No. 49 September-October 1989 $3.95

THE MICRO TECHNICAL JOURNAL

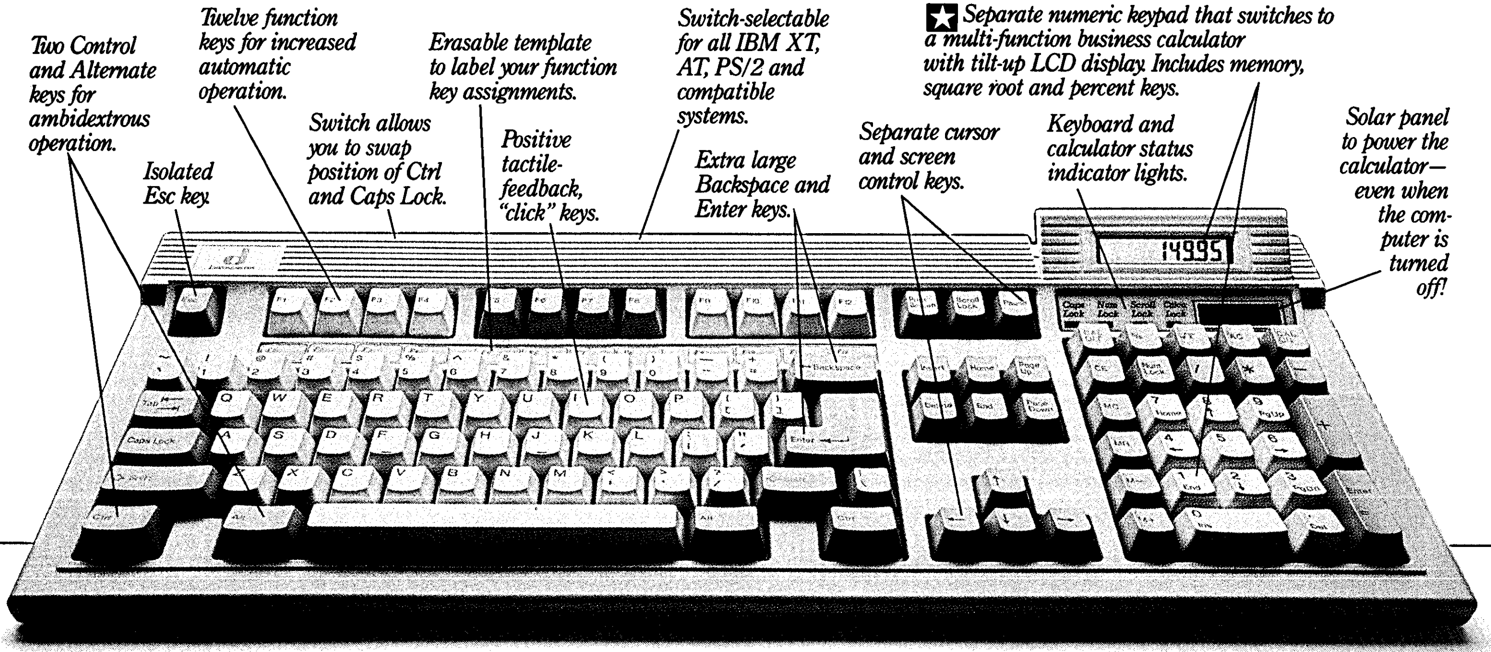
MICRO CORNUCOPIA

I/O, I/O, It's Off To Work We Go

Want to know where the hackers are? They're busy automating industrial processes. It's easy. Very easy.

LIMITED OFFER

Pay us the street price for Quattro and we'll give you the keyboard to drive it with.



Two Control and Alternate keys for ambidextrous operation.

Isolated Esc key.

Twelve function keys for increased automatic operation.

Switch allows you to swap position of Ctrl and Caps Lock.

Erasable template to label your function key assignments.

Positive tactile-feedback, "click" keys.

Switch-selectable for all IBM XT, AT, PS/2 and compatible systems.

Extra large Backspace and Enter keys.

Separate cursor and screen control keys.

★ *Separate numeric keypad that switches to a multi-function business calculator with tilt-up LCD display. Includes memory, square root and percent keys.*

Keyboard and calculator status indicator lights.

Solar panel to power the calculator—even when the computer is turned off!

$149.95 buys you *both* the hot-selling spreadsheet *and* the TurboCalc-111 Keyboard/Calculator.

For just $149.95—less than Quattro's street price, and a lot less than its $247.50 suggested retail price, you can now get *both* Borland's best-seller *and* the keyboard you need to drive it at top speed.

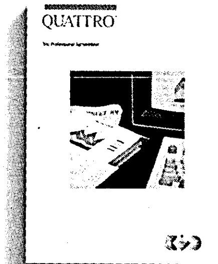
Namely, the TurboCalc-111™ Keyboard/Calculator from Datadesk.

Boost your overall performance.

With its built-in, presentation-quality graphics, intelligent recalcs, unlimited macros, easy installation and compatibility with leading spreadsheet and database software, Quattro is made to order for your business.

And TurboCalc-111 is made to order for Quattro. Or for any other software you like to drive.

Because, as you can see, it's loaded with features designed to turbocharge your spreadsheet and typing performance.



"More than 1-2-3® at less than half the cost." That's what PC Magazine says about Quattro,® the hot-selling spreadsheet from Borland. Imagine what they'll say about this extraordinary deal!

Like our famous tactile, positive-response keys that give you a much better feel for the road. So you can type faster with fewer mistakes than ever before.

And the new, enhanced IBM®101-key layout with some logical improvements—including separate numeric and cursor keys that let you cruise through spreadsheet data entry without ever having to shift Num Lock.

Get better mileage from your desktop.

In case you haven't noticed already, the keypad doubles as a full-function business calculator complete with its own pop-up LCD display. Which saves space on your desktop and lets you perform any calculation with a single keystroke—no matter what software you're driving.

What's more, the keypad packs a solar panel, so you can start up the calculator even when your computer is idle.

We wouldn't steer you wrong.

Frankly, getting into a Datadesk keyboard would be an inspired idea at this price even if you didn't get Quattro in the bargain.

After all, as *InfoWorld* says, "if you haven't looked at Datadesk's keyboards, you ought to."

According to the Washington Post, "for ingenuity of design and sheer dollar value, Datadesk can't be beat."

And when it comes to your peace of mind, nothing beats our two-year warranty.

What's more, if Quattro and TurboCalc-111 don't blow the doors off the vehicles you're currently driving, just send them back within 30 days and we'll cheerfully refund your $149.95.

No questions asked.

How, you ask, can you take advantage of this remarkable offer? Just fill out the coupon and send it in.

Better yet, call us toll-free.

And tell us to step on it.

DATADESK

All individual trademarks and copyrights are acknowledged.

RISK-FREE

$149.95 **Bundle includes Datadesk's TurboCalc-111 Keyboard/Calculator for IBM® and compatibles *and* Borland's Quattro spreadsheet.** Add $10 shipping and handling per unit (Continental U.S. only). CA residents please add $9.75 sales tax per unit.

# of Units:________ Amount Enclosed:________

Computer Type*:________ Disk Size: ☐ 3½" ☐ 5¼"

*If PS/2, include additional $5 for cable adapter.

Payment: ☐ VISA ☐ MC ☐ AMEX ☐ CHECK MC

Card No:________ Exp. Date:________

Name

Company Name

Daytime Telephone *Address*

City *State* *Zip*

Mail to: Datadesk, 7651 Haskell Ave., Van Nuys, CA 91406. FAX: (818) 780-7307

Or Call: (800) 826-5398. In CA, Call: (800) 592-9602

30 DAY MONEY-BACK GUARANTEE

Reader Service Number 8

QuickMod V2.0 Compiler...
Just think what the two of you can do.

You can make software engineering faster and easier than it's ever been before. The high performance QuickMod by Stony Brook delivers a compilation speed that exceeds 15000 lines per minute on AT machines. And it's a true two-pass, Modula-2 compiler with an intelligent environment that determines dependencies automatically and builds your program with a single keystroke. You do little more than enter the program text—then let Stony Brook handle the work load.

QuickMod is more than a supercharged compiler—it is a fully supported package backed up by a text editor, a linker, a library manager, a symbolic debugger and a runtime library that offers more functionality than anything else on the market. All for $95. Versions available for DOS and OS/2.

© 1989 Gogesch Micro Systems, Inc.

And when you need more horsepower, you can move into the Professional Modula-2, Stony Brook's fully optimized compiler. You get the same high productivity environment with code generation and flexibility you can't find in any other product in the industry.

Stony Brook—we design our products specifically to improve developer performance. And we know software engineering. Put us to work for you.

Call us direct and we'll mail product information to you within 24 hours.

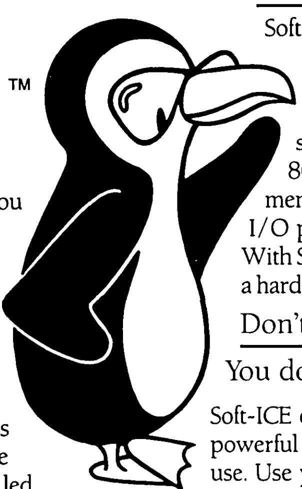
800/624-7487
805/496-5837 **California and International**
805/496-7429 **Fax**

Reader Service Number 152

Stony Brook
SOFTWARE

Your Partner
in Software Development

187 East Wilbur Road, Suite 9,
Thousand Oaks, CA 91360

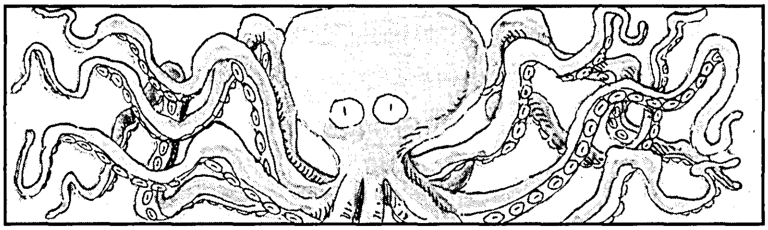

TM

THE MICRO TECHNICAL JOURNAL

MICRO CORNUCOPIA

SEPTEMBER/OCTOBER 1989 - ISSUE #49

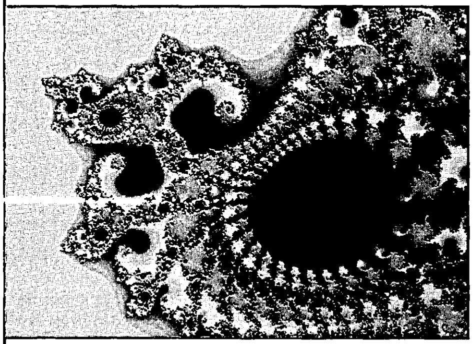
FEATURES



COLUMNS

FUTURE TENSE

COVER



Cover PostScriptal by Larry Fogg.

THE MICRO TECHNICAL JOURNAL

MICRO CORNUCOPIA

Editor and Publisher
David J. Thompson

Associate Editors
Gary Entsminger
Larry Fogg
Cary Gatton

Contributing Writers
Anthony Barcellos
Bruce Eckel
Michael S. Hunt
Scott Ladd
Laine Stump

Advertising & Distribution
Jeff Hopper

Accounting
Sandy Thompson

Order Department
Tammy Westfall

Graphic Design
Carol Steffy

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 (surface)	$36.00
1 yr. Foreign (airmail)	$50.00

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 to:

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

CUSTOMER SERVICE:
For orders and subscription problems call
503-382-8048, 9 am to 5 pm, Pacific time, M-F.

