trace."
500 PRINT "Press CRTL-BRK to exit program."
510 PRINT:PRINT "Press any key to start data display..."
520 IF INKEY$="" THEN 520
530 GOSUB 2000
550 FOR I%=40 TO 639
555 REM ENTER THE FOLLOWING FOUR LINES AS ONE:
560 PSET(I%,187-8*((1/STICK(0))-OF(1))/F(1)):
    PSET(I%,139-8*((1/STICK(1))-OF(2))/F(2)):
    PSET(I%,91-8*((1/STICK(2))-OF(3))/F(3)):
    PSET(I%,43-8*((1/STICK(3))-OF(4))/F(4))
570 NEXT
590 BEEP
600 IF INKEY$="" AND ST=0 THEN 600 ELSE 530
610 CLS:PRINT"Press any key to start taking data..."
620 IF INKEY$="" THEN 620 ELSE 630
630 CLS:PRINT"Taking data..."
640 OPEN "c:data.dat" FOR OUTPUT AS #1
650 PRINT #1, F(1);F(2);F(3);F(4);OF(1);OF(2);OF(3);OF(4)
660 T1$=TIME$
670 FOR I=1 TO 600
680 PRINT #1, STICK(0);STICK(1);STICK(2);STICK(3)
690 NEXT
700 T2$=TIME$:PRINT #1, DATE$,T1$,T2$
710 CLOSE
720 GOSUB 2000
730 OPEN "c:data.dat" FOR INPUT AS #1
740 INPUT #1, F1,F2,F3,F4,OF1,OF2,OF3,OF4
760 FOR I%=40 TO 639
770 INPUT #1, S0,S1,S2,S3
```

(continued from page 49)

DADS as a four-channel data display system, displaying one screen of data at a time. The screen of data can be examined at leisure, and even dumped to a printer providing that GRAPHICS.COM or equivalent has been loaded by the operating system. You can grab another screen of data by pressing any key (except escape). Escape takes you back to the main menu.

The second choice is like the first except the screen is continuously updated. This mode most nearly emulates the performance of a four-channel oscilloscope. It's very useful for hands-free operation during circuit analysis.

The third choice allows a data set to be taken on each channel, and simultaneously displayed on the screen and written to ramdisk. The software is set so that 600 data points (for each channel) are taken per data set. This provides exactly one screen full of data. It is an arbitrary number, and can be changed by modifying the assembler routine.

The fourth choice displays data sets, and simultaneously records them to ramdisk, continuously. Look out, this one eats up ramdisk space at a rate of about 1.5K per second. As above, datasets consist of 600 points per channel. The user is asked to press any key to begin taking the first data set. From then on the process is automatic and inexorable.

Data is taken and displayed (Main Menu options 1 and 2), or taken, displayed, and written to ramdisk (options 3 and 4) as fast as the software allows. This data taking rate can easily be slowed down by adding delays to the routines. I'll talk about the data rates shortly.

PC-DADS Performance

Let's look at two aspects of performance. The first is resolution and dynamic range. The second focuses on sampling rates and the effects on sampling rates of compiled BASIC versus assembler.

The circuit has been designed to be "usable" from 0 to about 9 volts input. The word usable is in quotes above because, by the time you get to 8 or 9 volts, the resolution is only 0.4 or 0.45 volt (about 5%). As a contrast, at around .5 to 1 volt input, the resolution is approximately 10 millivolts (about 1%).

So it follows that the circuit provides maximum resolution at low input voltages. In fact, a full seven bits of resolution is available for inputs ranging from

Figure 2 — Continued

```
775 REM ENTER THE FOLLOWING FOUR LINES AS ONE:
780 PSET(I%,187-8*((1/S0)-OF1)/F1):
     PSET(I%,139-8*((1/S1)-OF2)/F2):
     PSET(I%,91-8*((1/S2)-OF3)/F3):
     PSET(I%,43-8*((1/S3)-OF4)/F4)
800 NEXT
810 CLOSE:BEEP
820 LOCATE 1,1:INPUT"Do you want to save to disk?";SV$
830 IF SV$="Y" OR SV$="y" THEN 840 ELSE 610
840 SHELL "copy c:data.dat b:data.dat"
850 CLS:GOTO 320
860 SCREEN 0:SHELL
870 SCREEN 2:CLS:GOTO 320
880 CLS:INPUT "How many points per data set";PPS
890 PRINT:INPUT"How many data sets per day";DSD
900 OPEN "c:dataset.dat" FOR OUTPUT AS #1
910 PRINT #1, F(1);F(2);F(3);F(4);OF(1);OF(2);OF(3);OF(4);PPS;DSD
920 CLOSE:CLS
930 SHELL "copy c:dataset.dat b:dataset.dat"
940 CLS:PRINT "Press any key to take first data set"
950 IF INKEY$="" THEN 950 ELSE 960
960 CLS:OPEN "c:dataset.dat" FOR OUTPUT AS #1
970 TS$=TIME$:TS=TIMER:PRINT"Taking data..."
980 FOR I=1 TO PPS
990 PRINT #1, STICK(0);STICK(1);STICK(2);STICK(3)
1000 NEXT
1010 TE$=TIME$:PRINT #1, DATE$,TS$,TE$
```

Figure 3 — Output counts vs input voltages.

Input Voltage (volts)	Output Counts -1 to 1V	Output Counts 0 to 2V
-1.0	222	n/a
0.0	120	210
1.0	84	115
2.0	64	79
3.0	52	61
4.0	44	49
5.0	38	42

0 to about 1.9 volts, or from about -1 to +1 volt on the -1 to +1 primary scale. See Figure 3 for readings of STICK(n) versus input voltages. The calculated output reading is proportional to 1/STICK(n).

It would seem reasonable for many applications, then, to use an input range of about 0.0 to 2.0 volts, which gives slightly better than seven-bit resolution. This is why the above range is called a "primary seven-bit" range.

Sampling Rates

The subject of maximum possible sampling rates has proven to be interesting, although relatively little time has been available to explore it. I originally tested the PC-DADS software in two forms:

(1) The source code as listed in Figure 2 (using the BASICA interpreter), and

(2) A compiled version (EXE file) of the source code, compiled using Microsoft QuickBASIC version 2.0.

The maximum sampling rate (for the four-channel system above) achievable using the source code and interpreter, was about 32 samples per second for each channel. The maximum rate using the same code compiled with the QuickBASIC Compiler was about 40 samples per second.

Talk about disappointed! Maybe I was naive, but I sure expected a better improvement in speed with compiled code. (I also compiled with IBM's version 2.0 BASIC compiler with the same disappointing results.)

At this point, I'd like to state that I consider programming in assembler just slightly more fun than a root canal. I did not, however, see any way around writing an assembler routine if I was going to acquire 100 samples per second - the absolute minimum to look at ECG signals.

So I did. And it worked.

The sampling rates in both the display only and the display and save modes are about 200 samples per second for each channel when using the routine written in assembler. Saving to ramdisk seems to reduce the rate by only about 5%. (I used my trusty stopwatch to time these things, so all measurements are approximate.)

Final Notes

If you're using interpreted BASIC, you must call STICK(0) to get any data no matter how many channels you're using. The values for STICK(n), n=0 to 3, are all obtained with the STICK(0) request in interpreted BASIC, and any subsequent request will only obtain the value received with the most recent STICK(0) call.

One other interesting note relates to the factor of 0.72 in line 302 of the "master" basic program, PCDADS2C.BAS. When I first compiled, assembled, and linked the composite program that consists of BASIC and assembler parts, the calibration appeared to be off (you guessed it, by a factor of 0.72). The interpreted BASIC routine gave me the right answer, but not a compiled-assembled-linked version.

For some reason you get a different reading depending on whether you use interpreted or compiled code to read the port. (They were wrong even when I emulated the interpreted BASIC with my assembly code). It was a darn sight easier to put in a correction factor than perform major surgery on the assembler routine (remember the root canal).

If you would like more information on the game port, I suggest that you refer to Peter Norton's *Programmer's Guide to the IBM PC*, and to a book by Lewis Eggebrecht entitled, *Interfacing to the IBM Personal Computer*.

Acknowledgements

The portion of the work leading to this article that I did at the JHU Applied Physics Laboratory was supported by the National Aeronautics and Space Administration's Ames Research Center. Mr. Herb Finger is NASA's Program Manager for this effort. I'd also like to thank Scott Foerster, Professor of Math Science at Howard Community College, Columbia, Maryland, for his inspiration and assistance. ■ ■ ■

86 WORLD

Rearranging SORT

By Laine Stump
Development Foundation of Turkey
Tunali Hilmi Cad. 22
Ankara, Turkey

Laine modifies SORT (for all Turkish sorts) and starts linking ASM files into Logitech's Modula-2. But, of course, all that has to wait while he describes the muted pleasures of Istanbul.

Charming as it is (and always will be no matter what happens), Istanbul must be one of the noisiest places in the world. And as if the normal, unavoidable sound of screeching tires and boat horns wasn't enough, Istanbullus seem to have a passion for making as much extra noise as possible.

Witness little Ömer with the 10 liter olive oil can and the stick, banging away in rhythm with some unknown prehistoric chant only ancient insects would understand or figure out. And Mehmet with his new second hand Murat (Turk Fiat) parked out at the curb, rapping the engine up until you're sure it's going to explode; just making sure his friends know he has a car.

And Osman, the service station attendant, who at 6 a.m. is sure that the only way his customers will hear him is if he shouts at the top of his lungs. And just when you thought you could get some sleep, Ergun the policeman comes out on the night beat, blowing his giant whistle to make sure that all those potential criminals know they are being searched out (and that they know exactly where the searcher is).

The poet Orhan Veli knew way back in 1949 what I know today: Istanbul is a city of sounds (and smells, too, especially in summer, but I won't belabor you with those right now). His Istanbul and my Istanbul are somehow related, but I wonder if he would believe the incredible din now if he hadn't drunk too much raki and stumbled into a hole to break his neck so many years ago.

Istanbul was last week. Now I'm back to the grind, and I will be for at least another week. Then maybe, if I'm lucky, I'll get to go south for a few days and bake on the beaches. (That was week before last, down in Bodrum, the biggest little Greek-looking town in Turkey).

X-16s

For now I've got my hands full worrying about 82 newly-arrived X16 motherboards. The biggest worry is where I'm going to put them, since the cabinets, power supplies, 3.5 inch disk drives, etc., that go with them are coming from Hong Kong. And I just learned that they were sent by boat!

Not only that, I've got to spend some time learning how to write device drivers for Microsoft Windows, using SYMDEB as a manual (i.e., I'm reverse engineering). It seems that when I load Windows (to run my new PageMaker program), it puts in its own keyboard driver which replaces my beautifully crafted multi-layout Turkish keyboard driver.

PageMaker can't display all of the Turkish characters, so I guess I shouldn't care too much. For the moment. But just wait until I talk them out of a font editor program for their font files (or information on how to write my own). The multi-lingual problems just never seem to stop.

Speaking of Multilingual

I ran across an interesting problem in my work at the Redhouse Press in Istanbul last week. Actually I had run across the problem before, but never got around to looking into it until last week. (I have become an expert at procrastination since I came to Turkey. When in Rome...)

The people at Redhouse wanted to make indexes and other lists for several books they are publishing. I recommended that they just type in the text, then sort it with the SORT program included with MS-DOS.

SORT worked just fine for this purpose, except that it did weird things with Turkish characters. For instance, Ç and C were treated as equal by SORT, but in Turkish C always comes before Ç. Other letters not originally present in the IBM character set were just shoved out to the end of the list.

This was not acceptable, but I didn't really feel like writing my own sort program to solve the problem. I started thinking about it, though, and realized that if SORT is able to treat Ç the same as C, it MUST have a sort order table somewhere.

Modifying SORT for Turkish

After first telling the people at the office that it would be completely impossible to fix (I love surprises), I renamed SORT.EXE to SORT.BIN, and loaded it up with SYMDEB. Sure enough, starting at address 592 (hex, for those of you who don't yet speak my language) was a table that mostly followed the ASCII order, with upper case letters in place of lower case, and normal characters in place of characters like Ç, Ü, and the other members of the IBM extended character set.

Knowing that the table existed, I just made a list of the complete Turkish alphabet, assigned each one a number, and placed the proper number at each character's position in the table. Then I wrote the file back out to disk. I didn't even change the checksum in bytes 12-13 and it worked! Now all their sorting is "otomatik." Happy day.

Don't forget to rename the EXE file to a different extension before you read it with SYMDEB (or DEBUG). If you don't rename it, the debugger will load up all the segments and separate them in preparation for running, which *isn't* what you want. You want to read in the load image, not the run image.

Cool New Program

Remember a few issues ago when I talked about Logitech Modula 2/86 and said that one of its big disadvantages was that it couldn't easily interface to assembly language modules? Well, the days of those problems are over now that I have EXE2LNK, a program from Blomsoft of Maalov, Denmark.

EXE2LNK

EXE2LNK is a utility program that transforms an EXE file of assembly language procedures (assembled with MASM) into an LNK file that can be linked into your Modula programs just like any other LNK module. EXE2LNK works with versions 1.1 and 2.0 of Logitech Modula 2/86.

Other than following the register passing and stack conventions outlined in the Logitech manual, all that is required to code an assembly language module suitable for linking with your Modula programs is a small table at the beginning of the EXE file that tells:

- How many bytes of static data space the module needs,
- How many procedures are in the module, and
- The addresses of the procedures, including an initialization procedure.

You have to follow a few simple rules.

- There must be exactly one segment declared in the module.
- Exported procedures must be listed at the beginning of the EXE file and must be declared with the FAR attribute.
- Any number of local (non-exported) procedures may be declared. These procedures should be declared as NEAR.
- The first word in the EXE file must contain an "m2". This is how EXE2LNK makes sure it isn't messing with the wrong file.
- There can't be any code that leaves unresolved addresses to be resolved at load time (i.e., no FAR calls within the module).

Other than that, you can do pretty much what you want: call DOS, write directly to the screen, beep the stupid bell. The manual even says you can reserve space for local data in the same segment as the code, but if you're planning on doing any multitasking, I wouldn't do that. Much better to allocate data on the stack like a good boy (remember my pains of last issue dealing with the non-reentrancy of Turbo Pascal's I/O library).

Of course, the usefulness of EXE2LNK can really be seen when a piece of code needs to be optimized for speed. It can also be useful if you want to trim down the size of your library; first write everything in Modula, then go back later and write assembly language modules that fit the specifications of the DEF files.

But then, I'm sure most of you are convinced of the usefulness of being able to mix assembly language and Modula modules. And those of you who aren't won't allow yourselves to be convinced by me anyway. So why don't we talk about a few details of EXE2LNK.

Example

Figure 1 is a DEF file for a simple example that I more or less lifted straight from the manual. It defines two procedures for export - one of them a function with a single value parameter, the other a normal procedure having two parameters; one a value, and the other a variable parameter. count1 and count2 are examples of exported static data.

Figure 2 is the ASM file to go with the DEF. It contains the required header as described above, as well as the code for the two procedures declared in the DEF file, and an initialization procedure. Both procedures do basically the same thing; they return a value from a given position in a table of numbers. Proc1 returns the number as a function result. Proc2 returns it in a variable parameter.

I was going to write my own examples for this review, but the example provided encapsulated so many variations in such a small space that I was hard-pressed to come up with something better. Here we have, in one simple little bit of code, examples of passing by value, passing by reference, returning as function result, and use of static data. Also notice the use of MASM's STRUC directive to allow easy addressing of data on the stack as well as the static data.

In order to create a working module, you just have to execute the following commands:

```
M2C EXAMPLE.DEF
MASM EXAMPLE;
LINK EXAMPLE;
EXE2LNK EXAMPLE
```

I learned nearly everything I needed to know about writing assembly language modules just from this one ex-

(continued next page)

Figure 1 — DEF file for EXE2LNK example.