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

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

ISSN 0747-587X

Audit Bureau of Circulations

By David J. Thompson

No Lack Of Drive

Hard Drives

I've been having trouble with my little 20 meg 3 1/2" Miniscribe. Just use it for text mostly (not the heavy stuff that uses all eight bits), but I began hearing this siren sound as the spindle motor slowed down and then came back to speed. I called MicroSphere.

"Make sure the DC power connector is firmly attached. Take needle nosed pliers and push in the little metal connectors."

That seemed reasonable, but the symptoms got worse. Then, not content with slowing down, the motor began speeding up (trying to improve transfer rate I suppose). Certainly not a power problem.

Ah well, I backed up everything (using Fastback) and trucked the little hummer back to MicroSphere (aren't warrantees nice?). Within half an hour I was back at Micro C with another 20 meg 3 1/2" 8425. (I really like the little drives because they're very quiet.)

Fired up the system, booted off floppy, did a low level format (g=c800:5), FDISK.... Whoa, wait a minute. I'm the guy who tells people to let their drives run at least half an hour before formatting. This would be a great excuse to sit back and....

I started another low level format, made a cup of tea, and glanced through an article in the *Wall Street Journal* about Compaq's new 33 MHz 386 system. (Something about beating IBM, whoever they are, in the performance race.)

Boy, formats go fast when you're reading something light. After the second format finished, I started a third. That's three low-level formats in a row. I followed up with FDISK and then ran DOS FORMAT—twice. If those little platters aren't impressed, they'll never be.

Finally I restored the files using FASTBACK and the drive was back. So far this one has run flawlessly. I must say, though, humming along with the other one was much more interesting.

Nondestructive Reformatting, Again

It seems like every time I say something about hard drives, I get mail. This time I got lots of mail and Gibson Research pulled its ad for SpinRite.

It's not Gibson's fault that some XT controllers have problems doing a single-track format. Gibson has to talk through the controller just like everyone else.

Continued on page 71

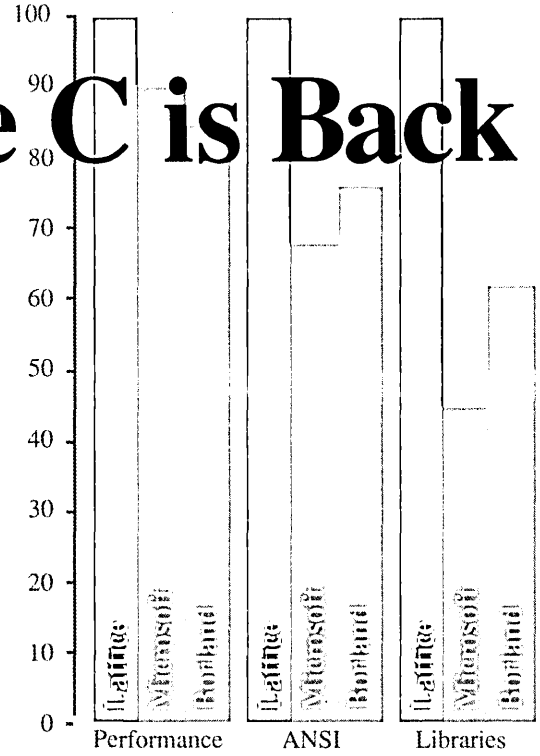
Lattice C is Back on Top!

The Leader in Performance
Lattice C 6.0 for DOS and OS/2 is the fastest compiler overall on the eight *PC Magazine* benchmarks* in both large and small model. Lattice C is over 10% faster than Microsoft 5.1 and over 15% faster than Borland 2.0. These results are due to our new optimizer and the many performance improvements in the library. You can make your applications run even faster by using our new register variable support and built-in functions. *Lattice 6.0 is the performance leader.*

The Leader in Compatibility
Lattice C is fully compliant with the ANSI standard. We pass not only the *Computer Language* ANSI Test Suite but 100% of the Plum Hall Test Suite Version 1.09. *Lattice is the compatibility leader.*

The Leader in Debugging
Lattice C includes CodePRobe™, a new full-screen symbolic debugger for DOS and OS/2. CodePRobe has the familiar look of the CodeView, and can be used with a mouse. It also enables you to easily debug family mode programs, Presentation Manager applications, and OS/2 multithread applications. And, for those humongous programs and system killer bugs, CodePRobe can be used remotely from a different PC. *Lattice is the leader in debugging.*

The Leader in Innovation
Lattice has long supported popular language extensions such as *near, far,* and *pascal*. In our previous version, we introduced *align, chip, volatile, interrupt, nopad,* and *pad* to handle important memory management problems in DOS, OS/2, and embedded systems. Version 6 continues this evolution with *critical* and *private*. *Critical* defines a function that must run as a critical section, and private defines data that must be replicated for each execution thread. Future versions will continue to improve the language for use in cutting-edge environments such as OS/2 and AmigaDOS. *Lattice is the innovation leader.*



And the benchmarks show it.

The Leader in OS/2
Lattice C includes bindings and header files so that you do not need to buy an expensive OS/2 development kit. The library also supports multithread programming with no built-in limits to the number of threads. The Lattice linker, librarian, editor, debugger and utilities run in DOS, OS/2 and family mode. *Lattice is the OS/2 leader.*

The Leader in Productivity
The Lattice C Development System now includes the utilities that our customers–professional C programmers–use most often. Of course, the system is based upon our C compiler, editor, linker, librarian, and debugger. In addition, we now include our powerful project maintenance tool *lmk*, which is an advanced version of the UNIX *make* utility. *Build, diff, extract, file, splat, touch,* and *wc* enable you to easily maintain your source files, and *cxref* produces comprehensive cross-reference reports. *Lbind* produces family-mode executables that can run under both DOS and OS/2. *Lattice is the productivity leader.*

The Leader in Libraries
Our library has often been praised for its comprehensive and sophisticated features. Now we include *graphics, communications,* and *curses* libraries. The *communications* library supports XMODEM, YMODEM, KERMIT and ASCII protocols. The *graphics* library provides a graphic windowing model. *Curses* provides a UNIX compatible screen manager. *Lattice is the library leader.*

The Leader in Convenience
Lattice C uses an automated installation procedure that has been praised by reviewers. The compiler automatically configures itself so that you do not need to specify complicated options. Yet, a full set of options is available to the professional who needs to customize to a particular environment. *Lattice is the convenience leader.*

The Leader in Documentation
Lattice C includes over 1500 pages of high quality documentation in ring binders. It contains all the information that professional programmers need in a thoroughly indexed format. *Lattice is the documentation leader.*

The Leader in Support
Best of all, Lattice support comes free with Lattice C. Lattice's bulletin board and telephone support are the best in the business. *Lattice is the support leader.*

30 Day Money Back Guarantee
Lattice C 6.0 is available for $250. To order, send check or money order to: Lattice, Inc., 2500 S. Highland Avenue, Suite 300, Lombard, IL 60148. Or order by credit card at (800) 444-4309. FAX # (312) 916-1190, TELEX 532253.

Lattice

Professional Programming Tools Since 1981
2500 S. Highland Avenue, Lombard, IL 60148

---

*Send to Lattice for a free complete report on the benchmark analysis.

Reader Service Number 153

Letters

Large Video Characters

Regarding large characters for those with vision impairments: a big monitor would be ideal, of course. On my Zenith lap-top, and probably most CGA-compatible machines, "MODE BW40" or "MODE CO40" makes the characters double width. Not double height, but considerably bigger.

This works for all DOS commands (more or less) and probably some application programs, older ones more likely. A modern text editor which seems to work acceptably in this mode (scrolls the line horizontally and so forth) is my copy of the estimable QEdit. I think it's still available as shareware.

J. G. Owen
21 Glenview Ave.
Fort Salonga, NY 11768

Talking Amigas

Issue #48 was very interesting (as usual). I have a comment on Debee Norling's article, "Selecting a Talking Computer for a Blind Friend."

Although the IBM PC family is *the* business computer family, I was surprised to see no (zero, nada) references to the Amiga computer. Every Amiga can talk, right out of the box—no special hardware or software required. At a command line prompt, just type—

```
say {string to speak}
```

The general purpose text-to-speech algorithm seems pretty good, at least to me. If it isn't, you can send it phonemes instead. This speech capability is also available as a system level device, just like a printer or a serial port. You control the volume, speed, voice type (male, female, robotic), pitch/frequency, and rate.

Some Amiga applications take advantage of this ability to talk. If these do not meet your needs, there are a number of programmable text editors and applications that might work.

The widespread adoption of the ARexx language, with its interprocess communications ports, also facilitates linking applications together in novel ways. At a programmer's level, it's relatively simple to insert a task into the chain of event handlers (e.g., to speak keystrokes, including ctrl/alt/shift) and to access the speech libraries from within a program. The Amiga's true multitasking and ability to play high quality sampled sounds without any extra hardware are bonuses.

The Amiga has capabilities for the sight-impaired as well. It has become the standard computer for "Desktop Video." Its standard NTSC (or PAL) output and low-cost GenLoc options mean it can interface to a wide variety of video equipment. Its standard analog RGB output will drive the newer large analog monitors (or TV/monitors).

The bit-mapped display and system fonts mean you can select larger fonts for many applications. There's even a little public domain "magnifying glass" tool that lets you enlarge any portion of the screen.

In conclusion, the Amiga is a very capable machine and certainly in the running as a blind or disabled person's "window to the world."

Kurt Wessels
917 N. 87th
Seattle, WA 98103

Letters *From* The Editor

Debee's articles in Issue #48 generated lots of interest. One thing we didn't tell you: the products mentioned are available through Debee and Christy at their company, Grassroots Computing. Obviously, they can support the products from the perspective of experienced users. Or you can contact the manufacturers directly.

DECtalk—Digital Equipment Corp.
Contact: Bill Metcalf
2525 Augustine Dr.
Santa Clara, CA 95054
(415) 651-5474

Echo PC—Street Electronics Corp.
Contact: Mark Pfeffer
6420 Via Real
Carpinteria, CA 93013
(805) 684-4593

Video Voice—Grassroots Computing
P.O. Box 460
Berkeley, CA 94701
(415) 644-1855

BBS Phone # Correction

In the list of alternative BBSes for downloading Micro C's issue listings, published last issue, we embedded an incorrect phone number for the Generation 5 BBS in Washington, D.C. The correct number is—

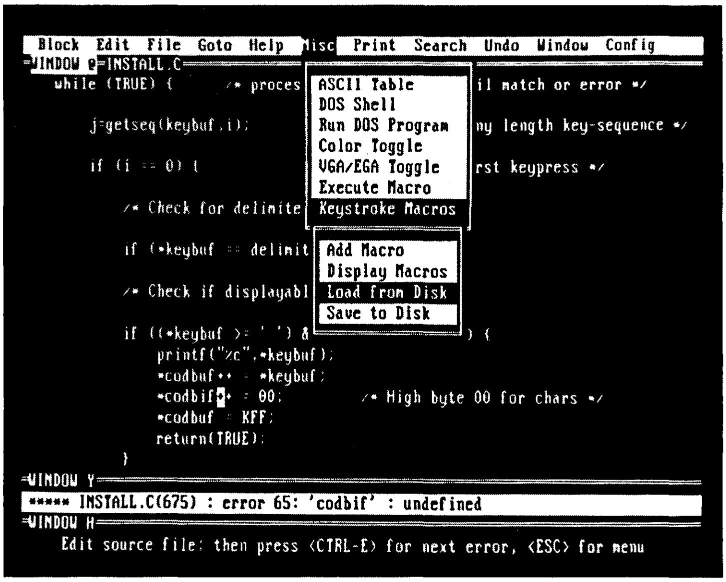
Generation 5 BBS
(301) 495-2932

This is a subscription board, but you don't have to be a subscriber to download Micro C code.

Good Magazine, But....

I've really appreciated your magazine for the past few years. You have

Continued on page 75



```
=WINDOW @=INSTALL.C
  while (TRUE) {       /* proces                    il match or error */

        j=getseq(keybuf,i);                         ny length key-sequence */

        if (i == 0) {                               rst keypress */

            /* Check for delimite

            if (*keybuf == delimit

            /* Check if displayabl

            if ((*keybuf >= ' ') &                  ) {
                printf("%c",*keybuf);
                *codbuf++ = *keybuf;
                *codbif + = 00;           /* High byte 00 for chars */
                *codbuf = KFF;
                return(TRUE);
            }
=WINDOW Y
***** INSTALL.C(675) : error 65: 'codbif' : undefined
=WINDOW H
   Edit source file: then press <CTRL-E> for next error, <ESC> for menu
```

Introducing . . .

The 1st Family of Low Cost, Powerful Text Editors

VEDIT Jr.	$ 29
VEDIT	$ 69
VEDIT PLUS	$185

FREE Evaluation Copy*
Call 1-800-45-VEDIT

Finally, you can choose the best editor for your needs without compromising performance or paying too much. And organizations that want the "same" editor for everyone can pick VEDIT® for most users and VEDIT PLUS for their power users.

The new family of VEDIT text editors are upwards compatible, easy to use and offer exceptional performance, flexibility and stunning speed. (3 to 30 times faster than the competition on large files where speed really counts.)

Call for your free evaluation copy today. See why VEDIT has been the #1 choice of programmers, writers and engineers since 1980.

VEDIT Jr.—Unmatched performance for only $29.
All VEDIT editors include a pull-down menu system with "hot keys," context sensitive on-line help, pop-up status and ASCII table, a configurable keyboard layout and flexible, unlimited keystroke macros. Edit files of any size and any line length. Perform block operations by character, line, file or column. Undo up to 1000 keystrokes— keystroke by keystroke, line by line, or deletion by deletion. Automatic indent, block indent and parentheses matching speed program development. Word wrap, paragraph formatting, justification, centering, adjustable margins and printing for word processing. Run DOS programs.

VEDIT—A best value at only $69.
Simultaneously edit up to 36 files and split the screen into windows. Search/replace with regular expressions. Includes the best compiler support available—menu driven, easy selection of compiler options, supports "Include" files and MAKE utilities.

VEDIT PLUS—Ultimate programmer's tool for only $185.
VEDIT PLUS adds the most powerful macro programming language of any editor. It eliminates repetitive editing tasks and permits creating your own editing functions. The macro language includes testing, branching, looping, user prompts, keyboard input, string and numeric variables and control over the size, position and color of windows. Source level macro debugging with breakpoints and tracing. Macros developed with VEDIT PLUS also run under VEDIT.

30 day money-back guarantee. Call for pricing of XENIX, OS/2 and FlexOS versions. Very attractive quantity pricing is available for schools, hardware and software vendors.

Compare Features and Speed

	VEDIT	BRIEF 2.10	Norton 1.3	QEdit 2.07
Pull-Down menus	Yes	No	No	Yes
Pop-Up ASCII table	Yes	No	No	No
Keystroke macros	100+	1	No	100+
Regular Expressions	Yes	Yes	No	No
"Cut and Paste" buffers	36	1	1	100
Text (book) markers	10	10	No	No
Undo keystroke by keystroke	Yes	Yes	No	No
Undo line by line	Yes	No	No	No
Normal/max Undo levels	500/1000	30/300	–	–
Variable tab positions	Yes	Yes	No	No
Configurable keyboard	Yes	Yes	No	Difficult
Integrated mouse support	Yes	No	Yes	No
FILE LIMITS				
Edit files larger memory	Yes	Yes	Difficult	No
Maximum line length	>8096	512	65,535	512
Maximum lines/file	8,388,607	65,535	>65,535	20,000
COMPILER SUPPORT			None	None
Menu driven	Yes	No	–	–
Select Compiler options	Menu	Difficult	–	–
Support "Include" files	Yes	No	–	–
BENCHMARKS 50K FILE				
Simple search	0.2 sec	1 sec	1 sec	0.3 sec
Save and continue	1 sec	2 sec	2 sec	1 sec
1000 replacements	**3 sec**	**19 sec**	17 sec	2.5 sec
BENCHMARKS 3 MEG FILE				
Simple search	1:40 min	1:36 min	Cannot	Cannot
Save and continue	1:05 min	3:23 min	Cannot	Cannot
60,000 replacements	**3:18 min**	**1:44 hour**	Cannot	Cannot
Block-column copy (40 x 200)	**2 sec**	**30 sec**	Cannot	2 sec
Insert 1 Meg file in middle of 1 Meg file	**1:11 min**	**15:13 min**	Cannot	Cannot
PRICE	**$69**	**$195**	**$75**	**$54.95**

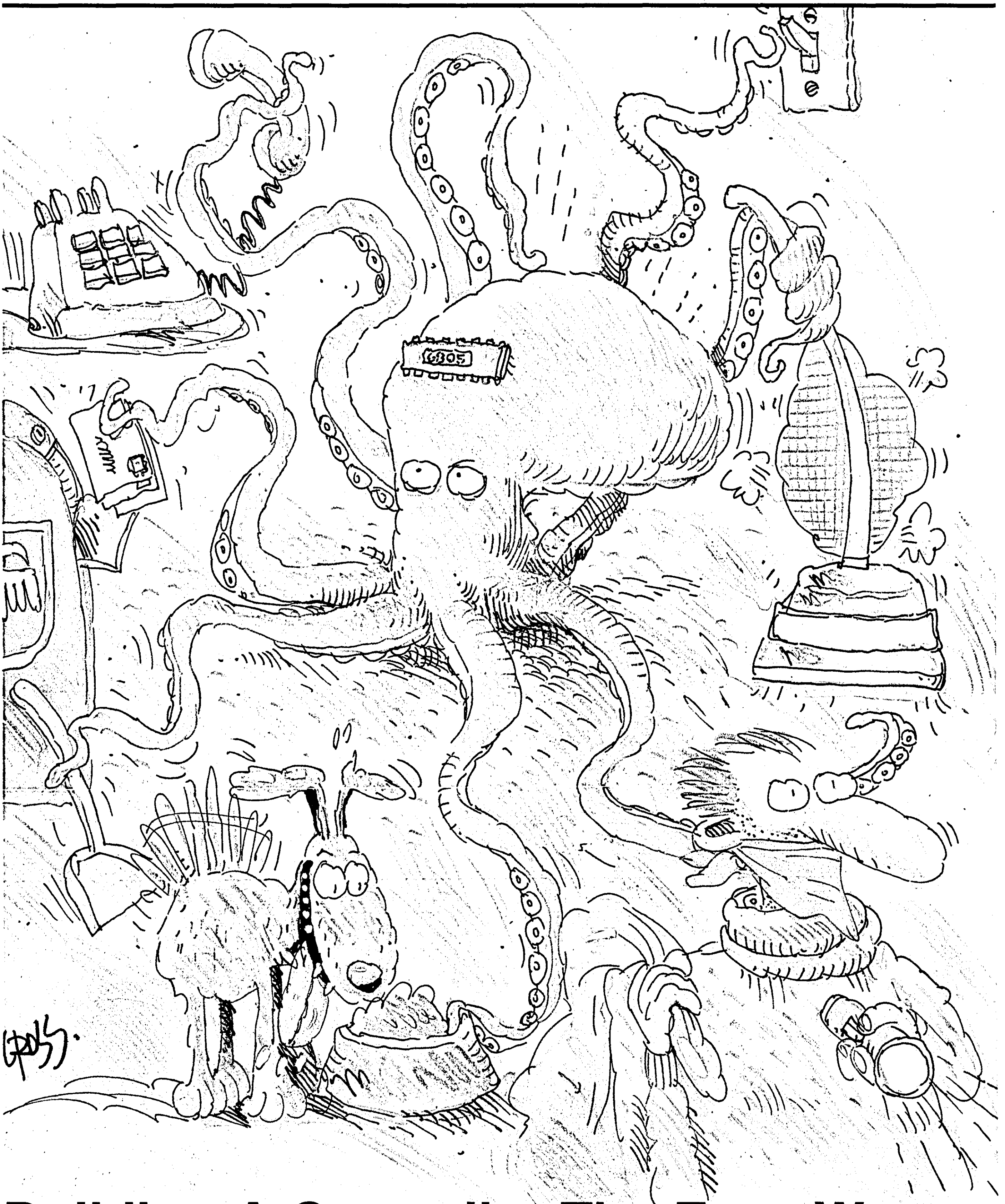
VEDIT and CompuView are registered trademarks of CompuView Products, Inc. BRIEF is a trademark of UnderWare, Inc. Norton Editor is a trademark of Peter Norton Computing Inc. QEdit is a trademark of SemWare.

*Supports IBM PC, XT, AT, PS/2 and clones with CGA, MGA, EGA, VGA, Wyse 700, Amdek 1280 and other displays. Also supports Concurrent DOS, DESQview, Microsoft Windows, PC-MOS/386 and most networks.
*Also available for MS-DOS (CRT terminals), TI Professional and others.
*Free evaluation disk is fully functional and can edit small files.

CompuView

1955 Pauline Blvd., Ann Arbor, MI 48103
(313) 996-1299, Fax (313) 996-1308

Reader Service Number 7



Building A Controller The Easy Way

Using The 68HC705

By Karl Lunt

4730 W. Menadota Dr.
Glendale, AZ 85308
(602) 869-6146 (work)
(602) 582-5863 (home)

Not satisfied with undocumented 68000 systems, Karl switches to a very well documented, very low power, low chip count controller. Sounds like Motorola has a real winner.

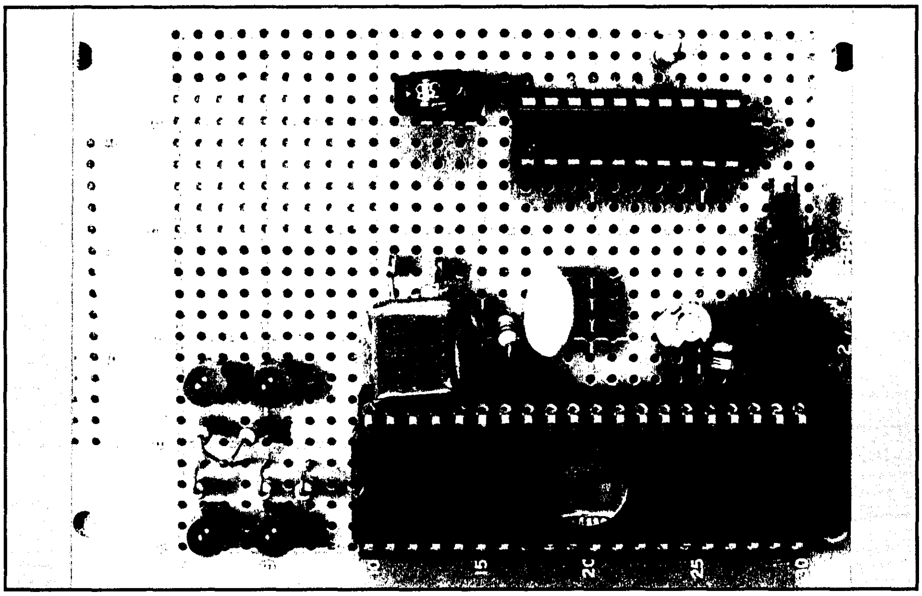
The Motorola 68HC705 micro-controller unit (MCU) contains enough EPROM, RAM, and I/O power to solve many controller problems. Use it with the Motorola 68HC05PGMR development board to build single-chip projects such as burglar alarms, robotics sensors, data acquisition units...just about anything you can dream up.