```
DEFINITION MODULE Example;

    EXPORT QUALIFIED count1, count2, Proc1, Proc2;

VAR
    count1 : CARDINAL;
    count2 : CARDINAL;

PROCEDURE Proc1 (index : CARDINAL) : CARDINAL;

PROCEDURE Proc2 (index : CARDINAL; VAR value : CARDINAL);

END Example.
```

(continued from page 55)

ample. Which brings me to the next subject: documentation.

The Manual

The manual is just nine pages long, which may seem inadequate. Operation is so simple, though, that nothing more is really needed. I understood everything I needed to understand from the list of rules, instructions for running, and the example module.

There are a few details that could have been included (and that I'm still not sure about). For instance, is it possible to declare and use more data space than the space required by the EXPORTed variables declared in the DEF file? I think it should be, but I haven't had time to experiment yet. Also, they didn't mention how to call one of the EXPORTed procedures from one of the other EXPORTed procedures. (But I know how! Declare a dummy NEAR label, push CS register, and call dummy label. See Figure 3).

Otherwise, I was immensely pleased with the manual. It's small enough that I can slip it in the back cover of my Logitech manual, and I don't have to go digging through mounds and mounds of paper to find something. On the other hand, it has everything I need to know to run the program.

Problems

As with any program, there is a rough edge or two that could stand polishing.

For starters, EXE2LNK doesn't know anything about the environment variables used by Modula to tell where DEF and SYM files are located (M2DEF and M2SYM). That means you have to move the SYM file (created by compiling the DEF file with the Modula compiler) into the same directory as your ASM and EXE files. It would be much more organized if the SYM and DEF files could be put into their own directories with the rest of my SYM and DEF files instead of cluttering up my ASM directory.

Another problem is that, although you can EXPORT procedures and data from your assembly modules, EXE2LNK doesn't know how to IMPORT. This is probably a bit much to ask, but I have this vision of slowly converting entire programs from Modula into optimized assembly language, leaving the Modula source as documentation.

And it isn't clairvoyant! No ice-maker attachment either! Can you believe it? When are these guys gonna learn???

Figure 2 — ASM file for EXE2LNK example.

```
CODE     SEGMENT
         ASSUME  CS:CODE

Link     EQU     6                   ; CS, IP and BP

Example  STRUC                       ; structure to address exported statics
count1   DW      ?
count2   DW      ?
Example  ENDS

Param1   STRUC                       ; structure to address Proc1 parameters
         DB      Link DUP (?)        ; account for CS, IP and BP
index1   DW      ?
Param1   ENDS

Param2   STRUC                       ; structure to address Proc2 parameters
         DB      Link DUP (?)        ; account for CS, IP and BP
value2   DD      ?                   ; VAR parameter - takes up 2 words on stack
index2   DW      ?                   ; 1 word for offset, 1 for segment
Param2   ENDS


         ORG     0
DATA     LABEL   WORD                ; at runtime 'm2' is replaced by
                                     ; segment address of exported data
         DB      'm2'                ; for id by EXE2LNK
         DW      SIZE Example        ; size of definition module data
         DW      2                   ; number of procedures
         DW      Proc1               ; procedure 1 address
         DW      Proc2               ; procedure 2 address
         DW      Init                ; initialization code address

Table    DW       2,  3,  5,  7      ; some table
         DW      11, 13, 17, 19
         DW      23, 29, 31, 37
         DW      41, 43, 47, 53


Proc1    PROC    FAR                 ; Proc1 code
         PUSH    BP
         MOV     BP,SP
         MOV     DS,DATA             ; static data segment
         INC     DS:[0].count1       ; INC(count1)
         MOV     BX,[BP].index1      ; get index parameter
         SHL     BX,1                ; make word offset
         MOV     BX,Table[BX]        ; function result is returned in BX
         POP     BP
         RET     SIZE Param1-Link; return and pop parameters
Proc1    ENDP

Proc2    PROC    FAR                 ; Proc2 code
         PUSH    BP
         MOV     BP,SP
         MOV     DS,DATA             ; static data segment
         INC     DS:[0].count2       ; INC(count2);
         MOV     BX,[BP].index2      ; get index parameter
         SHL     BX,1                ; make word offset
         MOV     BX,Table[BX]        ; move table value to BX
         LDS     SI,[BP].value2      ; get address of VAR parameter
         MOV     DS:[SI],BX          ; store table value
         POP     BP
         RET     SIZE Param2-Link; return and pop parameters
Proc2    ENDP


Init     PROC    FAR                 ; initialization code
         MOV     DS,DATA             ; static data segment
         MOV     AX,0
         MOV     DS:[0].count1,AX; count1 := 0;
         MOV     DS:[0].count2,AX; count2 := 0;
         RET
Init     ENDP

CODE     ENDS
         END
```

Figure 3 — Calling an EXPORTed procedure

```
Proc1Near   PROC    NEAR
Proc1Near   ENDP
Proc1       PROC    FAR

(( code of Proc1 ))

Proc1       ENDP

  (( in some other procedure
       in same module ))