Take a look at the example design, complete with assembly language source code. The project provides 11 channels of A/D, maintains an internal time-of-day clock, controls three LEDs, consumes just 30 mA from a battery supply, and uses only two ICs!

The Chip's Hardware

The 'HC705 MCU is the latest addition to Motorola's line of 6805 micro-controllers. Being CMOS, it typically draws about 4.7 mA at 5.0 volts, making it ideal for low-power (e.g., remote) projects. The on-board 7.7K EPROM and 176 bytes of RAM provide room for surprisingly powerful programs. Since each of the 24 bidirectional I/O pins can drain up to 25 mA, the 'HC705 interfaces directly to LEDs, beepers, LCD displays, and other low-current devices.

You can run this chip with an external crystal of up to 4.2 MHz (V_{dd} = 5.0 volts); the internal bus runs at half the crystal frequency. I/O and option registers (special-purpose addresses) give you software control of the CPU's RAM/EPROM layout, I/O ports, counter/timer, and communication systems.



Proto Board for Example Project—Small, Right?

The on-board Serial Communication Interface (SCI) handles asynchronous serial I/O, though you must provide your own RS-232 level shifting (+12 V to -12 V). You control baud rate and port setup through I/O registers. A Serial Peripheral Interface (SPI) adds high-speed synchronous I/O, up to 1.05 Mbit/sec.

You can use the internal 16-bit counter/timer system as part of a real-time clock, frequency counter, or waveform generator. You can select from a wide variety of interrupts on the counter/timer.

For those really critical functions, the chip contains a Computer Operating Properly (COP) timer. When enabled, this feature will force a reset should your program fail to periodically refresh the COP register. Additionally, the chip can provide a reset if the system clock falls below 200 KHz.

The 'HC705 also offers two additional low-power modes (as if 4.7 mA isn't low enough!). WAIT suspends CPU operation but keeps the timer, SCI and SPI running. Any of these three systems can bring the CPU back to full RUN mode.

WAIT consumes about 1.7 mA; the STOP mode typically uses only 2.8 µA (no, that's not a typo). STOP shuts down all internal processing; only an external interrupt or reset can bring the CPU back to life.

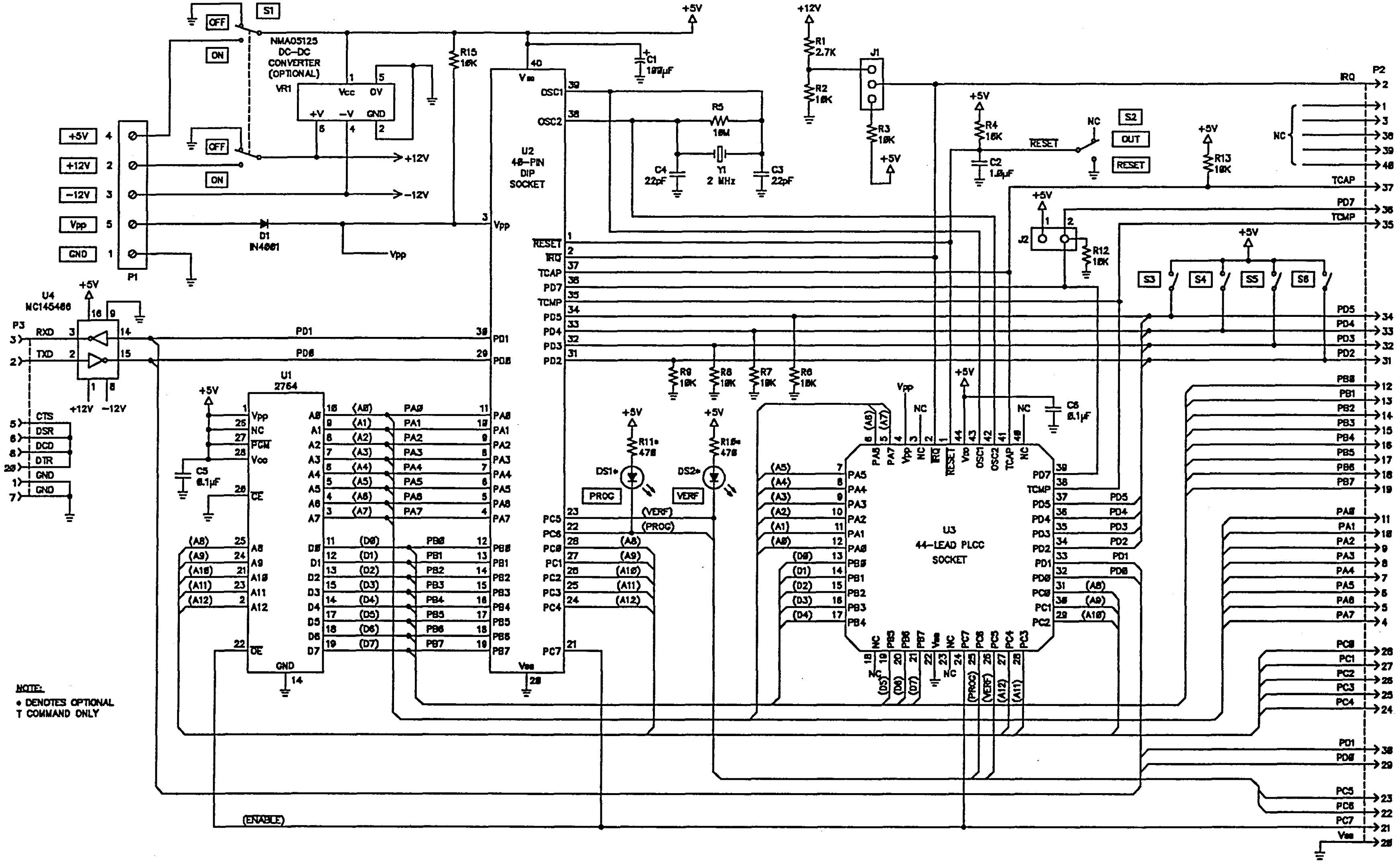
The on-chip EPROM erases with a standard shortwave UV lamp. If you burn your program into the MCU and find it doesn't work, just erase the MCU, change the software and try again.

When it comes time to ship your final product, Motorola also offers the MCU as a one-time programmable device (OTPROM). This plastic package does not have a UV erasing window—once you've programmed it, you've programmed it.

Inside The Programming Model

At first glance, the 'HC705 programming model seemed like a cruel joke. The

Figure 1—PGMR Board Schematic Diagram.



Reprinted from the Programmer Board User's Manual with permission from Motorola.

single accumulator and single index register each hold only eight bits. The fixed-position stack wraps after only 64 bytes. The entire address space is a mere 8192 bytes. Quite a change from my 68000.

The more I used the instruction set and studied the internal design, the more respect I gained for the chip's developers. This stripped-down model offers plenty of power for the jobs it will be asked to do. Ten addressing modes, with heavy emphasis on bit manipulation, give you lots of control over the many I/O registers. The flexible interrupt and timing systems move much of the clock-watching and baby-sitting duties into hardware, freeing up the CPU.

Using the SPI and SCI for real-world communications amplifies the chip's power considerably. These systems handle sophisticated interface jobs smoothly, with minimum CPU time.

Even the pint-sized stack won't get in your way. Since a subroutine call uses two bytes and an interrupt only costs five, the stack provides plenty of elbowroom. Keeping the stack size small frees up more of the valuable zero-page RAM (the stack runs from $FF down to $C0) for your programs. (Accessing zero-page RAM saves a cycle per instruction, since the CPU knows the operand's address lies in the lowest 256 bytes.)

The bottom line: this chip's streamlined architecture is fun to work with. Don't be fooled by the paucity of registers and tiny stack—you can handle serious control projects with the 'HC705.

The Development Board

The 'HC705 does not support external RAM or ROM, so programming can be a problem. Fortunately Motorola offers the M68HC05PGMR development board. Used with a PC-based interface program (PROG7—supplied by Motorola with the 'PGMR board), you can download software to the 'HC705, program the EPROM, and test your code. So the board constitutes a complete 'HC705 development system.

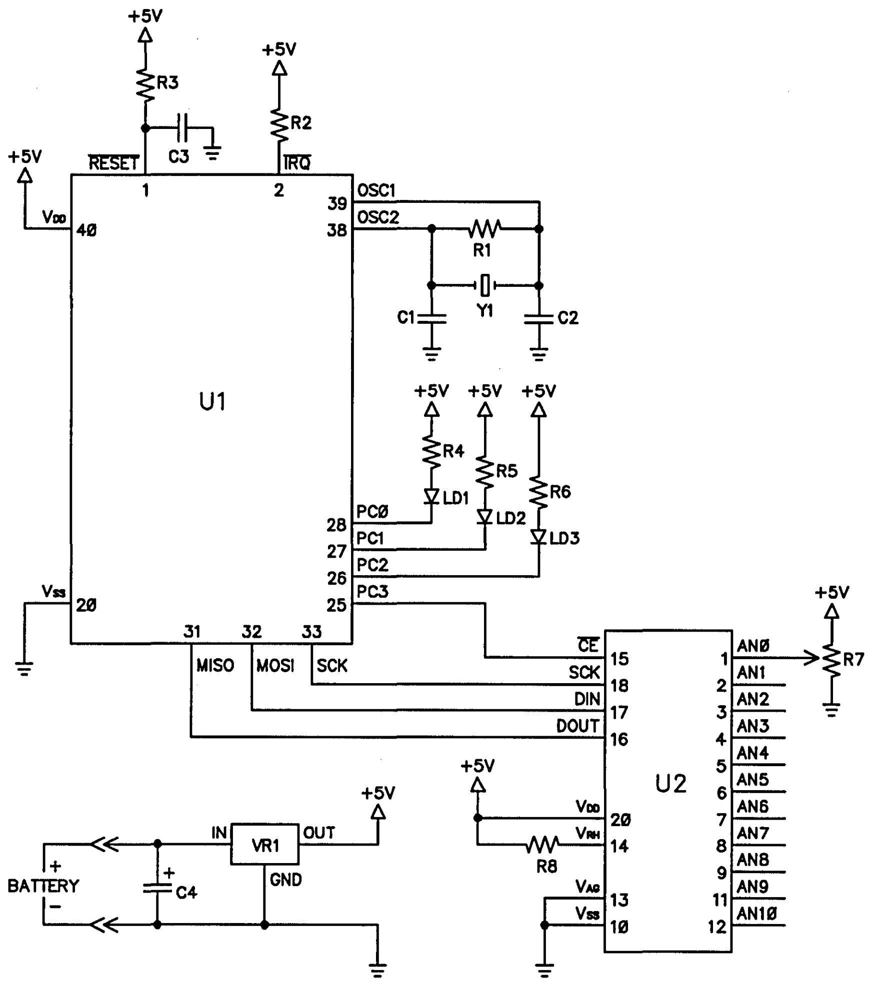
The 'PGMR board contains little more than RS-232 level shifters, a crystal oscillator, LEDs and pullups, and necessary connectors. (See Figure 1.) All timing and control for EPROM programming and serial I/O get done by PROM code in the 'HC705.

The 'PGMR board sets up easily. Connect a serial cable to your PC, plug an 'HC705 into the processor socket, and supply +5, +12 and -12 volts. Programming the processor's EPROM requires adding a switchable +15.5 volts (V_{PP}).

After burning your software into the MCU's EPROM, you can reconfigure the 'PGMR board as a test bed for debugging. All 40 pins of the MCU socket appear at an IDC connector on the card's edge. By connecting a ribbon cable to a 40-pin header, you can "plug" the development board's MCU into an external circuit. Since the 'PGMR's oscillator and RS-232 port remain connected, your prototype only needs to provide interface circuitry.

The 'PGMR can even serve as a low-volume production facility. You simply plug a 2764 EPROM containing your finished program into the on-board 28-pin socket. Next, reconfigure the 'PGMR board to perform an automatic copy on reset. Put an erased MCU into the socket, apply all voltages (including V_{PP}) and bring the 'PGMR board out of reset. Firmware in the erased MCU copies the 2764's contents into the MCU's EPROM.

Figure 2—Schematic for the example MCU project, reprinted from the Programmer Board User's Manual with permission from Motorola.



Wiring layout is not critical; a Radio Shack experimenter's board (276-168A) works well. Use high-quality sockets for both ICs. The battery voltage can be between 8 and 30 volts.

The Power Of The SPI

Motorola's high-speed SPI adds a new dimension to micro-controller design. By replacing the older 8-bit parallel interface with the SPI's three-wire synchronous bus (serial in, serial out, and clock), wiring complexity between the MCU and support chips drops significantly.

For example, the Motorola MC145041 serial A/D chip puts 11 channels of 8-bit A/D in a 20-pin package. It takes only five wires, including ground, to interface this chip to the 'HC705. The '041 boasts a maximum cycle time of 40 µsec.

The Motorola MC14499 LED display driver (again, with serial interface) controls a four-digit display. You provide the common-cathode seven-segment displays, eight resistors (seven segments plus decimal point), and four driver transistors. The '499 translates serial data from the MCU into the display pattern, and even handles digit multiplexing. You can also cascade additional LEDs.

Motorola also offers several SPI-controlled PLL frequency synthesizers, such as the MC145159-1. The MCU can vary the reference counters in this 20-pin device; add a loop filter and VCO, and you have a frequency synthesizer.

Wiring the 'HC705 SPI to a peripheral chip is simple; you connect the two serial data lines, the clock line, ground, and chip enable between the devices. The first three signals come from dedicated pins on the MCU. You normally use one of the MCU's outputs as a chip enable line.

To transfer data between the two devices, simply drop the chip enable line, load the proper data into the MCU's SPI data register, and wait for the transfer complete bit in the SPI's status register to go high. Return the chip enable line high to deselect the device, then read the data transmitted by the device from the MCU's SPI data register. Piece o' cake!

The *M68HC05 Microcontroller Applications Guide,* available from Motorola, gives a sample thermostat project using an 'HC705 MCU and MC145041 serial A/D. I used the same connections between these chips in my example design; for details, see Figure 2.

The Guide contains full source listings for the thermostat software; source code is also available from the Motorola FREEWARE BBS. The routine A2D (see Figure 3) contains all the 68HC705 code needed to interact with the MC145041, using the circuitry shown.

The Example Project

I wanted a simple demo project, using the 'HC705 MCU and the MC145041 serial A/D. So, I defined a small, one-channel voltage sensor.

The sensor samples channel 0 of the A/D once per second. The A/D returns an 8-bit value from $00 to $FF. Depending on the value returned, a specific two-LED pattern will be displayed: if the value lies from $00 to $3F, a green LED lights; if the value lies from $40 to $7F, the green LED blinks once per second; if the value lies from $80 to $BF, a red LED lights; if the value lies from $C0 to $FF, the red LED blinks once per second.

I wrote the assembly language code on my PC clone, using the SideKick notepad editor. The PC-based AS5 assembler, available from the Motorola FREEWARE BBS, took care of assembling the code. I then used the PROG7 interface program to move the code into the 'HC705 EPROM.

The project's software uses less than 256 bytes of EPROM, but offers considerable flexibility and power. It should serve as an excellent starting point for other micro-controller projects. (You'll find a lot of example 'HC705 programs on the FREEWARE BBS.)

The A2D routine shows how to control the A/D converter chip with the MCU. Upon entry, the accumulator (ACC) should hold the channel number in the low nybble.

An Example

Let's say you've just read A/D channel 1 and now you're asking for data from channel 2. When you ask for channel 2, A2D returns the voltage reading from channel 1. This is most important—a call to A2D does not return the requested channel's value, but the value of the previously requested channel.

DEST Facsimile Pac™

The Real thing, as reviewed in PC Magazine, April 11, 1989.

Features:

- ⋆Full CCITT Group III $^{(9600 \text{ baud})}$ capability
- ⋆True Background operation
- ⋆Telephone port
- ⋆Keeps daily activity log
- ⋆Stores speed dial numbers
- ⋆Automatic requeuing, Broadcast mode
- ⋆Compatible with IBM PC™ XT™ AT™ PS/2 Model 30™ and compatibles
- ⋆DEST scanner input
- ⋆3.5" and 5.25" Software diskettes
- ⋆Pop-up Menu for ease of use
- ⋆Comprehensive instruction manual

JUST RECEIVED,

SPECIAL PURCHASE!

SORRY, NO ILLUSTRATION AVAILABLE!

Now, for the cost of a clone, the real thing, a full featured fax card that supports a wide variety of printers (Epson-Okidata-HP LaserJet+ (150 & 300 dpi) and others). Incoming fax can be printed during reception, spooled, or saved to disk for selective printing after review on your graphics display.

This single slot card is supplied with the normally *optional* Hayes™ compatible 300/1200 baud plug-on modem to permit sharing the same phone line between the FAX and your normal modem tasks.

This is not some OEM packed unit, It is the real thing, *(current list price $995.00)*

Our *low price is* **$379.00** (optional 300/1200 baud modem provided at no additional cost!)

Quantities are limited. Order Now!

REQUIRES DOS 2.0 AND ABOVE, DEDICATED PRIVATE RJ-11 SINGLE PHONE LINE, 640K RAM, HARD DISK, GRAPHIC DISPLAY (CGA EGA,OR HERCULES™)

THIS IS ONLY ONE OF THE HUNDREDS OF EVERYDAY BARGAINS AVAILABLE FROM HSC ELECTRONIC SUPPLY..

ALSO AVAILABLE AT HSC ELECTRONIC SUPPLY OF SACRAMENTO (916) 338-2545 AND HSC ELECTRONIC SUPPLY OF SANTA ROSA (707) 792-2277

American Express
Card
VISA

HSC ELECTRONIC SUPPLY

3500 RYDER STREET • SANTA CLARA, CA 95051

Toll Free (Orders Only)	**(800) 4-HALTED**
California Residents	(408) 732-1573
HSC FAX Line	(408) 732-6428
HSC Electronic Resource BBS	(408) 732-2814
Customer Service	(408) 732-1854

VALID THRU 10/1/89

Shipping: UPS Surface unless otherwise specified by purchaser. All items subject to prior sale. Orders shipped by means other than UPS Surface will be at the prevailing carrier rate plus $5.00 surcharge. All orders under $20 subject to $2.00 fee, our minimum order is $10 + shipping. All orders shipped FOB Santa Clara, California. Orders to APO and FPO addresses are shipped via U.S. Mail.

© 1989 HSC Electronic Supply a Halted Specialties Company

Reader Service Number 11

A2D first clears the SPI data transfer flag by reading the SPI status register. It then pulls the A/D chip enable line low by clearing bit 3 of MCU port C. Next, the A/D moves the channel number to the high nybble as expected. The MCU transmits the channel number to the A/D by writing to the SPI data register.

The loop at SPIFLP simply waits for the data transfer flag to go true, indicating that the two chips have exchanged data. The MCU then pulls the chip enable line high, deselecting the A/D. Finally, the byte transferred from the A/D loads into the ACC and control returns to the calling program.

This explains why the A/D always lags the MCU by one conversion. Each SPI operation exchanges data between the MCU and the A/D. Following the exchange, the A/D reads and converts the voltage on the newly selected channel. The value it reads on that channel isn't returned to the MCU until the next SPI exchange.

Figure 3—A2D Read Selected Serial A/D Channel

```
*  Enter with A/D channel number in ACC.  Returns analog value
*  in ACC.  Assumes chip select line to A/D chip is driven by
*  port C, bit 3 (active low).
*
*  Note that the value returned by A2D corresponds to the most
*  recent previous channel number transmitted, not the current
*  channel number!  Therefore, unless you know for a fact that
*  the last channel number you sent was the one you want to read
*  now, you had better call this routine twice, using the same
*  channel number.

A2D
        TST     SPSR       CLEAR SPIF
        BCLR    3,PORTC    PULL CHIP SELECT LINE LOW
        ASLA               MOVE CHANNEL NUMBER TO HIGH NYBBLE
        ASLA
        ASLA
        ASLA
        STA     SPDR       SEND CHANNEL TO SPI

SPIFLP  BRCLR   7,SPSR,SPIFLP     WAIT UNTIL COMPLETE
        BSET    3,PORTC    RELEASE CHIP SELECT
        LDA     SPDR       GET VALUE RETURNED
        RTS
```

♦ ♦ ♦

Burning And Testing

Once you've written the software, it's time to download the program into the MCU's EPROM. I hooked up my 'PGMR board to the PC, started the PROG7 program and applied power to the 'PGMR unit. It only took a few minutes to burn the MCU's EPROM.

After programming the MCU, I installed it in my test board (see Figure 2). I hooked up a Radio Shack experimenter's board as a test bed, using a 5K potentiometer across V_{dd} and ground for my voltage input to the A/D chip.

Applying battery power caused the MCU to go through its reset sequence; it immediately began running my program. If I varied the setting on the potentiometer, the LEDs blinked in the appropriate pattern. I measured the following voltages, using an analog VOM: 1st threshold (green LED starts blinking) = 1.2 volts; 2nd threshold (red LED lights) = 2.4 volts; 3rd threshold (red LED starts blinking) = 3.7 volts; full scale voltage (analog reference voltage) = 5.0 volts.

I need to stress a point about CMOS circuit design. Motorola recommends that you tie all unused input pins of the MCU to V_{dd} through 10K resistors. Several such pins exist in the example design, but the schematic does not show any pullups.

While you can usually get away with this in the prototype stage of noncritical designs, it is not good practice. Your project is more vulnerable to static damage and may draw significantly more current.

Talk About Support

Motorola is really supporting the 6805 family. I purchased the 'PGMR development board through my local Motorola distributor for less than its normal $168.05 (a promotion by Motorola). Even at list, this development board is a bargain. Any shop building custom MCU projects should check out this system.

I have mentioned the FREEWARE BBS often; Motorola sponsors this BBS as a way to distribute technical information, software, support tools, and literature updates. You can access the BBS at (512) 891-3733. It uses the standard eight bits, no parity, one stop bit at 300, 1200, or 2400 baud.

The BBS carries a variety of well-written software modules for many different 68xx(x) machines. For example, you can find the source code for the Motorola BUFFALO monitor for the 68HC11; a full floating-point package for the 'HC11 and 6809; cross-assemblers for chips from the 6800 to the 68HC11; even a few fully-developed MCU designs, complete with source code. If you're interested in the Motorola chips, be sure to try this BBS.

Motorola also carries quite a library of application notes on the 6805 family. Some sample titles include:

A General Purpose Frequency Counter Using an M6805 HMOS/M146805 CMOS Family Microcomputer (AN885)—describes a five-chip frequency counter using four-digit LED display. Gives schematic and full assembler source code.

Telephone Dialing Techniques Using the MC6805 (AN940)—shows how to use an MCU as a telephone dialer. Covers pulse and DTMF dialing. Includes schematic and assembler source code.

You should be able to order application notes from your local Motorola distributor. You can also contact:

Motorola Literature Center
P.O. Box 20912
Phoenix, AZ 85036

Where From Here?

I've become fascinated by the power available in such a small package. Building instruments such as VOMs and frequency counters would be a snap. Because the 'HC705 directly interfaces to switch matrices and multi-character LCD displays, you can easily build a large-display digital meter that monitors up to 11 voltages, using only two chips.