      ;push argument onto stack
      PUSH    AX
      ;account for coming RETF
      PUSH    CS
      CALL    Proc1Near
```

The Envelope Please

Even with its two (three?) small problems, I think that EXE2LNK is one of the most useful additions to Logitech Modula that I have. Of course, I don't have Logitech's runtime debugger to try out yet, so maybe I don't have a fully-qualified opinion. Whatever the rating in relation to other programs, I recommend EXE2LNK to anyone who is as much of an assembly and Modula freak as I am.

Price is 550 DK, which was around $82 or so last time I looked. That's a bit expensive, but if you program for a living, it will likely pay for itself.

Blomsoft
Postbox 37
DK - 2760 Maalov
DENMARK

Coming Up Next

I know I keep promising it and I never deliver, but I *swear* this time that I'll be ready for issue #38. Time for my long-awaited treatise on the Great Desert Whale of Southern Africa. You have my word on it. And by the way, I think I can also make you a great deal on some Morrow Designs stock.

Ever Program On A Silver Platter??

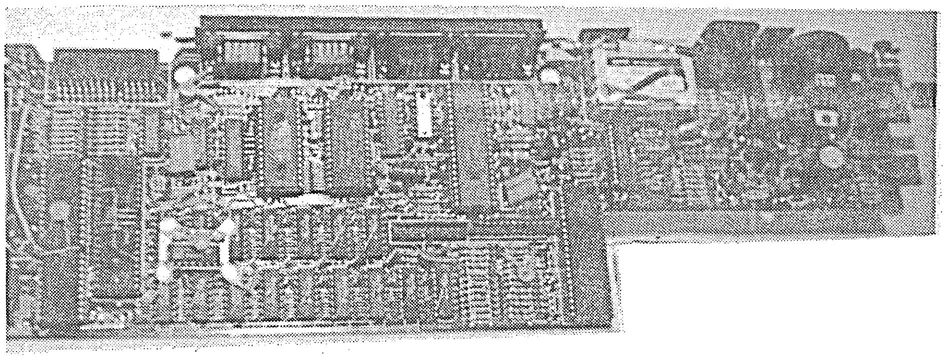
How much would you expect to pay for a 32 bit MC 68000 computer that's a mainframe condensed down into a keyboard? How about $389.00!!?? If it makes you feel any better simply add a zero to the price when you order! But that's actually our price!!! The most powerful computer money can ever buy is now the most inexpensive computer money can buy!!! So don't buy the name! Buy the power!! The power is **not** in the name!

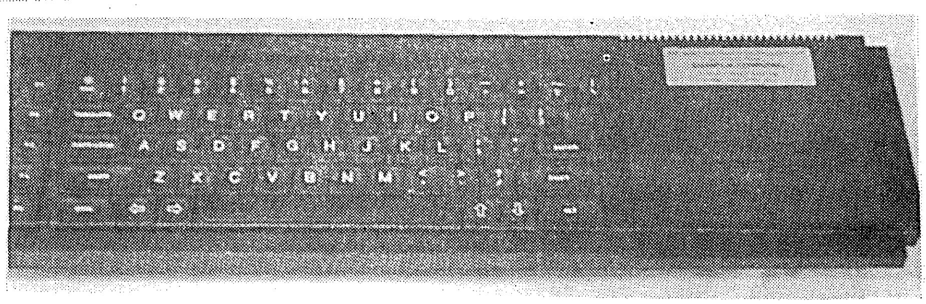
If **you** had the opportunity to work amongst Machine Code ROM Designers, VAX & UNIX wizards in a research laboratory, designing an MC 68000 based computer that's 2nd to none. . .

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!!





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

Technical Info & Assistance – –
Telex: 9102500026
Compuserve ID # 76625,2214

QLine BBS: 328–2919

Quantum Computing, Box 1280, Dover, NJ 07801

C'ING CLEARLY

Taking The Plunge Into C

By Ron Miller
1157 Ellison Dr.
Pensacola, FL 32503

Ron has finally reached star status. The way Borland's new C compiler has been selling (first month sales have set records), he expects a lot of C observers to become C hackers. Here's his description of C's hurdles and how to get over new-language fright.

Borland's Turbo C is finally out. And - if my acquaintances offer a representative sample - there are now *lots* of folks out there who are buying their very own compilers and are "at last ready, really ready, really really ready to learn C," as the formula goes.

Indeed, learning C, losing weight, and giving up tobacco may be the three most futile New Year's resolutions among PC hobbyists. If Borland handles the interface between compiler and editor half as well in C as they did in Pascal, you really don't have any more excuses.

Since I've been amazed to discover that this column is read by more people who are vaguely interested in C than those who actually use it, I'll offer some observations that might help you kibitzers get cracking.

Do Scary C Stories Keep You Awake at Night?

The first step, as we literary critics would say, is to de-mystify the topic. An aura of the abstruse and the exotic has developed around C: it's difficult to learn, incorrigibly obscure, and frightfully dangerous to use. Or: only people who sport beards, wear mountaineering boots, and speak it as their native tongue can ever be truly C comfortable. The fact that C is now fashionable adds to the put-off.

Horsefeathers. The unfamiliar always seems odd. I spent half a day the other week coercing a chained-together 135K accounting package in interpreted BASIC to compile under Microsoft QuickBASIC. I came away (partially successful, I might add) utterly in awe of you folks out there who can comprehend a run of unindented code, chock full of lines containing five or six unrelated statements separated by colons. No self-documenting subroutine names; no way to tell whether a variable is being used somewhere else.

The idea that complex applications have been written in such a stream-of-consciousness medium strikes me as a genuine triumph of the human spirit. This is something on the order of putting ships in bottles, or engraving the Declaration of Independence onto the back of a postage stamp, or manufacturing silk purses out of animal parts.

No doubt C can be written in an impenetrable manner. Have you ever seen Don Libes' annual "Obfuscated C Code Contest" in *Micro/Systems Journal*? Those perfectly wonderful horrors pay tribute not to C's obscurity, but to its freedom to be obscure.

C offers an intimidating wealth of operators and control structures, but you can begin with the "ifs" and "fors" that you know so well. Though you can certainly overstuff a line of C by trying to do too much at once, you don't have to write Baroque code any more than you have to write Baroque prose. I know: I tend to lapse into both. Indeed, my advice is that at first you force yourself to do one thing per line, just as in good old Pascal.

The very freedom that allows C to be written with such spectacular obscurity will permit you to write C after the pattern of BASIC or Pascal. "Goto" is even permitted. I promise not to tell anyone else if you don't. Then, as you get a feel for the natural way to use the resources at hand, your use of C, as C, will develop spontaneously.

The Real Difficulty With C

The real stumbling-block in C is not abstruseness but just the opposite, extreme literalness. C is a very physical, tangible, material language. To exploit the resources of C, and not just write transliterations from other languages, you must remain conscious of what a string looks like viewed under Debug and what incrementing a pointer really means.

You can program in BASIC, and to some degree in Pascal, without knowing about bits and bytes and storage spaces and memory allocations and instruction pointers. But in C, as in assembly language, these are everything. The syntax of C was devised to translate the immediacy of computer architecture into an efficient quasi-English. This allows you to avoid assembly language's busy-work: creating loops, keeping track of the stack, getting things into and out of the registers.

Nonetheless, C is little more than a veil through which assembly language's details often peek. Anyone who forgets this is soon reminded. Usually the reminder is a rogue array index or pointer that wipes out the operating system and locks up the computer.

Tell Pascal to do something quirky, and it will decide whether the request falls within the bounds of propriety. Tell C or assembler to do the same thing, and if they understand it they will do it.

In other words, it's not the *syntax* of C that is alien, but rather the reality its metaphors address. Its metaphors are constantly de-metaphorizing themselves into bits moving on and off and across registers and words pushing and popping the stack. (These, of course, are themselves metaphors, but I'll avoid getting too anti-metaphysical here.)

The intimate contact between the idiom of C and the hardware is the primary source of the power of the language, the reason for mastering it, and the main source of frustration.

Language Games

Pascal or BASIC or Cobol or Fortran (I don't know about FORTH) are essentially geometries, sets of rules which define the game and how it can be played. Translate your problem into the game's tokens, manipulate them according to the rule book, and out comes the answer.

For example, I was translating that BASIC package into QuickBASIC. I blithely treated strings as abstraction. I manipulated them without caring whether QuickBASIC uses one-byte or two-byte headers to store the string length. I didn't need to know whether there was an internal data format at all. For all I could tell, strings and integers and floating point numbers were all point sources, like ducks and quarks.

To convert the same program into C, I would of course be calling on my familiarity with the syntax of C and the fundamental string functions provided by the standard library. I would also find myself on at least two dozen occasions worrying whether I was overlaying the null at the end of the string, or whether I really wanted to begin some process at the zeroth or the Nth byte.

In other words, two dialogues would be going on in my head: one between me and high level C, played by the language's rules; and one between me and the hardware, as we both watch C carry out machine-level operations that would have been much more tedious to code in assembler.

All this sounds pretty intimidating, and I don't mean it to be. On the simple "Hello world!" level of operation, C acts like just a higher-level language, with clear structures and convenient devices for preventing side-effects. It's also breathtakingly fast.

C offers an intimidating wealth of operators and control structures, but you can begin with the "ifs" and "fors" that you know so well.

But the REASON for mastering C is that like assembly language, it lets you do darned near anything, prudent or imprudent. Such power requires a greater level of consciousness than other, more protective languages. And this higher consciousness, not some supposed difficulty in C, provides the challenge. But then heightened consciousness is what life's all about, right?

What Other Languages Should You Know?

So what's the best way to get into C, if the power of C seems appealing to you? They say that C ought not be anyone's first programming language, and undoubtedly they say right. First ride the bike with the training wheels. I propose two alternative routes, one through another structured language and one through assembly language.

If you are a Turbo Pascal programmer, relax. You already understand the divide-and-conquer approach of structured programming. Plus, you're already building your own subroutines and using labels to make the code self-documenting. Most of Turbo's extensions to generic Pascal have moved it toward the low-level leanings of C.

If you know how Turbo passes variables between routines, and occasionally use Pascal pointers and Turbo's absolute addressing, you are 75% of the way there. Even the syntax is similar. What's the difference between "begin" and "{" or "end" and "}" except the number of keystrokes? A C "structure" and a Pascal "record" are identical. What you will need to get used to is having to create procedures and functions that Turbo Pascal, with its Fibber Magee's closet full of run-time routines, keeps around for you, whether you want them or not.

Modula 2 programmers, from what I can see, have it even easier.

BASIC To C

Before playing with QuickBASIC the other week, I would have advised a BASIC programmer to learn Pascal first, and then go to C. But QuickBASIC and, I understand, Turbo BASIC permit named subroutines, line-number-free indented listings, multi-line if-then-else structures, private variables, and in fact structured programming. Why memorize another set of reserved words when you already know one perfectly adequate set?

I'd suggest you get one of these two inexpensive packages, grab the most complicated piece of spaghetti code you've ever written, and clean her up. Spend a month or so writing every piece of code you need to write, and a few you don't, in the style prescribed in the manuals, and you'll be ready to dive into C. But it won't be as easy as coming from Pascal (you'll have to learn structures and pointers).

Assembly Language To C

If you do much work in assembly language, you're home free. You already think like a C programmer. (To be honest, I'd say it was vice-versa.) You're used to the distinction between direct and indirect addressing. You have divided your code up into routines and given them names. You've tested loops for completion and called the operating system explicitly. You know about locals, externals, publics, and linking. You can just lean back and enjoy the fun of

(continued next page)

(continued from page 61)

letting the compiler patch in routines that you would have written on your own.

Interpreters And Bourbon

From brief experience, I'd argue against the use of C interpreters as transitional devices. They look seductive, with their promise of immediate feedback and line-by-line help. But Borland's C is surely going to provide error-pointing with interpreter speed. It will cost 25% of what an interpreter costs, and yet you will be working with a real compiler.

The very line-by-line orientation of interpreters moves the focus of programming away from the overall design of the whole toward the molecular structure of the individual expressions. Surely the greatest lesson taught by structured programming is that strategy, not tactics, is the heart of the game.

Worse than that, the one interpreter/incremental compiler that I played with forced me to learn another word processor and a very, very full set of metacommands. As I told the manufacturer (they were trying to coax a review out of me), if I wanted to learn another language, it would be Italian - and I'd go off and read Dante on the side. Or failing that, I'd swing for FORTH.

Getting folks into C by making C feel like BASIC is like mixing Coke with bourbon. You don't get to enjoy either one.

Books On and Projects In C

Okay. Now you've decided you have adequate preparation and want to push onward to C itself. What books do you buy? Many people seem to believe that if they can only find the right book, the clouds will part, the sun will shine, and all their loops will be finite ones. Life isn't nearly that kind.

We learn a language by using it rather than by memorizing the contents of grammar books. When I resolved to learn C, I bought a couple of books; and I was disappointed by their two extremes - jargon without communication and affability without content.

It didn't really matter, however, because I also bought a compiler, sat down at my old Kaypro, and wrote little routines. I then gave myself a nice low-level project (a disk, memory, and file editor). That way I learned C the only way anyone learns C, by solving problems.

I can't say I've read lots of books on C, and therefore can't offer a consumers' report thereon. When I get stuck on a technical detail, I tend to reach first for Stephen Kochan's *Programming In C* (Hayden: 1983), though that may be more for his fine appendices than for the text.

A moderately experienced Turbo programmer could - and should - go directly into Kernighan and Ritchie, which is a *fine* book, humane and delightfully written. It is directed, however, more toward the compiler-builder than toward the inexperienced programmer. Nowhere else, however, do you actually get a feeling for the choices that went into the syntax of the language. And for me, that is essential. I want to know *why*. *How* comes from staring at the screen and imagining those little bits dancing in my head.

In an ideal world, I'd run a training regimen for C programmers like so:

(a) Master Pascal as a first language.

(b) Read Kernighan and Ritchie.

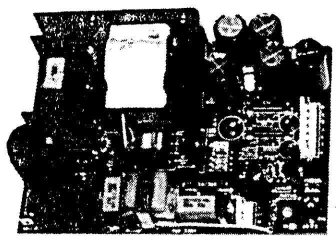
(c) Pore over assembly language listings in the PC magazines and do their tricks over again, with style, in C.

Good luck, and for heaven's sake don't experiment on a RAM disk unless you like writing code over and over again. From scratch. ■■■

SWITCHING POWER SUPPLY IDEAL FOR DISK DRIVES

50 WATT
+5V 4 AMP
+12V 3 AMP
4" x 6" x 1¾"

$29⁹⁵

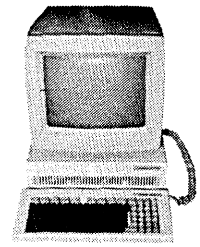


CORVUS CONCEPT WORKSTATION

- 6800 Based System
- 512 Memory
- Connects to Omninet
- 132 Column Monochrome Display
- Display in Vertical or Horizontal Format

USED WORKING CONDITION

$295⁰⁰



CORVUS MIRROR SERVER

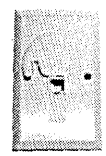
VCR HARD DISK BACKUP FOR CORVUS OMNINENT

NEW!!! **$25⁰⁰**

CORVUS Utility Server Printer Interface For Omninet

Includes Manual & Software

NEW! **$35⁰⁰**



4 PIN MODULAR WALL JACK WITH STRAIN RELIEF

10 for $4⁹⁰ Case of 325 for $99⁰⁰

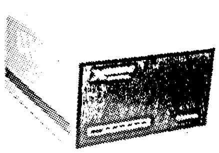
CORVUS SOFTWARE

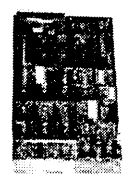
FORTRAN, PASCAL & MUCH MORE! CALL!!!

XEBEC 'ACCESS' 10MB EXTERNAL HARD DISK

- IEEE-488 (GPIB/HPIB) Interface
- Attractive Enclosure 12" x 6" H x 10" W
- Linear Power Supply and Fan
- 10 M Tandon Drive
- Brand New

$289⁰⁰

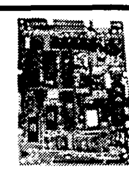




QBUS SASI HOST ADAPTER

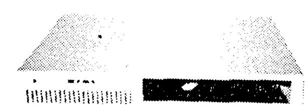
- Made by XEBEC
- Brand New

$35⁰⁰



XEBEC 1410 SASI HARD DISK CONTROLLER

$99⁰⁰



HALF HEIGHT EXTERNAL DRIVE ENCLOSURE

- Attractive Low Profile Chassis 19x15x2¾
- Fits nicely directly under PC
- Standard IBM Colors
- Bezel for 5¼" and 3½" Drive
- 60 Watt P.S. w/DC Fan and Drive Cables

$89⁹⁵/$29⁹⁵
complete chassis only

XEBEC IEEE-488 HARD DISK CONTROLLER CARD

$99⁰⁰

INTEL SBC 86/30

$395⁰⁰

TANDON 10 MEG HARD DRIVE

$119⁰⁰

WITH CONTROLLER $179⁰⁰

SHUGART SA801R 8" FLOPPY DRIVE

- 600K SSDD
- Still Sealed in Factory Boxes

$149⁰⁰

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.

CULTURE CORNER

A Second Opinion On:

In-House Experimental Verification Of Nonconservation Of Parity And Quantum Mechanical Tunneling Of Macroparticles

By Irv Mullins
Rt. 1 Box 312c
Talking Rock, GA 30175

Forward

I bought your Micro Cornucopia because on the cover it said it was about mouse control. We've got a real problem with mice in the corn around here, and I was hopin to use those PC mouse drivers to drive the mice away.

After readin the artikles, I think your goin about it the hard way. It'd be a sight cheaper to get a cat.

Imagine my surprize to run across an artikle on Quantum Mechanics by Trygve Lode in your magazine - I didn't think small farmers would be much inerested in Quantum Mechanics. Bein an expert on the subjek myself (1), I immedyately grabbed the implycations of it all, and decided that what was needed was a little more field work to cornfirm his hypothysis.

Bearin in mind Prof. Lode's warnin about the importance of bein as far as possible from any physics laboratories, I decided to do the research out at cousin Luther's place. You couldn't get no further than that and still be in the continentul U.S. of A.

Experiment 1: Nonconservation Of Parity

With a modest grant from Pabst (meanin a few six-packs of Blue Ribbon), me and Luther set out to gather a datum or two on the subjek.

Right off the bat we ran into trouble. Couldn't find no black socks to cornfirm the Parity 'speriment, and wadn't sure white socks would work. Only guy around who wears black socks is Caleb the undertaker, and he was usin his.

Closest thing we could find was a pair of cousin LuBell's panty-hose (2). We run 'em thru the nearest macroparticle dehydrator/storage-cylinder accelerator until we run out of quarters. Followin each run, both me and Luther counted how many legs was on the panty-hose. Ever time we both got two 'cept once (3). We also got some stares from the ladies in the Launder-mat.

Based on this, I just cain't cornfirm Prof. Lode's findins on parity. Sorry bout that.

Experiment 2: Macroparticle Tunneling

When Luther pointed out that we still had approxymately several Blue Ribbons left, we retired to his front porch to ponder Macroparticle Tunneling. I right quick showed him how personal observation tends to support the Lode theory. Fer example: Cal (4), Luther's blue-tick hound, was lyin under the house. Luther allowed as how, earlier in the day, Cal had been lyin in the kitchen. As no one can actually remember seein Cal move to the lower energy state (5) under the house, or for that matter ever move much at all, this strongly suggests that Cal's mollycules musta migrated through the floor.

The fact that it was a hot afternoon and Cal had reached higher than ambient thermal energy probly provided the impetus required for Quantum Mechanical Tunneling, as suggested by Prof. Lode. In further support of the theory, Luther noted that Cal seldom if ever disapeared from the kitchen on cold days.

As a corollary, LuBell pointed out that thermal underwear, bein warmer than reglar skivvies, always migrates to the bottom of the drawer by late fall.

Inconclusion

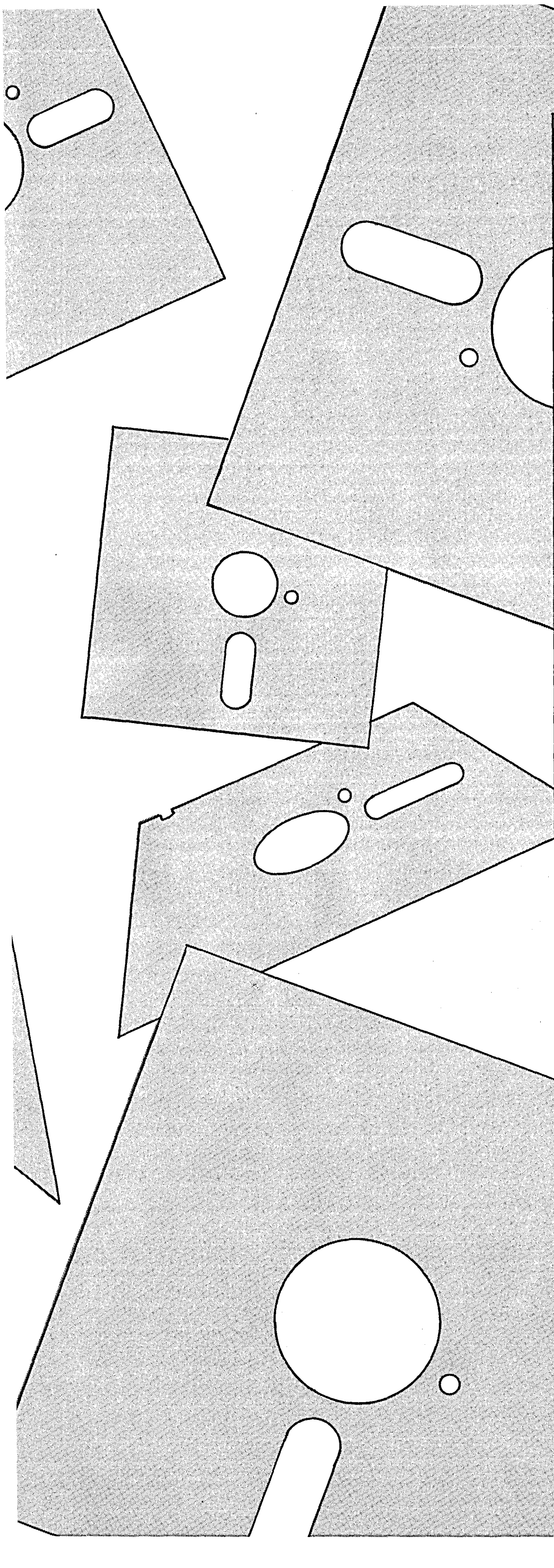
What we couldn't evalyate was whether Macroparticles invaryably migrate to a lower energy state. LuBell knows a guy named Mac who moved to Mississippi, which is just about the lowest energy state we could think of, but we need more research befor we can veryfy this hypothysis, and as long as it's this hot, we ain't settin foot off this porch. (6)

Sincerely, your friend in science,

Leeward P. Sailors

Notes

1. Hav'n seen the videotape from Volkswagen six times down at Leroy's garage.
2. Don't know 'bout the panty-hose, but LuBell's average mass is pretty signiffycant.
3. After Luther had killed the first six-pack, he counted three, but I was unable to verify his count.
4. Short for Calcitrant.
5. Cal is generly in a state of low energy.
6. Lest of course we run out of Pabst.



MS-DOS UTILITIES

Welcome to public domain software for MS-DOS. This is software that the Micro C staff (and readers) have found we can't do without.

We've written some of the software ourselves, the rest has been carefully selected from the thousands of public domain and shareware programs in the Micro C library. We think you'll enjoy these special programs as much as we have.

Available in 2 formats:
360K MS-DOS (5 ¼") **$8.00**
720K MS-DOS (3 ½") **$8.00**
Micro Cornucopia Subscriber special rate **$6.00**

#MS1
Essential Utilities

This is it—the essential utilities disk for copying, transferring, viewing, squeezing, unsqueezing, finding, and organizing files.

SWEEP allows wildcard tagging and mass file copying, jumps, relogs drives, and lots more.

LU, LDIR—A complete Novosielski library utility, LU creates a library file of files.

WHEREIS—This is one of the niftiest 2K programs in the public domain. Lets you find files in subdirectories. Very handy for keeping track of those files that try to get lost.

SQUEEZE/UNSQUEEZE—Complete file squeezing and unsqueezing utlities let you conserve disk space.

WASH-Forerunner of SWEEP, WASH is a menu-driven file utility that views files very quickly. It isn't as flexible as SWEEP, but it's faster.

LS—Written in C (includes source), LS is a UNIX-style directory program written by R. Edward Nather.

BACKSCRL—A bi-directional scrolling utility, BACKSCRL buffers screen scrolling so you can recall with a few simple keystrokes data that's been saved from the screen. Read BACKSCRL.DOC for a thorough explanation of setup.

#MS2
Cheap Assembler
Disassembler, RAMdisk

CHASM—Written by David Whitman, CHASM is a subset of MASM and fits into 64K. It's good for writing short subroutines to call from BASIC, or for just learning 8088 assembly-language.

It allows you to define labels, but doesn't support macros.

ASMGEN—A disassembler written by J. Gerbach and J. Damke, ASMGEN will generate 8086, 87, or 88 code. It's MASM-compatible, and output can be directed to the console or to a disk file. Handles up to 64K files. Includes a long doc file.

MEMBRAIN—Creates a file named 'MEMBRAIN.SYS', a DOS device driver for a RAM disk drive.

FSPOOL—This neat little program redirects output to a diskfile. Very handy for creating a file from DEBUG.

UNWS-A menu-driven BIT7 of the DOS world. Resets bit 7 (which has been set high in some characters in WORDSTAR), turning your WORDSTAR doc files into standard ASCII files.

DEBUG.DOC—A file of tips on using DEBUG. Good for the beginner.

***.ASM**—These source files for SDIR, RAMDISK, and UNWS will really help you get your feet wet in assembly language programming. Or if you already know the ropes, you can improve these programs.

#MS5
Util, ST, PC-WINDOW, Z

Z.EXE—Move about hard disk directories.

PROTECT—Make sure that your .exe and .com files cannot be erased by the erase command.

UTIL—moves files between subdirectories, sorts directories, redefines the keyboard, lets you type directly to your printer, pipes output, and lots more.

PCWINDOW—A semi-sidekick, PCWINDOW combines notepads, multiple timers, ASCII reference code and other features.

DOSEDIT—A simple editor for DOS commands.

#MS15
Utilities

Here are utilities to make your life more efficient.

DESKMATE is a 'Sidekick' lookalike with notepad, calculator, calendar, and access to DOS commands.

EASY-ZAP, a disk inspector, will allow you to examine and modify sectors. It works on hard disks as well as floppies.

UNERASE is the essential utility to save you from your own recklessness. If you've unintentionally erased a file, UNERASE will undo the damage. Handy.

#MS22
Dynamite Utilities

We've included some genuine gems on this disk.

V20-80—CP/M emulator software which enables IBM PC compatibles (i.e. personal clones) equipped with the NEC V20 CPU (See Micro C Issue No. 29 for details) to run 8080-coded CP/M programs.

LIST—A dynamite TYPE lookalike (the best we've used), with line up, line down, page up, and page down in 16 variable colors.

SPEEDUP speeds up and quiets your drive by changing the step rate from 8 milliseconds to 4 milliseconds.

TURBO HELP—A memory resident help facility to help you learn (and use) TURBO Pascal. It's ready at a keystroke in an attractive window.

INLINER—Translates your assembler mnemonics into TURBO Pascal inline code. Written in Turbo Pascal; includes source.

LASTCOM—TURBO Resident program to save your last 10 MS-DOS commands. Includes source.

The **SECRET** Group (MD, CD, and RD)—lets you make, remove and find secret files.

#MS25
Ultra Utilities

The three Ultra Utilities programs will allow you to map disks, unerase files, format non-standard disk, interrogate sectors, and much, much more.

This is a very useful set of utilities (a poor man's NORTON). Many hours of work and frustration can be avoided by learning them, so have at it.

#MS27
System Primer

We think this disk will satisfy at least some of your curiosity about MS-DOS systems programming. We've included lots of assembler source code, so dig in.

SCAV finds and marks bad blocks on both floppies and hard disks.

WHEREIS finds files anywhere within a directory structure.

DIAGS—Special serial, parallel, and video diagnostics for the PC. Use this excellent program to explore your system.

ASYNC—Loadable asynchronous device driver for MS-DOS.

LPTX intercepts BIOS interrupt 17, the line printer interrupt. It redirects the output of LPT1, LPT2, or LPT3 to a disk file. All three may be active at the same time.

DOS1, ROLLDOS1 & 2, DRIVER & DRIVER1—Stop wondering how device drivers really work and explore these tutorials. Good examples of character device drivers and de-bugging techniques.

STUFIT stuffs your least used files into the inner tracks of the disk. This frees the outer tracks for work space and speeds access times considerably.

#MS36
General Utilities

BATMAKER helps create .BAT files. Perfect when using FIND on all .TXT files, for instance. Very handy.

BWVID lets you see what is happening on the screen when you have a color graphics card (CGA) and a monochrome monitor.

CED is called a Command line EDitor but it's far, far more than that. Includes macro definitions, control of DEBUG, repeating and editing of previous commands, etc.

DEBUG.DOC is a simple but very handy quick reference guide to DEBUG.

EXPAND and SHRINK are detab and entab utilities.

PC-STAT—Reports system information—memory available, drive status, etc.

PC-TEST is similar to Norton's speed test, but its test takes longer and it doesn't report such wildly optimistic speed figures.

POPALARM is really neat. It's a memory resident alarm clock that reminds you to do what you'd otherwise prefer to forget.

RECALL remembers the last 50 DOS commands. Commands may be edited and/or reexecuted.

REMIND—This is a daily black-book that stores its data on disk.

SCR*.*—Utilities for creating batch files which incorporate screen images. This is a great extension to MS-DOS batch capabilities.

FILTERS—The remaining files are classics from the Software Tools book. One of the real attractions of these filters is that they come with assembly language source.

#MS37
Disk Utilities

COVER prints out directories in compressed format to be pasted on floppies.

CRC67—Finally, an MS-DOS cyclic redundancy checker (CRC) that works (Fast!). Checks CRC values for files against a previously recorded list of CRC's.

DISKORAY checks floppy rotation speed and allows stepping of the head.

DISKPARK parks the heads of all hard disks in your system using the innermost track.

DISKWIPE—Be careful. This completely erases a disk, including the formatting.

FDATE allows editing of the time and date stamp on DOS files.

FILES—A very complete directory program.

MOVE2—Intelligent COPY routine.

REFRESH rerecords data on a disk. It does 12 retries on reads and 2 on writes so it may be able to recover those "bad" sectors.

SDIR—Version 5.0 of the super directory program.

SST—Just what every busy hard disk needs. SST reorganizes files into contiguous sectors on the disk. This really speeds up disk accesses.

TIMEPARK parks the heads on a running hard disk after a user specified amount of time without accesses.

WD*.*—Everything you always wanted to know about Western Digital's WD1002S-WX2 hard disk controller. Also information on optimizing its performance with the Seagate ST225 drive.

WHEREII searches for one or more files through all the directories on a specified drive. Supports wild-cards.

For Technical Help:
(503) 382-8048 (9 a.m.—noon, PST, M-F).

Technical Tips

EXEC Problems

I was very interested in Laine Stump's discussion of running programs from within other programs (see Micro C issues #31 and #32). This was just the capability I needed for a Turbo Pascal update of VFILER that I've been working on.

I tried the technique on an NEC APC and a Compaq 386 and it worked great. But several IBMs got lost returning from the child program. I had to resort to the Big Red Switch to get things going again.

It wasn't too hard to find out what was going wrong. In describing function 4Bh (in *Programmer's Guide to the IBM PC*), Peter Norton says: "Warning: The Load Or Execute function clobbers most of the registers, including the stack registers, SS and SP. These should be preserved before and restored after using this function."

Clearly Laine's assembly code version didn't do this and I suspected Turbo's *MsDos* procedure didn't either. Without storing the return address for the calling program, there was no way to get back to it.

What I needed was a new *MsDos* procedure that would save and restore SS and SP. The easiest way to get it was to locate and modify the code in Turbo. I used Turbo's *Ofs* function to find the code. Since *Ofs* only works on user defined procedures, I had to be a little tricky. I wrote a program which did nothing but print the offset of a procedure that included an *MsDos* call.

Then I ran the program from DEBUG, started unassembling at the offset that was printed, and there was the code I wanted. Just as I suspected, Turbo's *MsDos* didn't save the stack.

I added two lines to save SS and SP before the INT 21h call and two more to restore them afterwards. Then I assembled the new code with MASM and converted it to a COM file. By declaring it an external procedure and using it in place of *MsDos*, I can now successfully run child programs from within my Turbo programs on any system.

George L. Florman
4106 Carmen St.
Torrance, CA 90503

Identifying Intel Microprocessors

Rather than writing software using only 8086/88 instructions, it is very useful to allow execution of 80186, 80286, and 80386 code on systems with these processors. A general method for doing this tests for the type of processor and branches to processor-specific routines. This allows a program to take full advantage of each processor's capabilities while retaining the ability to run on 8086/88 systems.

One method for deciding which processor is running is to examine the flag register. While undefined on all processors, bit 15 of the flag register does take on different values. A "one" identifies the 8086/88 or 80186/188 processor. A zero means we're looking at either an 80286 or 80386.

Bits 12 - 14 are used only in Protected Mode. An 80286 in Real Mode can't set any of these bits. An 80386 can set all three although they have no effect.

The 8086/88 and 80186/188 flags behave identically. We'll have to look at how the shift instruction works to differentiate these two processors. The 8086/88 uses all eight bits of the CL register to hold the shift count. The 80186/188 uses only the lower five bits. So the 80186/188 uses CL MOD 32 for the shift count. This means a shift of 33 is the same as a shift of one for the 80186/188, while all those shifts on the 8086/88 guarantee a result of zero.

See Figure 1 for code which returns the processor type in the AX register.

The preceding was a condensed version of an Intel *TechBits* paper sent in by:

Pat O'Leary
Ardrew, Athy
County Kildare, Ireland

■■■

Figure 1 — Assembly language routine to detect and return processor type.

```
proc_type   proc        near

    pushf               ;save flags
    xor     ax,ax       ;clear AX
    push    ax          ;push it on stack
    popf                ;zero the flags
    pushf               ;try to zero bits 12-15
    pop     ax          ;recover flags
    and     ax,0f000h   ;if 12-15 are 1
    cmp     ax,0f000h   ;then processor is
    jz      is_0_1      ;8018x or 808x

    mov     ax,07000h   ;try to set bits 12-14
    push    ax
    popf
    pushf
    pop     ax
    and     ax,07000h   ;if 12-14 are 0
    jz      is_80286    ;processor is 80286

is_80386:               ;else it's an 80386
    mov     ax,386h     ;return 386
    jmp     done

is_80286:
    mov     ax,286h     ;return 286
    jmp     done

is_0_1:                 ;it's 808x or 8018x
    push    cx          ;save it
    mov     ax,0ffffh   ;set all AX bits
    mov     cl,33       ;will shift once on 8018x
    shl     ax,cl       ;or 33 times on 808x
    jnz     is_80186    ;nonzero bits mean 8018x

is_8086:                ;else it's 808x
    mov     ax,86h      ;return 86
    pop     cx          ;restore CX
    jmp     done

is_80186:
    mov     ax,186h     ;return 186
    pop     cx          ;restore CX