The FREEWARE BBS and application notes provide more design possibilities and information than I have time to deal with right now. However, I know I'll stay with this chip for a while. There is simply too much serious fun here to pass up!

Editor's Note

This kind of processor and this kind of support really make designing your own controller fun. (And, easy.) So I called United Products, my usual source for prototype boards and parts and asked if they carried Motorola's 6805 development kit.

"Nope." responded Phil Clyde. "But we have something you might like better. It's the Hitachi 68P05V07 with a built-in piggyback socket."

It turns out that the chip is compatible with Motorola's 6805 but you don't need a custom development kit. You just burn your code into a standard 2732 and piggy back the ROM onto the 68P05. To change the code you burn a new ROM. Plus, the chip costs only $19.95.

Hitachi has a data book and a programming book for this new release and I understand they supply cross assemblers that run on the PC as well as sample 6805 code.

Anyway, once you finish the prototype, whether it's on the Motorola development board or on the Hitachi piggyback version you have quite a number of options.

Both Motorola and Hitachi make a plethora of code-compatible parts in CMOS or NMOS, flat packs or dips—all available with more I/O lines, fewer I/O lines, more fast, less fast, more internal RAM, less internal RAM.... You get the picture.

For more information on the Hitachi 68P05 contact Phil Clyde at United Products 206-682-5025 or:

Hitachi America Ltd.
2210 O'Toole Ave
San Jose, CA 95131
(408) 435-8300

Additional Information

CMOS/NMOS Special Functions Data—Motorola publication DL130.

Technical Summary: MC68HC705C8—Motorola publication BR594/D.

M68HC05 Microcontroller Applications Guide—No Motorola publication number available; contact the Motorola Literature Distribution Center (address above).

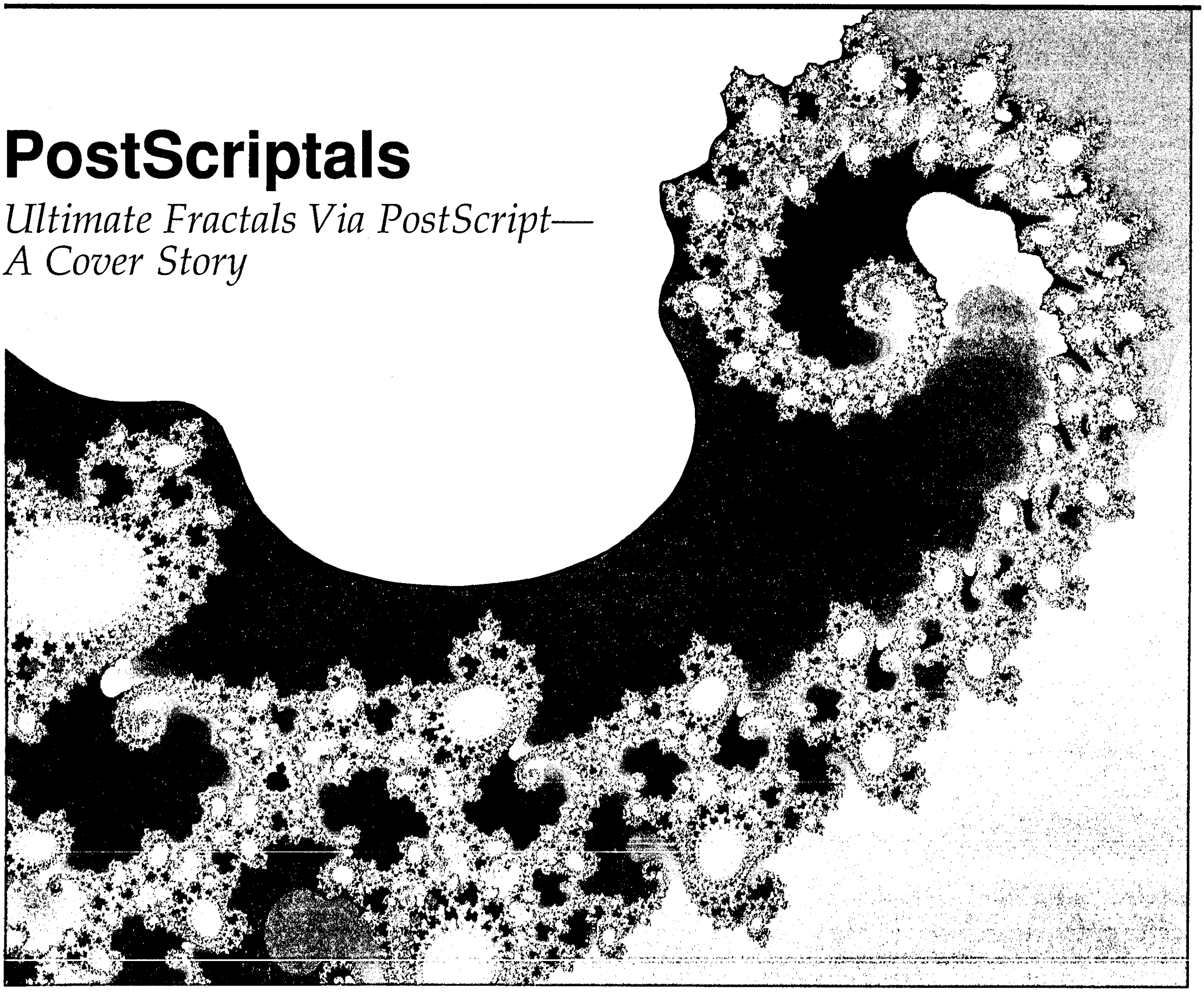
Microprocessor, Microcontroller and Peripheral Data, Volume 1—Motorola publication DL139.

M68HC05PGMR Programmer Board User's Manual #2—Motorola publication M68HC05PGMR2/D1.

◆ ◆ ◆

PostScriptals

Ultimate Fractals Via PostScript— A Cover Story



Larry's created the ultimate fractal—2540 dots per inch. (Unfortunately our print shop couldn't reproduce that kind of resolution, so we toned it down for the cover.) Here's how Larry created it.

It's hard to say when the notion first occurred; notions have a habit of sneaking in the door when I'm not paying attention, and later announcing their presence with a rude, "Where's dinner?" But sometime in the not too distant past, very high resolution printed output and fractals got linked up in my head.

See, the good folks at Salem Type (the ones responsible for typesetting *Micro C*) use a Linotype Linotronic 300, capable of printing 2540 dots per inch. That metal monster digests PostScript quite happily, and didn't I see some PostScript manuals lying around the office somewhere?

Think of it: 2540 dpi. Figure one of them fancy graphics cards displays 2K dots on a screen that measures around 10". That's something like 200 dpi. You can see why I started to daydream about printing the highest resolution fractals *ever seen by earthlings*. Right here on the cover of *Micro C*, too. Of course, it didn't quite turn out that way. So just slide your lawn chairs on up to the campfire and I'll tell you all about it....

PostScript

"What's a PostScript anyway," you ask. "And shouldn't it be at the end of this article?" Not really. PostScript is a page description language used to communicate with electronic printing devices. PostScript can describe all manner of graphic shapes, binary images, and printed text (in any orientation).

We use Ventura Publisher to generate PostScript output describing the pages of *Micro C*.

I never thought I'd be doing anything *remotely* FORTH-like, but you can't talk about PostScript without at least mentioning FORTH. That's because PostScript, like FORTH, is a stack-oriented language. Pascal, C, and many other high level languages might say—

```
c = a + b
```

where a, b, and c all have distinct locations in memory, apart from the stack.

But in PostScript, we have—

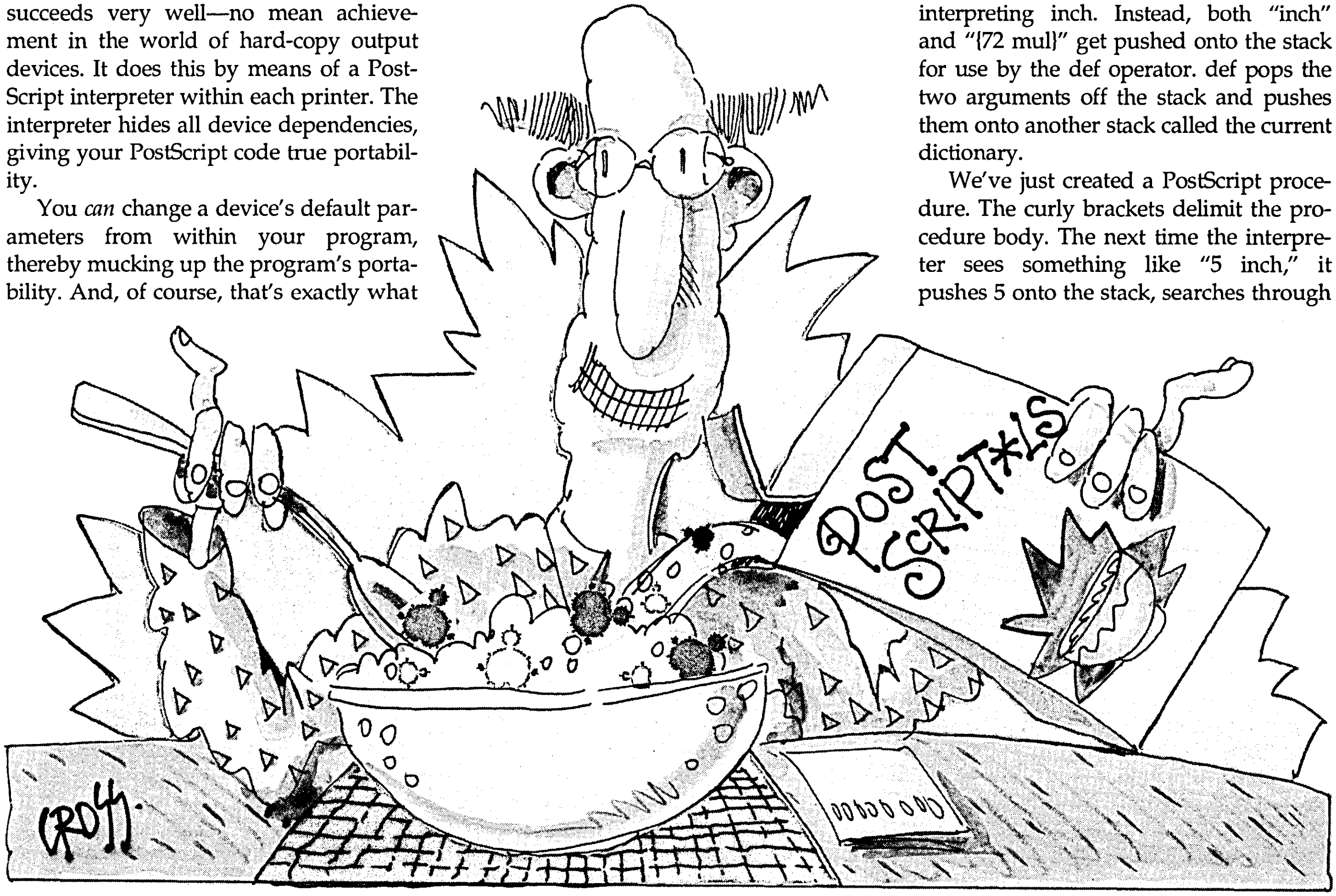
```
a b add
```

Here, a and b get pushed onto the stack, we invoke the addition operator, and the result, c, ends up on the top of the stack. Just like an HP calculator, or any other RPN device. Simple, but different.

By Larry Fogg
Micro C Staff

PostScript strives to be device independent. And, as far as I can tell, it succeeds very well—no mean achievement in the world of hard-copy output devices. It does this by means of a PostScript interpreter within each printer. The interpreter hides all device dependencies, giving your PostScript code true portability.