done:
    popf                ;restore original flags
    ret

proc_type   endp
```

PASCAL PROCEDURES

Starting The $6 Scanner

By John Paul Jones
6245 Columbia Ave.
St. Louis, MO 63139

John called me a few weeks ago, just bubbling with news about his new project. He was building a scanner out of a dot-matrix printer and $6 worth of parts.

"You mean a hobby scanner, right John?"

"No, this one creates real PC Paintbrush files."

I'd been interested in scanning for quite a while, especially since I'd begun using Ventura publishing. Ventura accepts PC Paintbrush files (among many) so when John mentioned PC Paintbrush, scanning, and manipulating graphics all in the same sentence, I really salivated. We hope you'll soon be enjoying the results of John's new project. As for Pascal and Modula? He'll be using these languages to develop the software.

In this column he begins the scanner and continues his introduction to Pascal and Modula-2.

The freehand drawing programs which are available in the PC world provide the tools, for those with some talent, to generate surprisingly good images. I am fascinated by these programs but have a problem - I can't draw worth a hoot. What I need is some way to get existing images into the computer. Once the image is in the proper form for the painting program, I feel confident that I'll be able to butcher it quite nicely.

Image and document scanners are available, but the least expensive of these is in the $1000 range, very difficult to justify for "fun." Much of the cost of the document scanners is in the mechanics needed for paper handling. This is also the reason that a home-brew version would be a *big* project.

An ad for a relatively inexpensive ($150) document scanner which attaches to the print head of a dot matrix printer made me realize that the mechanical problems had already been solved.

To define the project goals, I'm shooting for:

1. Low cost
2. Optical sensor for print head attachment
3. Interface between the sensor and the PC
4. Software to
   a. Scan a picture
   b. Display the captured image
   c. Process the image (contrast alteration, edge detection...)
   d. Save to disk
   e. Print an image
   f. Convert to paint program file format

Over the next few columns, I'll be developing the project in detail. Along the way we'll learn a bit about image capture (rasterization), image display details, image processing, dithering (pseudo gray scale on a straight mono display), and whatever else time and energy permit.

The first three entries in the goal outline turn out to be the easy part. The sidebar discusses the hardware needed to implement them.

The software will, of course, be developed in a modular fashion, primarily in Modula-2, but I plan to show how it can also be done in Turbo Pascal.

The first step will be to control the printer so we can rasterize the result. Next will be the capture of the scanned data and, finally, creation of the routines for displaying the data.

From this modest base, we'll add functions for image enhancement, printing, true gray scale (which will require a color graphics board), and some other fun stuff. Stay tuned for further developments.

Meanwhile I'll continue describing the similarities and differences between Pascal and Modula-2.

Assignment

The assignment statement is the most fundamental executable statement in both Pascal and Modula-2.

```
variable := expression;
```

The assignment operator (:=) should be read "becomes" or "is assigned the value." It is *not* equivalent to the "=" in an algebraic equation. Remember that data in these languages is strongly typed and strict rules are applied for assignment compatibility. If type boundaries need to be crossed, it must be done explicitly with a type conversion function. Or in Modula-2, with type breaking.

```
charvar := chr(intvar); { Pascal type conversion }
charvar := CHAR(CardinalVar); (* Modula type
                                 breaking *)
```

Go With The Flow

Unless the program is otherwise directed, it executes its statements in order. Two types of instructions alter this flow, iteration (looping) and branching.

Loops

Pascal has three iterative statements, Modula-2 four. For all of the examples, the word "statement" represents either a simple statement, or for Pascal a compound statement, within a BEGIN - END pair. Modula-2 doesn't use the BEGIN but requires an END, even after a simple statement.

FOR

When you want to execute a statement a certain number of times, a FOR loop can be used.

```
{Pas} FOR index := start TO { DOWNTO } finish DO
                statement;
(*M2*) FOR index := start TO finish { BY bump } DO
                statement END;
```

The loop index can be any scalar type except REAL. The index variable cannot be modified by statements within the body of the loop; you cannot exit prematurely as you can in a language like BASIC.

In Pascal, incremental and decremental indexing are controlled by the keywords TO and DOWNTO. The increment is fixed at +1 or -1. Modula-2's BY option gives more flexibility since the index can be changed by any integer amount with each iteration.

When the value of the index goes beyond the finish value, the loop execution terminates. The control parameters are evaluated before each iteration so the loop can be executed as few as zero times.

WHILE

The other top testing loop is the WHILE loop.

```
WHILE BooleanExpression DO statement; { Pascal }
WHILE BooleanExpression DO statement END;(*
Modula-2 *)
```

The loop will be repeated as long as the boolean expression evaluates to TRUE. This has two consequences; first the expression must involve a variable, and second that variable *must* be altered within the body of the loop. As with a FOR, a WHILE can be executed zero times.

REPEAT

A repeat loop tests for completion at the end, and therefore will always be executed at least once.

```
REPEAT statement UNTIL BooleanExpression;
```

Syntax is identical for the two languages, the keywords REPEAT and UNTIL serve as BEGIN and END. Again, an infinite loop can result if the variable(s) in the boolean expression are not modified in the loop.

LOOP

Modula-2's fourth iterative statement is the LOOP.

```
LOOP statement END;
```

This minimal syntax is correct, but results in an infinite loop. A LOOP can only be terminated with an EXIT statement; flexibility is provided since multiple EXITs are allowed within the LOOP body.

```
LOOP
  .
  IF x = y THEN EXIT END; (* END for the IF! *)
  .
  IF ch = 'z' THEN EXIT END;
  .
END;      (* LOOP END *)
```

Branching

Two statements allow conditional execution of sections of a program. The IF statement is used to branch based on BOOLEAN expressions, and CASE branches based on a scalar expression's current value.

IF (Pascal)

```
IF BooleanExpression
  THEN statement
  ELSE statement;
```

(continued next page)

The $6.00 Hardware

The hardware needed to scan an image using a raster scan technique turns out to be relatively simple. The requirements are: first, a photodetector small enough to be mounted on the print head or print head carrier of a dot matrix printer; and second, a circuit to convert its analog output signal to digital.

There are many possible choices for the detector, from a simple phototransistor to the premium LED (light emitting diode) photodiode assemblies made by Hewlett Packard. For convenience and low cost (~$2), I am using the OPB708 reflective photosensor from Optron Technology. This device has an infrared LED and matching phototransistor, both focused about .2" in front of the device. See Figure 1.

The circuit in Figure 2 lets us get the sensor's data into the computer. R1 and R2 provide variable current drive to the LED; R2 becomes our coarse brightness control by varying the amount of light striking the image.

The light reflected from the image causes the phototransistor to conduct, generating a voltage across R3 proportional to the reflected light intensity. This voltage is isolated from the rest of the circuit with an op amp wired for unity gain, then amplified with a second op amp.

R4 allows adjustment of the gain of this second op amp between about 4 and 100. This is the master brightness control.

For maximum flexibility, the amplified signal could feed an analog to digital converter IC to give a range of shades equal to the A/D's range. An 8-bit A/D would give 256 distinct levels of brightness. The other extreme is to only provide a single output bit with any brightness half-scale representing white. Since the plan was to input the data through the four push-button inputs of a dual joystick port (I had one, therefore there was no additional cost), a middle of the road approach was chosen.

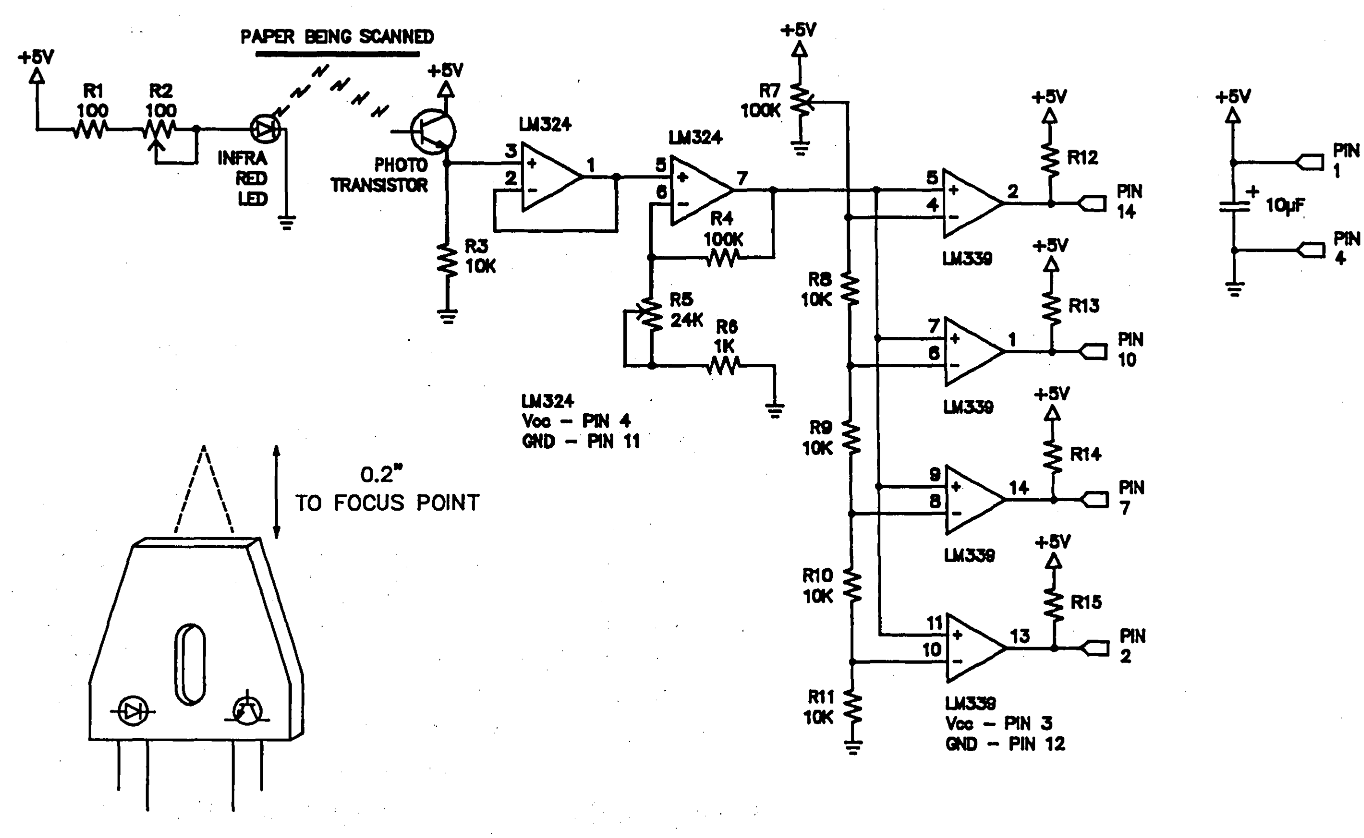
R7 and a multi-tap resistive divider (R8 - R11) provide equally-spaced reference voltages for four comparators. This provides five levels of brightness. Referring to Figure 2, LM339 pin 13 represents 0 or 1, pin 14 = 2, pin 1 = 3, and pin 2 = a brightness of 4. R7 gives us a contrast control. For single bit applications, use a single comparator and eliminate R8 - R11.

R12 - R15 will be needed if the input bits you use are not pulled up; the push-button inputs on a joystick port normally are.

At this point, the circuit has been breadboarded and gives a reasonable gray scale output. Not tested yet is the resolution of the scanned spot, or pixel size. If the active spot turns out to be large, an aperture may need to be put on the sensor input. Also, there may need to be some preliminary processing of input before it can be displayed.

This hardware is NOT suitable for scanning a color image. The LED/phototransistor pair are infrared devices, so a scanned color image would have its reds brighter and blues darker than perceived by the eye. I plan to photocopy any color pictures before scanning. This will introduce some brightness shift due to the copier's spectral sensitivity, but considerably less than that of the sensor.

Circuit of $6.00 scanner.



(continued from page 73)

The ELSE clause is optional; if it is absent, the statement immediately following the IF will be executed if the expression is FALSE. Since the statement after the ELSE can be another IF, you can use it to perform multi-way branching. Also, an IF after a THEN will give you nested conditional execution.

```
IF ch = 'a' THEN
   IF ch < M => 'z' THEN LowCount :=
succ(LowCount)
   ELSE NonLowCount := succ(NonLowCount);
```

Pascal is very picky about the construction of nested IFs. There are two rules. One, a semicolon can NEVER precede an ELSE since that would terminate the IF statement. The other rule is that an ELSE will always pair with the closest previous IF that lacks an ELSE.

You can construct some very complex decision structures with IFs. However, they can be incomprehensible if you aren't careful about formatting.

IF (Modula-2)

Modula-2's IF statement differs from Pascal's in two ways; it requires an END as terminator, and an optional ELSIF clause gives a more convenient way to construct complex decisions.

```
IF exp1 THEN statement;
   ELSIF exp2 THEN statement;
   ELSIF exp3 THEN statement;
   ELSE statement
END;
```

The only statement(s) executed will be those following the first TRUE expression. If none evaluate to TRUE, the ELSE will be executed. Only one END is needed for even a very complex IF statement. Each THEN is terminated by either an ELSIF or an ELSE.

An additional consequence is that semicolon placement is not as critical as for Pascal. Modula-2 won't mind an extra one here; or there;. (But the editor might.)

CASE

The CASE statement gives multi-way branching based on the current value of a scalar (non-REAL) expression.