You *can* change a device's default parameters from within your program, thereby mucking up the program's portability. And, of course, that's exactly what we'll do. But, I'm getting ahead of my story.



The Basics

Adobe Systems created PostScript and wrote three books that make up the PostScript Rosetta stone. They do read a bit like an IBM Technical Reference, but with a much better plot. (Anyone who quotes Woody Allen and Napoleon in the same book can't be all bad.) If you want to do serious work with a PostScript application, buy these books.

To work! Let's build the cover illustration for this issue of *Micro C*. Press-folk need alignment lines—called crop marks—to keep the work from being printed sideways. So first we need line drawing capability.

PostScript defines a page with origin at the lower left corner, a comfy spot for the mathematically inclined among us. The default measure is the point, with 72 points to the inch. Printers like points, but I grew up with inches. The PostScript line—

```
/inch {72 mul} def
```

teaches the interpreter the meaning of inch. The slash keeps the interpreter from interpreting inch. Instead, both "inch" and "{72 mul}" get pushed onto the stack for use by the def operator. def pops the two arguments off the stack and pushes them onto another stack called the current dictionary.

We've just created a PostScript procedure. The curly brackets delimit the procedure body. The next time the interpreter sees something like "5 inch," it pushes 5 onto the stack, searches through the current dictionary until it finds "inch," and pushes its definition onto the stack also. So, "5 inch" becomes "5 72 mul," or 360 points. Just right.

Variable assignments work the same way.

```
/pi 3.1416 def
```

assigns a value to pi in the current dictionary.

The term current dictionary implies that not-so-current dictionaries exist. That's the way the scope of variables and procedures can be controlled; variables and procedures exist only within the dictionary of their birth.

Line Drawing

To draw a line in PostScript, we deal with paths. The following code draws the first crop mark in the lower left corner of the page.

```
newpath
0.01 inch setlinewidth
1 inch 0.9 inch moveto
0 inch -0.5 inch rlineto
0.9 inch 1 inch moveto
-0.5 inch 0 inch rlineto
stroke
showpage
```

The newpath initialization deletes any path that might be lurking about. setlinewidth overrides the default width for lines. moveto does just what you'd expect. It takes the x-coordinate (1 inch), then the y-coordinate (0.9 inch) and moves to that position. (This x,y order for parameters remains consistent for all PostScript operators.)

rlineto stands for "relative line to" and constructs a line path to a point 0 inches along the x-axis, and -0.5 inches along the y-axis, relative to the point we moved to in the third line of code. You can also use an absolute coordinate version called lineto. But I prefer rlineto since you never really have to know where you are.

By the way, don't yield to the temptation to code something like "NewPath"; PostScript's case sensitivity will shoot you down in flames.

stroke actually draws the line; until now it was defined, but invisible. As you can see from the example, a path need not be connected. Finally, showpage outputs our efforts to the printer—at least it does in theory.

The Un-text File

PostScript interpreters take pure ASCII text. I made the rash assumption that my trusty Turbo Pascal editor could generate PostScript programs. Well, it can—except for the last byte.

When I found that even a single line program (showpage) wouldn't execute, I dug out my documentation (always a last resort) and found the answer. PostScript files require a ^D end-of-file mark instead the more familiar MS-DOS ^Z. You can still use your favorite editor (in ASCII mode) to write PostScript programs, but you'll need to take an extra step.

If your text editor lets you insert control characters, then append a ^D to the code. Otherwise fire up a disk editor like EASY-ZAP (on Micro C user disk #MS-15) or even DEBUG (if you're desperate) and add a ^D (the EOF).

(After a quick rampage through the text editors at Micro C, it looks as though most will preserve the ^D once you've entered it. So you shouldn't have to edit the EOF after every file save. Your editor may tack on a ^Z for good measure, but the PostScript interpreter won't care.)

A bit of a bother, what? I got around the problem by generating PostScript code with a bunch of fprintf() statements in C. A final putc() writes the ^D EOF. You might think it odd to use C to generate a text file for an interpreter. But it makes sense in this case since we have to use C anyway to crank out the massive amount of text defining each PostScriptal (over 3.5 MB for our cover).

Binary Images

A binary image provides the simplest way to do our first PostScriptal. (Carol, one of our many office clowns, thought PostScriptals was a new breakfast cereal.) *Editor's note: This is probably the only chance I'll get to tell you that Larry really eats Sugar Frosted Flakes, the breakfast of nerds.*

Just like monochrome video, a binary PostScript image relates one bit of data to one dot of output. So we can control each of the 2540 dots in an inch of L300 output, or each of the 300 dots per inch on our laser printer.

Figure 1 shows the PostScript code to print a monochrome, 1500x1500, 5" square image in the center of an 8½"x11" sheet of laser printer output. Right off you'll see the means for commenting; a percent sign directs the interpreter to ignore all text until the end of the line.

The procedure called fractal introduces the image operator. image takes its operands like this: number of horizontal elements in a line of the image (1500); number of lines (1500); bits per image element (1 here, but can be 2, 4, or 8 for grey-scale images); transform matrix; and a procedure to provide the image data.

The transform matrix, as written—

```
[1500 0 0 1500 0 0]
```

reproduces our 1500x1500 image with no changes. But consider an image with origin at the *upper* left corner, like a video image. We'd like to invert it to match PostScript's expectations. The following matrix will do the inversion.

```
[1500 0 0 -1500 0 1500]
```

The other matrix elements define rotation and scaling of the image. But PostScript operators exist to perform these functions much more easily, so we won't fool with the elements containing zeros.

Next, image's data procedure reads strsize hexadecimal bytes from the current PostScript file being interpreted. For us, that's one horizontal line. Note that a hexadecimal byte is represented by two ASCII characters in the PostScript file. PostScript expects the image data to be the first thing following the call to fractal in the main program.

image repeats the data procedure until it has processed enough data to fill the image defined by the first three parameters.

Be sure to provide exactly the right

Figure 1—Ultimate Resolution Laser Printer PostScriptal

```
% definitions
/strsize 188 string def       % set length of hex string

% procedures
/inch {72 mul} def            % "inch" procedure

/fractal                      % fractal procedure
  {1500 1500 1 [1500 0 0 1500 0 0]              % set up transform matrix
    {currentfile strsize readhexstring pop}  % read the hex data
    image                     % generate the image
  } def                       % end of fractal definition

% main program
1.75 inch 3 inch translate    % locate image in center of page
5 inch 5 inch scale           % make it 5 inch by 5 inch
fractal                       % call the fractal procedure
  0F12...hex data...3C0A      % padded hexadecimal image data
showpage                      % print the page
% end program
```

◆ ◆ ◆

Figure 2—Monochrome Hex Data Generation

```
unsigned char color, count, buffer, mask;

count = 7;     /* corresponds to location of point within data byte */
buffer = 0;                              /* this will hold 8 points */
for (each point in the image)
{
  find color (color equals 0 or 1) /* Mandelbrot code provides this */
  mask = color << count;     /* shift color to the correct position */
  buffer |= mask;                          /* place it in the buffer */
  if (count == 0)                        /* is the 8-bit buffer full */
  {
    if (buffer < 0x10)  /* pad hex byte with leading 0 if necessary */
      fprintf (fp, "0");
    fprintf (fp, "%x", buffer);    /* print the byte in hex format */
    count = 7;     /* reset these in preparation for next 8 points */
    buffer = 0;
  }
  else                   /* buffer ain't full, so get another point */
    count--;
}
```

◆ ◆ ◆

amount of data, since image will read as much as it's told, regardless of how much data you actually place in the file.

Here, the right amount means too much. We want to print 1500 dots per line. With eight dots per byte of data, 187 1/2 bytes would do the trick. But image doesn't understand nybbles and would get out of synch by four dots every line. So the first line of Figure 1 assigns 188 to strsize and image throws out the extra nybble at the end of each line.

The main section of the program locates the image at the center of the page, scales it to 5"x 5", and calls fractal. In real life, all 282,000 bytes of data would follow fractal. But I didn't think it necessary or prudent to print each and every one; just use your imagination.

Images defined by a manageable amount of data can use a direct data procedure like—

```
{<0F12A377>}
```

"<" and ">" tell the interpreter that it's dealing with hex values. I used this method at first and it worked fine until I got into larger images. It turns out that a hexadecimal string cannot exceed 64K. This wasn't obvious to me, so I went whining to Adobe's developer's hotline and had the answer in seconds. Good outfit.

Generating The Hex Data

I won't bore you with the specifics of the Mandelbrot code again; it's been covered in *Micro C* before (issues #39 and #42). Log onto our BBS or get the issue disk for a look at the complete program.

Figure 2, a mixture of pseudo-code and C, shows the steps necessary to fill a byte with 8 points of the image, and print the byte to a file. fprintf() conveniently

QEdit™ ADVANCED

The fast, easy to use, fully featured text editor at an affordable price.

THE Quick EDITOR

NOW AVAILABLE FOR OS/2 AT THE SAME LOW PRICE OF ONLY $54.95

"This small, blazing-fast editor lets program instructions, memos, letters, and assorted text databases flow easily between brain and computer."
David M. Kalman, Editor-in-Chief, Data Based Advisor (September, 1988)

"The editor's speed, windows, and other features make it among the best text editors I've ever used."
George F. Goley IV, Contributing Editor, Data Based Advisor (September, 1988)

If you are looking for the right combination of price and value in a text editor, then give QEdit a try. At ONLY $54.95 and a money-back guarantee—you just can't go wrong.

QEdit is fast, easy to use, and simple to install. At the same time you get all of these features and more.

- Completely configurable, including keyboard and colors
- Edit as many files simultaneously as will fit in memory
- Open up to eight windows
- 99 scratch buffers for cut-and-paste or template operations
- Exit to DOS (or a DOS shell) from within QEdit
- "Pop-Down" menu system and customizable Help Screen
- Column Blocks
- Easy to use macro capability including keyboard recording
- Wordwrap and paragraph reformat capabilities
- Recover deleted text
- Automatic indentation for C programming
- Import files and export blocks
- Locate matching braces and parentheses
- Execute command line compilers from within QEdit
- QEdit supports 101 key keyboards, EGA 43-line mode, and VGA 50-line mode
- Great for use with laptops—QEdit edits files entirely in memory, saving drain on laptop batteries
- Compact—Even with all these features, QEdit requires less than 50k of disk space

Full 30 day money-back guarantee

System Requirements
QEdit requires an IBM PS/2, PC/AT, PC/XT, PC, PC/Jr, or compatible. Minimum system requirements are 64 KB of memory, PC-DOS 2.0 or MS-DOS 2.0 or greater, 50 KB of disk space. QEdit runs GREAT on floppy based systems and laptops.

To order direct call
404-428-6416
MasterCard VISA AMERICAN EXPRESS Cards
Add $3.00 for shipping—$10.00 for overseas shipping. UPS 2nd DAY AIR available within the U.S. for ONLY $5.00
COD's accepted—please add $3.00
Georgia residents add 4% sales tax

SEMWARE™
730 Elk Cove Ct. • Kennesaw, GA 30144

QEdit and SemWare are trademarks of Applied Systems Technologies, Inc.
© 1989 Applied Systems Technologies, Inc.

Reader Service Number 127

translates buffer into its ASCII hexadecimal equivalent. But watch out for single digit hex numbers; PostScript's image operator demands two digits for each hex byte. So any number less than 0x10 gets a leading zero.

Color Printing

"...under suitable conditions, full color can be represented as three color separations each defined by a gray-scale image. Such image processing is beyond the scope of this manual." (From p. 72 of the *Postscript Language Reference Manual.*)

I just love it when they talk like that. It's a challenge, a gauntlet thrown. We've done monochrome, but color's a must for the cover.