```
CASE expression OF { Pascal }
   const1 : statement;
   const2, const3 : statement;
   const4 : statement
END;
```

```
CASE expression OF (* Modula-2 *)
   const1 : statement |
   const2..const3 : statement |
   const4 : statement
   ELSE statement
END;
```

The two significant differences between the languages are Modula's use of the vertical bar (|) to signal the end of a selected statement group and the optional ELSE clause (Turbo Pascal has this extension also). Modula-2 also supports a subrange constructor (c..cc), and since Modula-2 allows constant expressions in constant definitions, they are allowed here.

The first statement group which has a match to the expression value in its constant list will be executed. If none of the constant values match the expression in Pascal, none of the statement groups will be executed. For Modula-2, if this happens with no ELSE clause, you'll get a run time error.

To come...

Next time, I'll begin presenting the software associated with the scanner project. I'll provide an outline of the entire system, and begin development of some of the low level modules.

■■■

Turbo C Comes To Micro C

Hot Stuff!

Larry gives us a quick peek at Borland's newest release, Turbo C. Considering the response, it appears that everyone with Turbo Pascal is in line for Turbo C. Considering the quality of the manuals, it's obvious Borland expected as much.

A lot of software finds its way to the offices of Micro C - some interesting, some not so interesting. (It's hard to get overly excited about yet another "super" directory program.) One package we're all jazzed about is the latest offering from Borland International, Turbo C.

I've just started to use Turbo C and so far it looks great. I'll be giving an overview of the package here. For an impression of the user interface, take a look at Gary's *Tidbits* in this issue. He also gives the compiler a pretty good workout in the speed and smarts departments. So what do you get for your hard-earned shekels?

Library

Turbo C comes with an *almost* complete set of library functions. You'll find all the standard functions plus the following:

- Complete directory manipulation including a function to search the path for a file.
- Process functions which allow you to spawn and control child programs from within a parent program.
- Interface functions to take advantage of machine specific, low-level characteristics.
- A floating point emulation library. Turbo C can generate code for the 8087 math coprocessor. At run time, if an 8087 is present, it will be used. If not, the emulation routines take over.
- Conversion and manipulation functions for time and date.

What you *won't* find is any sort of screen-handling support. I can understand a lack of advanced graphics functions, but simple functions like clear screen and cursor control would have been nice.

Recently I had a call from a reader who runs Turbo C on a Kaypro 2000. His immediate solution to the lack of screen handling was to implement the ANSI.SYS codes in a series of functions. A good temporary kludge. I'm sure that graphics packages will appear soon. When Borland comes up with one, I trust it will be better than their somewhat funky Pascal graphics toolbox.

Low Level Goodies

Turbo C gives the programmer three methods for getting into low-level trouble. Through the use of "pseudo-variables," you can go in and muck with the entire register set (with the exception of the flags and the instruction pointer). For example, _AX gives read/write access to the AX register. Pseudo-variables and the library function geninterrupt make the use of interrupts simple.

The second method will have Turbo Pascal users smiling. In Turbo C Borland has done away with the dreaded Inline statement and implemented true inline assembler code. So instead of something like:

```
Inline ($31/$C0/     { xor ax,ax }
       $8E/$D8);     { mov ds,ax }
```

We have:

```
asm xor ax,ax   /* set ax to 0 */
asm mov ds,ax   /* set ds to 0 */
```

It's a bit annoying to have to preface each assembler statement with "asm" rather than use #asm and #endasm to delimit a whole block of code. But who's complaining - it sure beats machine code. Of course, separate assembler code can also be linked into a program. But for short routines, inline is the way to go.

Finally, Turbo C eases the job of writing interrupt handlers. Declaring a function to be of type "interrupt" automatically takes care of housekeeping chores like saving and restoring the registers and issuing an IRET on exit from the handler. getvect and setvect let you manipulate the interrupt vectors in low memory.

Utilities

Turbo C includes a command line version for all you UNIX types who blanch at the sight of an integrated environment. Certain things (like the inline assembler discussed above) require the use of the command line version. Several utilities support this mode of programming.

The preprocessor and linker do just what you'd expect. MAKE is a nice hand-me-down from UNIX. It allows you to specify exactly how your program files depend on each other. MAKE then recompiles only those files which have been affected by your last round of editing.

It seems like the only thing missing is a debugger. The integrated environment includes a "Debug" option, but it only manages error messages. You can't track individual variables, single-step through the code, or perform any other true debugger functions.

Miscellaneous

Turbo C generates six different flavors of executable code - all the way from Tiny to Huge. The idea here is to optimize the use of memory and pointers. For example, the Tiny model restricts all segment registers to the same address and uses only near pointers. So EXE2BIN can create .COM files out of Tiny programs. Use this model for Terminate and Stay Resident programs.

One of the most interesting features of Turbo C is its ability to link with Turbo Prolog modules. Letting the two languages talk to each other opens up endless possibilities. Gary will be paying

By Larry Fogg

special attention to this capability in future issues of Micro C.

Documentation

You'll notice a change in Borland's philosophy with Turbo C's documentation. Very few assumptions are made about the reader's knowledge of programming in general, and none about knowledge of C. You can actually learn C by working through the 300 page *User's Guide*. A lot of effort goes into comparisons of Turbo Pascal and Turbo C, which should make the Pascal folks comfy.

A second volume, the *Reference Guide*, contains descriptions of all library functions, error messages, C syntax, and more. I just love to see good documentation, and Borland has done a great job on theirs.

A Winner?

You bet. As with Turbo Pascal, Borland has thrown in everything but the kitchen sink. Sure it deviates from "the standard" here and there, but it's pretty close. And what's a little deviation between friends anyway. A good debugger and a few screen functions are the only additions I could ask for.

It'll be interesting to see how the public responds to Borland's latest. If initial orders are any indication, they have another monster on their hands.

Gary, Dave, and I will be playing with Turbo C a lot, so stay tuned for our ongoing reactions.

■■■

TIDBITS

Turbo Outstanding

By Gary Entsminger
1912 Haussler Dr.
Davis, CA 95616

Gary just got Borland's Turbo C and he likes it. He likes it so much that, well, I'll let him tell you. Meanwhile if you're comfortable with Turbo Pascal and would like to go to C, now there's nothing stopping you.

I've just begun to explore the C Turbo, but I'm already tickled-to-Betsy. It's quick out of the blocks, easy to use, and compiles and executes like a thoroughbred.

Those of you who cut your teeth on Turbo Pascal and want to try C will really enjoy this environment. It's as good or better than what you're used to - with built-in editor, pull-down menus, split screens, nifty windows for simultaneous error listings and source view, and options up the Hwangpoo.

During compile, a window pops up in the middle of the screen, showing the compiler's progress through include files, its memory consumption, and the numbers of warnings and error messages.

Another window pops up, and you browse the error list, select the one you want to correct, and then switch to the location of the error in the source code which has been idling in another window.

As Robin Williams would say, "This program is BAD."

Poetry In Pointers

I wanted to test Turbo C's speed and intelligence, so I sifted through my collection of benchmarks and found two striking ones by David Clark. I diddled with them a little and came up with the two programs in Figures 1 and 2.

The first, "deref.c," tests a compiler's skill at tracking pointers. This is apparently complex business for a compiler, since many wander off into the pointer depths while de-referencing (finding the contents) of pointers.

A pointer, you'll remember, is a variable that contains the address of another variable. They're useful for *accessing objects indirectly*.

For example, if var_1 is a char, then -

var_1 = *pointer_1

assigns the contents of whatever pointer_1 points to to var_1.

If the contents of whatever pointer_1 points to is another pointer (and assuming var_2 is also

Figure 1 — deref.c

```
#define LOOPS   (unsigned) 50000

struct cptr1 {
        char ********************ptr1;
        };
main()
{
unsigned i;
char test;
struct cptr1 ********************pointer;

        printf("%u loops \n",LOOPS);

        for(i = 0; i <= LOOPS;i++);
                test = ********************
                        (********************pointer).ptr1;

        printf("%cfinished\n");
        exit(0);
}
```

a char) -

var_2 = **pointer_1

assigns the contents of whatever the contents of pointer_1 points to to var_2.

(If this sounds complicated to you, imagine what it's like for a compiler.)

If we take this pointer to pointer stuff farther than anybody knows what to do with, into 20 or so levels, we can get a pretty good feeling for a compiler's taste for complexity.

The benchmark in Figure 1 is a program gone so far. I tried it with four compilers, and three (Eco, Microsoft, and Turbo) managed to wind through 50,000 loops of 19 indirections in a few seconds or less. I quit before they did.

In defense of Aztec (which quit before I did, at Level 7, announcing, "data type too complex"), the proposed ANSI C standard requires only six levels of indirection.

The complete deref.c is in Figure 1.

Savage Floating Point

The second program, savage.c, is a bruiser, testing the speed and accuracy of floating point calculations. Twenty-five hundred loops of -

a = tan(atan(exp(log(sqrt(a*a))))) + 1.0;

All four compilers completed the test accurately (with an error frequency of 1 part in a billion or better; see Figure 3), with Turbo C winning the race by a landslide (Figure 4). But note: Eco, Aztec, and Microsoft compilers are 1986 versions; new versions may be faster (although I doubt as fast as Turbo C). Microsoft 4.0 C'ers (or anyone else with a fast compiler), send us your results.

Meanwhile, it seems clear enough that C compilers on the PC have matured significantly in the last year.

Three cheers for C (and Borland).

References

- Kernighan, B.W. & D.M. Ritchie. 1978. "The C Programming Language." Prentice-Hall Inc., NJ.

- Clark, D.D. 1986. An inexpensive MS-DOS C compiler. Byte 11:307-314.

- Feuer, A.R. 1982. "The C Puzzle Book." Prentice-Hall Inc., NJ.

The Turbo PROLOG Toolbox

Turbo PROLOGers, don't be dismayed. Your programming world, too, has just gotten a notch or two more out-

(continued next page)

Figure 2 — savage.c

```
#define LOOP  2500

extern double tan(), atan(), exp(), log(), sqrt();

main()
{
int i;
double a;

        printf("start\n");

        a = 1.0;
        for(i = 1; i <= (LOOP - 1); i++)
                a = tan(atan(exp(log(sqrt(a*a))))) + 1.0;

        printf("a = %20.14e\n", a);
        printf("done\n");
}
```

Figure 3 — Accuracy results from savage.c

```
correct result = 2500

ECO's          = 2.49999999968640e + 03
AZTEC's        = 2.50000000025354e + 03
Microsoft's    = 2.50000000000118e + 003
Turbo's        = 2.50000000000118e + 003
```

Figure 4 — Benchmark results.

```
All compilers tested on an 8MHz PC Tech X16b,
Seagate 225 hard disk, Ompti-controller */
```

	ECO(3.01b)	AZTEC(3.20d)	Microsoft(3.0)	Turbo(1.0)
deref.c --				
com/as/ln	45 seconds	n	45 seconds	14 seconds
size	9K	*	6K	6K
execute	y	n	y	y
savage.c --				
com/as/ln	46 seconds	23 seconds	46 seconds	13 seconds
size	16K	11K	21K	20K
execute	150 seconds	120 seconds	84 seconds	42 seconds

(continued from page 81)

standing, thanks to Borland (the burro guy) and the release of the "first" Turbo PROLOG Toolbox.

A must-have (i.e., dynamite) accessory, intended to help you professionalize your programs, the Turbo PROLOG Toolbox answers these (and other) exacting questions. How do I -

- Create professional menus (pull down, line, bar, tree, & multi-choice)?
- Design slick EGA and CGA (but not Hercules) graphics (boxes, circles, bar charts, and pies)?
- Design screens?
- Transfer files between Turbo PROLOG and Reflex, dBASE III, Symphony, and Lotus?
- Generate a parser?
- Communicate with remote serial devices?

I've added the Toolbox (80 example programs, 8,000 lines of PROLOG source code, and a 350 page user's guide and reference) to my bag of tricks (it beats juggling).

If you've been struggling to find a use for this very logical language, or want to do more with your discoveries, do yourself a favor and check it out.

From -

Borland International
4585 Scotts Valley Dr.
Scotts Valley, CA 95066
(800) 255-8008

And that, friends, is Tidbits.

■■■

An Uninterruptable Power Supply For Your RAM Disk

By Walt Pfiester & Gary Schumacher
1 Skadden Terrace
Tully, NY 13159

This is a simple construction project that should keep your 5 Volts up even when your power is down. Built as a RAM disk backup, it's also worthwhile for any system which uses just a few watts of 5V.

You, too, can build your own battery backed-up power supply from easy to find parts. This 5 volt UPS (Uninterruptable Power Supply) delivers power to my RAM disk for over five hours after loss of 117 V AC. Now my data is secure despite what happens to the AC power.

Either the built-in battery or the 117 V AC mains will provide 5V at 750ma. An automatic battery charger keeps the battery up.

Background

I use a 1Mb MicroSphere RAM disk as drive A on my modified Kaypro CP/M computer. If you think Turbo Pascal compiles fast on disk, you should see it fly on a RAM disk! The RAM disk also saves wear and tear on my 800K floppies. Combine the RAM disk with ZCPR and a 7 MHz system clock, and you understand why I like the system.

Problems

Over the years, I've used several different RAM disk power supplies. Unfortunately, with most of the supplies I'd wind up getting file errors after I'd left them on for a few days. (I use CRC.COM to check the files against CRCKLIST.CRC, a list of the files' CRCs. Checking 616K of RAM disk files takes 36 seconds.)

The CRC errors appeared when I used a separate power supply for the RAM disk versus directly obtaining power from my Kaypro. Several of my friends were also experiencing the same anomalies using external power supplies.

Gary Schumacher designed and built this simple and inexpensive battery backup system. I haven't lost any files since.

Description

The main piece of hardware for this project is a single-unit combination transformer, rectifier, and filter originally made for Commodore. It's available from Radio Shack (Archer AC-DC adapter #277-1026) or Digi-Key for under $5.

It generates 9.5 VDC at 1.0 Amp - plenty of power to charge the battery and run the RAM disk. (Be sure to note that the outer shell of the unit's output connector is positive.)

Battery

For the battery, we chose the Radio Shack deep cycle starved electrolyte lead acid unit. It's designed for cellular radio applications and has a rating of 2.5 AH. With a full charge it runs my RAM disk for five hours.

If you chose a smaller battery (say 1 or 2 AH), you could probably sneak the entire supply inside the RAM disk enclosure or put the whole mess inside the Kaypro. In our case, we used Radio Shack cabinet #270-252 ($3.99).

We mounted the connectors on the rear of the cabinet. The switch and LEDs graced the front. The LM2935 fastened directly onto a piece of angle stock which we'd bolted to the frame. It was perfect because the metal tab on the regulator is supposed to connect to ground.

We used nicad batteries in an earlier model. They are available surplus from Marlin P. Jones Associates, Lake Park, Florida, 33403-0685. Part #Ni-2362 has two batteries. Split it in half for a stack that's fine for this project. Other sources include: Amateur Radio journals, surplus houses, and local parts houses. Gel cells work just as well as the nicads.

(Editor's note: Lead acid batteries work best in a constant trickle charge situation like this. Nicads slowly lose capacity if they're not deeply discharged regularly.)

Theory Of Operation

D3 and D4 form an OR gate. The voltage at the cathode of D3 is normally higher than that at the cathode of D4, thus cutting off D4. Under this condition, the LM2935 receives its input voltage from D3. When the power input fails, the output from D4 is higher than D3, and the LM2935 receives its input from the battery. Thus, at all times, the LM2935 has sufficient input voltage to maintain regulation. We used the LM2935 voltage regulator because its voltage drop is only 0.65v. Most series regulators have a minimum drop of 1.2v.

Switching time is not a factor - there is none.

We chose R3 to trickle charge the battery at 1/100 of its rating (2.5 Ah). R2 (in parallel with R1 and D1) is the fast charge resistor. If you don't need a fast charge, then eliminate D1, R1, and R2 and the switch.

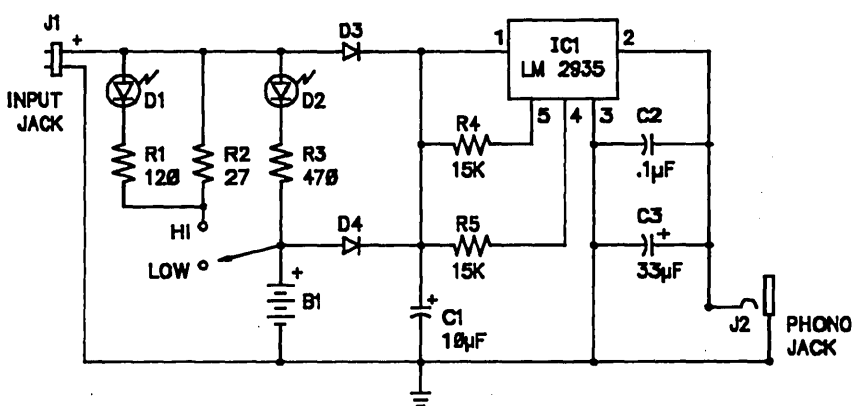
(For more sophisticated charger circuits, check out back issues of Ham Radio magazine.)

One last note regarding batteries: If you're planning to charge batteries rapidly, watch them carefully. If they get hot they could be damaged.

Finally, if the power goes down while a file is in system RAM, your data disappears, so save frequently to RAM disk. (I use a software macro for this, Save and Resume, generated with SmartKey.) That way all your data will be safely stored away. As they say, "Better safe than sorry."

■■■

Figure 1 — Power supply schematic and parts list.



J1	Input Jack 274-1565		
J2	RCA Phono Jack		
D1	Orange LED		
D2	Green LED		
D3	Shottky Diode 276-1165		
D4	Shottky Diode 276-1165		
R1	120		
R2	27		
R3	470		
R4	15K		
R5	15K		
C1	10Mf 25V		
C2	.1Mf 25V		
C3	33Mf 25V		
IC1	LM2935 (National Semiconductor)		
B1	6V 2.5 AH LED Acid RS	23-181	9.95
-	Cabinet	270-252	3.99

Not shown on Schematic: Input AC/DC Adapter: Radio Shack #277-1026

CPM Notes

The Z280 , AT Performance In An 84 Kaypro

Z280

The big news in the CP/M world is Zilog's new Z280. This processor executes all 8080 and Z80 instructions and has 45 new instructions plus additional addressing modes. Other characteristics of the chip include:

- Clock rate of 10 MHz or more
- Addresses 16 MBytes of memory
- 8-bit Z80 bus or 16-bit Zilog Z-Bus
- Paged memory-management unit with a 256-byte data and instruction cache
- Four-channel DMA controller
- Three 16-bit counter/timers
- Six-stage wait-state generator
- DRAM refresh controller
- 2.5 MHz UART

High Tech Research uses the Z280 in their new Ultraboard. The Ultraboard will run on the 84 Series CP/M Kaypros and the 10-83. High Tech plans a summer release and has a target price of less than $500.

This little gem promises to run at 12 MHz, which will yield an increase of about 10 times in processing speed. It will come with 1 MByte of RAM (expandable to 16 MBytes), configurable as a RAM disk by the user.

The Ultraboard also drives an external RGB monitor. High Tech replaced the Kaypro screen driver to keep up with the Z280's processing speed. The new driver runs at 18 MHz and has its own 256 byte Cache Memory for instantaneous screen updates (25 times faster than a standard Kaypro).

The Ultraboard should be comparable in speed to a standard AT. The question is whether there will be any software written to take real advantage of it. Standard CP/M software should run but it won't make use of the extra memory. We'll just have to see if developers come up with anything interesting.

For information on the Zilog Z280 contact:

Jim Magill, Richard Davies, or Tom Hampton
Product and Technical Marketing
Zilog, Inc.
210 Hacienda Ave.
Campbell, CA 95008
(408) 370-8000 or 370-5166

Those interested in the Ultraboard should contact:

High Tech Research
1135 Pine Street #107
Redding, CA 96001
(800) 446-3220/(800) 446-3223 in California

256K Upgrade Revisited

The 256K upgrade for 84 series Kaypros is alive and well on my machine. (See the Kaypro column in issue #34.) I fabricated a small printed circuit board for the extra chips and patched the signals to the bottom of the mother board via a 16-conductor ribbon cable. The patch cable alternated between signal and ground so I didn't have any problems with cross talk. I also replaced the 74HC04 with an HC14 in order to square up the clock.

I'm pleased with the setup and would be more than happy to provide the artwork for the board if anyone's interested. I made it double sided with targets for alignment.

R. Perfect
RD #4 Box 154
Reading, PA 19606

CP/M Kaypro Keyboards

A recent (May 1) direct mail ad from Kaypro announced that keyboards for their CP/M machines are available for only $20 plus $2 for shipping. You get everything except the case. This is a great opportunity for anyone with: broken keys, double characters, or the sticky habit of pouring Coca Cola Classic over everything on his desk. Order from:

Kaypro General Store
533 Stevens Ave.
Solana Beach, CA 92075
(619) 481-3958

David Shiller
Box 4859
Laguna Beach, CA 92652

Editor's note: I called Kaypro to check up on the keyboards and found out that other items will be available from time to time. Along with the keyboards, they now have those padded carrying cases for $20 and Juki sheet feeders for $130. We'll keep you posted as we hear of other deals. (And please let us know if you hear of some.)

Power Supply Feedback

Thank you and thanks to Larry Fogg for saving my Kaypro II-83! In your February/March issue (#34, pg. 48), Larry mentioned a fix for misbehaving 83 model Kaypros (resoldering the Molex pins on the power supply board).

My Kaypro used to take off into Never-Never Land with the drive motors turning on by themselves and the keyboard locking up. Then the keyboard began to lock up by itself (without notice, of course). In frustration I had to save New Word files after every paragraph. If I turned on the printer or even a fluorescent light, I was sure to lose my writing (even though I use a surge protector).

Well, your "fix" not only cured the lockups, but I swear the old Kaypro runs faster now. Along with saving an old machine from the junk pile, you've kept me (again) from buying an IBM.

Bob Katz
248 E. 90 St. #3B
New York, NY 10128

Editor's note: You'd run faster too if someone came after you with a hot soldering iron.

■■■

MICRO Ads

HOW TO PLACE YOUR AD IN THE MICRO C'S MICRO ADS

Print or type your message, or send camera-ready copy. Each ad space is 2¼" by 1¾". Include your complete return address and phone number. Rates are $99 1X, $89 3X, $79 6X per ad per issue. Send copy and complete payment (for all insertions) in check, VISA or Mastercard to: Micro Cornucopia, PO Box 223, Bend, OR, 97709. (Sorry, no ads accepted over the phone.) Materials due the 15th of the month, 2½ months prior to cover date. (Example: material received June 15 will appear in the Sept/Oct issue.) Ads received after the deadline will appear in the next scheduled issue. No ad will be published unless accompanied by full payment.

(continued from page 2)

Two people raised their hands.

I took a deep breath.

"How many of you are interested in desktop publishing or are planning to try it in the near future?"

All but about a dozen people raised their hands. (I estimated the audience at between 200 and 300.)

"How many of you are here?"

All but the same dozen raised their hands.

Finally I asked how many members of the group were earning their living as consultants, software writers, hardware engineers, or technicians. Two-thirds raised their hands.

My conclusion? Techies (when they're awake) are as interested in desktop publishing as real people. Sure, publishing packages are little more than crude word processors that integrate graphics with the text and make the text look good. But that's enough: especially when the text and graphics show up on the screen just like they'll appear in type.

You know how many manuals you work on. You know how many tables, schematics, drawings, and charts you include with your papers. And, you know the kinds of battles you fight with corporate graphics:

"Yes, the 8087 does look nicer with 36 pins, but when they designed it they felt it needed 40..."

If desktop publishing gives you anything, it gives you control. You don't need a masters degree in hot wax and razor blades to organize a page of information. And, it isn't a 15-hour cut and paste job to add two paragraphs.

Bookstores Like Desktop Publishing

I don't know how many bookstores carry technical books. I used to assume that most did. It appears most don't.

I just attended the ABA (American Booksellers Association) Convention in Washington, D.C. I went because I was certain that wherever there gathered thousands of bookstore owners, there would be hundreds of prospective purveyors of Micro C.

All I had to do was pick out name tags that had sounded like bookstore owners and then show them Micro C.

"I see you sell books."

"Actually my brother-in-law has a store, I just came to get these volumes autographed."

"I see you sell books."

"We don't sell books, we publish them. We used to sell books."

"I see you sell books."

"Oh yes, religiously. Do you sell Bibles?"

"I see you sell books."

"We sell knitting supplies, mostly. We're hoping to carry knitting books."

It was getting very, very frustrating. Even most of the distributors I found weren't much help.

"Yes, we distribute all kinds of computer magazines. Our five best titles cover Commodore and Apple."

Finally I spotted a name tag with the magic words "Technical Bookstore."

"Oh yes, we carry technical books. Which technical book do you publish?"

"Well, it's not a book, it's a magazine. Micro Cornucopia."

"We don't carry magazines."

"None?"

"Nope. We tried magazines, five Commodore and Apple titles recommended by a big distributor, but they didn't sell."

I had about given up trying to interest anyone in computer anything. I'd mention computer magazines in a crowd, and everyone would ignore me. I handed out copies of Micro C and reclaimed them from the trash bins.

I bought a Pepsi, then a Coke. Downed them both. Straight.

Then, in the middle of a crowd, I accidentally mentioned to the person next to me that we'd done the magazine with desktop publishing.

"Desktop publishing?"

It was worse than saying "EF Hutton" to a herd of steers. (I may be suffering from mixed brokers.) Anyway, everyone was instantly straining to hear my every word. Suddenly, so many people wanted copies of Micro C, I didn't have enough copies to go around.

Ah well, maybe someday they'll be interested in technical magazines.

For Newsstand Readers Only

Beginning with this issue there are more of you purchasing Micro C on the newsstands than subscribing. That's great, but it could be even greater: both for you and for us.

You can save time and money by subscribing (a one-year subscription saves you about 24%). As a subscriber, you'll also get your copy sooner and you won't risk missing out because the store didn't hold one for you. As a subscriber, you'll automatically receive the Micro C catalog of public domain (and shareware) software, and you'll get any special mailings, such as SOG (technical forum) info.

Meanwhile, we receive all of the $3 you pay for each subscription copy (we get only half of the $3.95 you pay on the stand), and it's cheaper for us to ship a copy to your home than to send it to your store.

So, just pop your name, address, and a check for $18 into the postpaid return envelope bound into this issue, and we'll take care of the rest. (We'll really appreciate you, too.)

The QL Responds

It didn't take QL fanatics long to respond to my editorial in issue #36. They hit the phones almost as quickly as the QL detractors.

The QL's rally squad started off with:

"You were way off base, the QL wasn't built by Timex, it was Sinclair all the way. You did a hatchet job on a wonderful system!"

The QL's detractors started off with:

"You were awfully wishy-washy when you described the QL's shortcomings. Looks like you're protecting someone."

The QL is running about 50-50, for and against. But it's doing much better than I am.

Obviously, if you get one of the little 68008-based systems, you're either going to become a QL crusader or a QL basher (FORTHers take note).

I got some information about the QL last fall ('86), and at the time it reminded me of the Commodore 128 or the Japanese MSX systems. Basically the QL looked like a fast TRS-80 with a fancy cassette drive. It was the $5 cassettes, the 8-bit data bus, the RF generator (you use a TV for a monitor), the calculator-type power supply, and the cheap keyboard that turned me off to the system.

(continued next page)

(continued from page 89)

If you added all the little extras it needed to make it workable - parallel port, RAM card, real-time clock, floppy controller and drive, decent monitor, power supply, and so on - you'd have spent the better part of $1,000.

However...

The fanatics make some good points. The U.S. version of QDOS is supposed to be clean. The system is cheap (how cheap depends on where you buy it). There's about 10 meg of public domain software, plus a growing number of commercial packages.

The QL's BASIC is similar to QuickBASIC (except QL BASIC still requires line numbers). The 68008 is compatible with the 68000 (68010, 68020...). And, the operating system appears to be quite powerful (I'm supposed to be receiving more information shortly). It's certainly much better than TRS-DOS, or MSX. (Some insist it's better than MS-DOS or CP/M.)

Because of the operating system, developers are stuffing QL boards into real cabinets and adding RAM expansion, a 68000 (or 68020) co-processor, a hard drive, floppies, a real power supply, an IBM-compatible keyboard, and additional ports. Obviously it's not a lightweight piece of code.

I'd like to find out what the rest of you think about the QL. If you have looked at the QL enough to have an opinion, just drop me a letter or postcard with:

Do you have one? (It's not necessary to own one to participate.) Are you pro or anti? (Or leaning?) What do you like about it? What don't you like about it? Have you had trouble with it? How? What are you using it for? How many hours per week are you using it? What add-ons do you have?

Send it to:

QL Survey
Micro Cornucopia
P.O. Box 223
Bend, OR 97709

I'll compile all the information and report back. If there's a lot of response (positive or negative), we'll take a closer look at this incongruous little system.

Where's August?

Before you look at this issue's cover and decide that we're not having August in Bend, please remember that all year long we anticipate August. It's a grand month.

This is really the August-September issue of Micro C. It was edited in May, produced in June, printed in July, and mailed the first of August, just like every other August-September issue. However, it takes two to four weeks for the newsstands to get their copies and put them out, so about a month ago I decided to start moving the deadlines up a week per issue until we'd gained four weeks.

That idea wasn't popular with: the writers, columnists, staff, advertisers, the Infinite Improbability Drive (IID), or Sandy. They got their heads together and the IID suggested we change the date rather than fighting a losing battle.

Now everyone's happy. You and I know this is the August-September issue, everyone else thinks it's the September-October issue. (What will the IID think of next?)

Culture Corner

As you noticed last issue, the Culture Corner is a forum where serious scientific principles are discussed in an unheated environment. In this issue's column, noted Physicist (musical soda jerk) and beer connoisseur Leeward P. Sailors gives us his observations and counter conclusions on the maintenance of parity. (Note: after hearing from Sailors, the Micro C staff coughed up a pair of men's black socks and sent them to him post haste. They should have arrived quickly because they were definitely ready for washing.)

If you, too, have been closely observing the parity of socks (or pantyhose) or the macroparticulate movements of dogs, plates, or other members of your family, you may want to make careful notes. (In the interests of science, if not family peace.)

Board Reviews

Every once in a while I get information that's too valuable to keep to myself, even though I can't credit someone with it (at least not in print).

A large U.S. software firm just finished testing a group of 8 and 10 MHz AT clones. Over the past few months it collected documentation on about 90 different systems. From the information about BIOS type, speed, and so on, it chose a dozen top systems to test on-site.

The tests were for hardware compatibility (video cards, I/O cards, hard drive cards, Bernouli drives, standard AT keyboards...) at full speed operation. (For instance, one board might run fine at 10 MHz with a Herc card and hard drive but only run dependably at 6 MHz after an EGA card was added.)

The Following Passed:

- The Zentel 10 meg AT card came out #1 (the only one that passed every test, including running Autocad with a PGA card at 10 MHz).
- The Everex 1800 10 meg board, the cheapest of the winners. It has an Award BIOS.
- The Samsung system, had a switch for 6 or 8 MHz and 0 or 1 waitstates.
- The Multi-tech system, another complete computer. It has an Award BIOS.
- The WiseTech system. It has a Phoenix BIOS.
- Compaq.

The company prefers boards with either the Award BIOS or the Phoenix BIOS, although it also has systems with Compaq and TI BIOSs.

PC Tech's X32

A couple of issues ago, I mentioned that PC Tech was working on an 80386 board. I also mentioned that when the system showed up, I'd have to change my definition of instantaneous.

I said those things to keep Dean and Earl (at PC Tech) from getting complacent. The way I figure it, technology wouldn't go anywhere if us editorial types didn't properly challenge them laggardly engineering types. You'd be surprised how quickly a little anticipatory journalism can cure a case of oversat duffs. (And you thought vaporware was the manufacturer's fault.)

Anyway, Dean and Earl got busy and put together:

- A 16 MHz 80386 (it'll run faster when Intel gets its parts unglued).
- Six PC/XT/AT slots, two with 32-bit extension cords.
- 80387, 80C287, or 80287 arithmetic option.
- 768K of static, no wait-state RAM.
- Battery-backed-up real time clock.
- Memory expansion to 32 meg (yep, a whole potato field of chips).
- Eight-channel DMA.

- Fifteen-source interrupt controller.
- All on a board that mounts into an XT or AT enclosure (for motherboard upgrades).
- Plus, it is made in the U.S. (we've got to keep these guys here).

I'd say my new definition of instantaneous will be over 20 times faster than a standard XT and about 5 times faster than a standard AT.

There's no question I've created a winner.

PC Tech
904 N 6th St.
Lake City, Minnesota 55041
(612) 345-4555

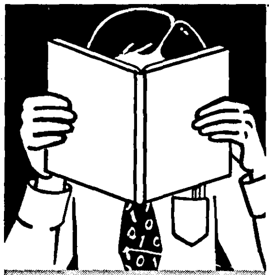
Speaking Of Hardware

John Jones is starting his $6 scanner this issue. He says it's not a toy but a real, honest-to-gosh, output to PC Paintbrush-type machine. All you have to have is a printer which can handle Epson or IBM graphics (or you write your own printer driver). The printer writes blanks while stepping the head across the document.

Meanwhile, Bruce Eckel's Real World series has generated a very avid following. Who knows, maybe hardware isn't dead after all.

And that's all from greater Bend.

David Thompson
Editor & Publisher



THE LAST PAGE

"Appearing Intelligent": From Eliza To Lucid Lucy

By Gary Entsminger
1912 Haussler Dr.
Davis, CA 95616

No one ever tells me anything. First, Gary gets married. Then he starts working on Lucy. Now he wants the rest of us to make her smart.

With all the chatter about intelligent computers (and programs) these days, it's no wonder I'm confused (and not feeling very intelligent).

I'm only human. Which is what a lot of people mistook the Eliza program for after Joseph Weizenbaum (of MIT) created it in 1964.

Eliza, named for Eliza Doolittle of "Pygmalian" (or "My Fair Lady"), was designed to act like a non-directive psychiatrist and to demonstrate Weizenbaum's belief that a computer could appear intelligent, without being intelligent at all.

But then, so do a lot of humans.

Eliza

The original Eliza and most of her imitators look intelligent by keying on the user's input and responding in some prearranged manner. A typical dialogue with Eliza might go -

"How are you?"
"Fine."
"You're fine?"
"Yes, but my girlfriend isn't."
"Your girlfriend isn't?"
"No, she isn't."
"Tell me more."

with Eliza recycling the user's input to generate her own response. The key to her "intelligence" lay in Weizenbaum's (a programmer) intuition for anticipating a user's conversation (input). (Notice that we can suggest Eliza's humanness, or intelligence, right from the bat, by using a personal pronoun, "her".)

I'm not particularly interested in psychiatry (or pseudo-intelligent dialogues with computers), but I am intrigued by Eliza's simplicity and the obvious advantages of having a program respond "intelligently" to input.

For example, suppose we want to let a user enter several different words to accomplish a task. In other words, we want to translate an input string (the user's command) into an action, which could be anything from a simple response -

(1) User enters, "Hello, there."
(2) Program responds, "Hello, yourself."

to the execution of some instructions or to a dialogue and some instructions -

(1) User enters, "Get me a file."
(2) Program executes "dir" (to show choices), and then responds, "Select a file, please."
(3) User enters, "test.txt".
(4) Program loads the file into memory.

We can easily write a program to accomplish this task in PROLOG, and make it "intelligent" by allowing it to be modified (at least to a certain extent) at runtime.

Expert System

To do it, we need to separate knowledge and inference engine, or create an "expert" (or "knowledge") system.

The inference engine will -

- Read the keyboard buffer (get user input),
- Compare the string it sees there with the input strings in its knowledge base, and
- If the input matches something in the knowledge base, find an appropriate response and output it.

Let's anticipate a large knowledge base by setting up an indexing system. And let's allow responses to "don't care" input, by using wildcards (_). A typical record of input information might look like this -

```
input(["hello"],1)
```

and a typical record of a response (output) might look like this -

```
output(1, "hello, yourself")
```

Figure 1 — Lucid Lucy

```
DOMAINS
        integerlist = integer*
        stringlist  = string*
DATABASE
        input(stringlist,integer)
        output(integer,string)
PREDICATES
        start
        main
        check_table(stringlist,stringlist)
        compare(string,string)
        process(string)
        string_to_list(string, stringlist)
CLAUSES

start:-                             /* Set up continuation */
        main.                       /* By using 2 predicates, */
start:-                             /* start and main, & fail */
        start.                      /* we can clear the stack, */
                                    /* and conserve memory. */
main:-
        readln(Sent),               /* Read input string. */
        string_to_list(Sent,S2),    /* Convert string to list. */
        input(L,ActNo),             /* Get an input record. */
        check_table(S2,L),          /* Compare input list */
                                    /* with input record. */
        output(ActNo,Action),       /* If match, get output. */
        process(Action),            /* Act. */
        !,fail.                     /* Continue by forcing
                                    /* backtracking with fail. */
main.

check_table([H|T],[H2|T2]):-        /* Recursively compare heads. */
        compare(H,H2),
        check_table(T,T2).
check_table([],[]).

compare(H,H2):-                     /* Do the actual comparison */
        H = H2.                     /* here. */
compare(H,H2):-                     /* If the record allows */
        H2 = "_".                   /* wildcards, succeed. */

process("greeting"):-               /* Execute an instruction. */
        write("Hello there."),      /* Expand Lucy by adding */
        nl.                         /* predicates here. */
process("greeting2"):-
        write("Hello, how are you?."),
        nl.
process("exit"):-
        write("Goodbye."),
        exit.
process(_).

string_to_list(S,[H|T]):-           /* Convert an input string */
        fronttoken(S,H,S1),!,       /* to a string list. */
        string_to_list(S1,T).
string_to_list(_,[]).

GOAL
        consult("input.dba"),       /* Load knowledge bases */
        consult("output.dba"),
        start.                      /* and begin. */
```

Figure 2 — Contents of input.dba and output.dba (and comments).

```
input(["hello"],1)                  /* If input is "hello", */
input(["hello","_"],2)              /* match with output 1. */
input(["quit"],3)                   /* If input is "hello + */
input(["_","quit"],3)               /* anything else, do 2. */
input(["_","quit"],3)               /* If input is quit, quit + */
                                    /* anything, or anything + */
output(1,"greeting")                /* quit, do 3. */
output(2,"greeting2")
output(3,"exit")
```

A don't-care record looks like this -

input(["hello","_"],1)

If we want a response to be the execution of a procedure, we write a record -

output(2,"greeting")

where "greeting" is a procedure ("predicate" in PROLOG) -

process("greeting"):-
write("hello there.").

Figure 1 is the complete inference engine (Lucid Lucy). Figure 2 is a little knowledge base (Lucy's intelligence).

Intelligence Testing

To extend Lucy's vocabulary, just add facts to her knowledge base. To extend her ability to execute, add predicates to her inference engine (and matching facts to her knowledge base).

To see how intuitive you are, try making Lucy more lucid, and enter your version in the Micro C Logical Contest before the November 1, 1987, deadline. There's still time to appear intelligent.

■ ■ ■

(continued from page 4)

Then Dave went home to Canada, George went home to Australia, and Definicon knocked out some sample boards using Dave's two-layer artwork. (They'd licensed Definicon to produce boards for commercial use.) Dan was supposed to get the same artwork so he could start having boards built for the kits.

Six months later Dan still hadn't received the artwork, Definicon had found that most of the two-layer boards didn't work, and anxious folks were getting restless. As time dragged on, midterms and finals came and went, Dan became less and less reachable.

Finally, this Spring, Dan called me. He was excited. Dave Rand had arranged to have Cybertools produce his new four-layer version and kits would be available in a couple of weeks. I called Cybertools and they told me yes, indeed, it was all true.

Two weeks later I heard that Cybertools had folded. I called Efron:

"You're kidding!" he responded. "I just talked to Chen. He didn't say they were bankrupt. I'll get back to you."

That was the last I heard from Dan.

Now Definicon is handling both the commercial sales and the kits. They loaned me one of their new boards. It didn't work on a Challenger 80186 clone, but it worked fine on PC-Tech's 80186-based X-16. Installation is fairly straightforward but you should have 15 meg of free space on your hard disk just to install and run UNIX. I'd recommend you use a 30 meg drive, minimum.

REC and Convert

We are certainly pleased by your coverage of Convert. (See Micro C issue #33.) One of the impediments to the dissemination of REC and Convert is the dearth of simple, expository material. Even our students frequently voice this.

There are a number of reasons why we haven't worked up more introductory material, although the fundamental one may well be good old laziness. English is difficult for the students while writing in Spanish is difficult for me.

Be that as it may, it hasn't been easy to find an attractive application for REC and Convert. One of our more successful applications has dealt with symbol manipulation in quantum mechanics, but it is far too esoteric for general consumption. We killed off another promising application (transforming Intel 8080 source code into 8086 code) by rewriting the program in machine language so as to gain an important increase in efficiency.

I'll continue to keep you informed of our projects here, and thanks again for a nice article. Let's hope it will serve to stir up a little interest in these languages.

Harold V. McIntosh
Universidad Autonoma de Puebla
Apartado Postal 461
(72000) Puebla, Puebla, MEXICO

Help!

They're gone! Two years ago I purchased a Columbia Data Products portable computer. It came with 256K, two drives, and lots of software. I was assured by the sales person that I could increase the memory by simply adding chips.

Now I'm ready and the computer store is gone. In fact, Columbia is gone too! I'd like to hear from any individuals or businesses that have information on the CDP.

Steve Taylor
2700 Cantu Ln. NW
Bremerton, WA 98312

ADVERTISERS INDEX
Issue 37

ORDERFORM

Postage-paid self-mailer: tear out, fold and staple edge as indicated.

☐ YES, I WANT TO SUBSCRIBE!

☐ NEW ☐ RENEWAL

	U.S.	CAN./MEX.	FOREIGN
1 yr. 6 issues	☐ $18	☐ $26	☐ $36
2 yrs. 12 issues	☐ $34	☐ $50	☐ $68
3 yrs. 18 issues	☐ $48	☐ $72	☐ $99

☐ **DISKS** ☐ MS DOS 5¼" ☐ MS DOS 3½" ☐ Other
Specify Disk # and size

DISK TOTAL

Save **24% Off** the newstand price

☐ **OTHER PRODUCTS**
Back Issues, T-shirts... specify size

OTHER TOTALS

GRAND TOTAL

TEAR HERE

☐ VISA ☐ MASTERCARD ☐ CHECK ENCLOSED Staple both ends of self-mailer if check is enclosed.

☐☐☐☐ – ☐☐☐☐ – ☐☐☐☐ – ☐☐☐☐   __/__ Expires

Are you a current Micro C subscriber? ☐ Yes ☐ No

NAME ....

COMPANY ....

ADDRESS ....

CITY .... STATE .... ZIP ....

STAPLE TO CLOSE

FOLD HERE

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 19 BEND, OR

POSTAGE WILL BE PAID BY ADDRESSEE

THE MICRO TECHNICAL JOURNAL

MICRO CORNUCOPIA

P.O. Box 223
Bend, OR 97709-0223

TEAR HERE

FOLD HERE