We print *Micro C* on a four-color press: yellow, magenta, cyan, and black. By laying down different amounts of each of the four colors, we can build a staggering number of new colors. But we can't do *any* color. Complications arise due to factors like the color printing order, and the discrete, rather than continuous, values for color intensities. Six-color presses exist that can print the normal four-color options we have, plus two custom mixed colors. But that's another story.

If you look at a printed page closely (use a magnifying glass), you'll see that it contains an array of dots, just like a video image. A square inch of image always has the same number of dots. But if each of those dots increases in size, the square inch becomes darker.

The array defining the dot locations is called a screen. A 0% screen has dots of zero size and gives a totally light (blank) image. The dots of a 100% screen are full size and give the darkest image. Screens between these extremes produce shades of color. To build a new color, we just specify a percentage for each of the four press colors.

Creating Halftone Images

Typesetters and laser printers print dots, that's all. On or off. And they print one size dot only. (Not quite true. The L300 has two modes: single density at 1270 dpi, and double density at 2540 dpi. But within each mode only one size dot exists.) In order to simulate shades of grey, PostScript uses halftone cells. Each of these cells contains some number of output device dots. By varying the number of dots printed within a cell, we can vary the darkness of the cell.

Earlier, I mentioned discrete levels of color; we could call this part the quan-

Figure 3—Generation of Four-color Separation for PostScriptal

```
void mandel(), file_header (), file_tailer (),
     draw_points (), get_layers ();
FILE *fp1, *fp2, *fp3, *fp4;


main()
{
  fp1 = fopen ("frac1.ps", "w");           /* open yellow layer file */
  fp2 = fopen ("frac2.ps", "w");          /* open magenta layer file */
  fp3 = fopen ("frac3.ps", "w");             /* open cyan layer file */
  fp4 = fopen ("frac4.ps", "w");            /* open black layer file */
  file_header (fp1);  file_header (fp2);      /* defs and procedures */
  file_header (fp3);  file_header (fp4);
  mandel ();          /* generate hex data defining Mandelbrot image */
  file_tailer (fp1);  file_tailer (fp2);   /* finish main part of... */
  file_tailer (fp3);  file_tailer (fp4);   /* ...PostScript programs */
  fclose (fp1);  fclose (fp2);                        /* close files */
  fclose (fp3);  fclose (fp4);
}  /* main */


void mandel ()
{
  int column, iterations;
  int point1, point2;

  for (each point in the 750x1200 image)
  {
    find iterations (Mandelbrot code provides this)
    if ((column % 2) == 0) /* this is the first of a pair of points */
      point1 = iter;
    else                           /* this is the second of the pair */
    {
      point2 = iter;
      draw_points (point1, point2);                /* draw the pair */
    }
  }
}  /* mandel */


void draw_points (pt1, pt2)
int pt1, pt2;
{
  unsigned char color;
  unsigned char yellow, magenta, cyan, black;
  unsigned char y_temp, m_temp, c_temp, b_temp;

  yellow = magenta = cyan = black = 0;
  y_temp = m_temp = c_temp = b_temp = 0;
  if (pt1 == maximum_iterations)       /* it's in the Mandelbrot set */
  { /* these color assignments draw the point in a deep black color */
    magenta = 0x90;/* assignments made to high order nybble for pt1 */
    cyan = 0x30;
    black = 0xf0;
  }
  else                                /* point not in Mandelbrot set */
  {
    get color (from 0-81)       /* according to number of iterations */
    get_layers (color, &y_temp, &m_temp, &c_temp, &b_temp);
    yellow = y_temp << 4; /* shift assignments to high order nybble */
    magenta = m_temp << 4;
    cyan = c_temp << 4;
    black = b_temp << 4;
  }
  y_temp = m_temp = c_temp = b_temp = 0;
  if (pt2 == maximum_iterations)       /* it's in the Mandelbrot set */
  {                                                /* make it black */
    magenta += 0x09;             /* pt2 goes in the low order nybble */
    cyan += 0x03;
    black += 0x0f;
  }
  else                                             /* not in the set */
  {
    get color (from 0-81)
    get_layers (color, &y_temp, &m_temp, &c_temp, &b_temp);
    yellow += y_temp;
    magenta += m_temp;
    cyan += c_temp;
    black += b_temp;
  }
  if (yellow < 0x10)     /* pad single digit (hexit?) numbers with 0 */
    fprintf (fp1, "0");
```

```
  if (magenta < 0x10)
    fprintf (fp2, "0");
  if (cyan < 0x10)
    fprintf (fp3, "0");
  if (black < 0x10)
    fprintf (fp4, "0");
  fprintf (fp1, "%x", yellow);   /* write the values to the 4 files */
  fprintf (fp2, "%x", magenta);
  fprintf (fp3, "%x", cyan);
  fprintf (fp4, "%x", black);
}  /* draw_points */


void file_header (f)
FILE *f;
{
  fprintf (f, "/inch {72 mul} def\n");             /* define "inch" */
  fprintf (f, "/picstr 375 string def\n");   /* bytes per image line */
  fprintf (f, "/fractal\n");                   /* fractal procedure */
  fprintf (f, "  {750 1200 4 [750 0 0 1200 0 0]\n"); /* build image */
  fprintf (f, "  {currentfile picstr readhexstring pop}\n");
  fprintf (f, "image\n}def\n");

  fprintf (f, "150 ");                       /* set 150 line screen */
  if (f == fp1)            /* yellow layer - 90 degree screen angle */
    fprintf (f, "90 ");
  if (f == fp2)                 /* magenta - 75 degree screen angle */
    fprintf (f, "75 ");
  if (f == fp3)                   /* cyan - 105 degree screen angle */
    fprintf (f, "105 ");
  if (f == fp4)                  /* black -  45 degree screen angle */
    fprintf (f, "45 ");
                                           /* obscure spot function */
  fprintf (f, "{dup mul exch dup mul add 1 exch sub} setscreen\n");
  fprintf (f, "1.75 inch 1.5 inch translate\n");/* center the image */
  fprintf (f, "5 inch 8 inch scale\n");            /* make it 5"x8" */
  fprintf (f, "fractal\n");               /* call fractal procedure */
}  /* file_header */


void file_tailer (f)
FILE *f;
{
  fprintf (f, "showpage\n");                      /* print the page */
  putc (0x04, f);                          /* ^D EOF for PostScript */
}  /* file_tailer */


void get_layers (clr, y, m, c, b)
unsigned char clr;
unsigned char *y, *m, *c, *b;
{
  switch (clr)
  {
    case 0: *m = 0x0f; *c = 0x0b; *b = 0x06;
            break;
    case 1: *m = 0x0d; *c = 0x0b; *b = 0x06;
            break;
    case 2: *m = 0x0c; *c = 0x0b; *b = 0x06;
            break;

    /* etc. */

    case 79: *y = 0x02; *m = 0x0f; *c = 0x0c;
             break;
    case 80: *y = 0x01; *m = 0x0f; *c = 0x0c;
             break;
    case 81: *y = 0x00; *m = 0x0f; *c = 0x0c;
             break;
  }
}  /* get_layers */
```

◆ ◆ ◆

tum theory of color printing. Since each cell contains a finite number of dots, the cell can take on only a finite number of shades of grey—not a continuous spectrum. I wouldn't worry about it though. Even if we can do only 16 shades in each of the four colors, we still have 16^4 possible combinations, or colors; 64K different colors should be enough to please anyone.

PostScript does the halftoning work for us. In the image operator example above, we specified one bit per image element for a monochrome image. Let's go to four bits per element. Two differences arise in the code. (Figure 3 shows the important program segments interspersed with pseudo-code.) First, each byte of hexadecimal image data now contains two points instead of eight. No real problem—we just shift left by four bits after loading a point into the data byte, instead of by one.

The second difference makes life interesting. Instead of creating one file, we need four, one for each layer of the color separation. But how do we choose the color intensity for each layer?

Color Assignments

I would have liked to treat the color assignment a little more intelligently, but it's an inherently stupid process. Shift gears and think about character generation for a moment. Once you've defined a matrix of dots for the letter "A," you can't just add one and get "B." I don't think you can say "puce + 1" either, so I opted for brute force.

Carol and I sat down with a color guide and picked out a bunch of good lookin' colors, arranged in a more or less continuous spectrum. Then, for every color, I approximated the percentages for each of the four layers of the separation (by choosing the 1-digit hexadecimal number closest to the percentage required). Then I added a bunch of new colors (since I had 16 levels to chose from instead of the 11 shown in the color guide), and plugged the values into that big, ugly switch statement in get_layers().

Odds And Ends

A few dangling mysteries and we'll be in good shape. In file_header(), there's a new operator called setscreen. The L300 defaults to an 80 line screen, or 80 halftone cells per inch. But we print *Micro C* at 150 lines. So the first of setscreen's operands forces the L300 to a 150 line screen setting. And here's where we finally muck up the program's device independence. Here at Micro C, 150 line screens look truly awful on the laser printer. A 300 dpi device works best with 60 line screens and doesn't appreciate being asked to do the impossible.

We also have to consider how the four layers of regularly spaced halftone cells might interfere with each other. If we don't rotate the layers correctly, wild moiré patterns appear. These separate rotations for each layer, called screen angles, minimize interference problems. Give the screen angle to setscreen as its second parameter.

Finally, setscreen requires a spot function. The spot function describes how halftone cells fill in as they get bigger. I lifted the version in file_header() directly from Adobe's documentation. (Have I no shame?) It grows from the middle of the cell out. I don't have the slightest idea how the function works.

Confession Time

The more astute (awake) among you will be grousing, "Hey! He said we'd be looking at 2540 dpi stuff. Where'd this 150 dpi ca-ca come from?" It's not my fault. I thought the presses could print anything we sent them. Not so; 150 line screens already push the mechanical limits of initial alignment and subsequent drift of the four layers.

So we're limited (in print) to 150 cells per inch—still better than most displays. If you want to see the unprintable 2540 dpi marvels, you'll just have to come by the Micro C World Headquarters in Bend and ask for a tour (open daily, more or less, unless the weather's nice).

Execution Speed

I pruned all the video stuff from Harlan Stockman's '386, 32-bit, fixed-point Mandelbrot code (see *Micro C* issue #43) to do the dirty work for these PostScriptals. Harlan used DeSmet C88, with assembly for the number crunching. The results were nothing short of amazing. A few statistics on the generation of the cover—

P_{max} = -0.7408
P_{min} = -0.7504
Q_{max} = 0.12036
Q_{min} = 0.105
`maximum iterations = 2000;`
`switch point = 200;`
`points in image - 750x1200`
`system - 20 MHz 80386`
`execution time = 2 ½ hours`

To place this in perspective, a standard 4.77 MHz XT, running Turbo C floating point code with no 8087, would take 87 days to do the same PostScriptal. I'd call that progress. Thanks, Harlan.

Early on I thought it would be nice to zoom way in for a PostScriptal. Harlan's fixed point representation can't deal with numbers smaller than 0.00001. So I switched over to the Turbo C floating point code and Dave picked up a 20 MHz '387 math coprocessor. The fixed point code was *still* five times faster.

You can see that I really couldn't have done this without Harlan's code and a handy '386. Although, I had to wonder about the hardware. Our '386 has a mind of its own—sometimes no mind of its own. At one point, it favored locking up during program compilation. The problem seemed to go away after the machine had been on for a while.

Aha, a heat-related hardware problem. But it only happened while using Turbo C. Perhaps the compiler needs to warm up, opined Dave. I had visions of a nice corduroy jacket to keep all the files on the hard drive toasty warm during the cool northwest eves; a compiler cozy if you will. But the problem actually came from our EMS management software. With that software removed, PostScriptal development went on unhindered.

Fini

As I pound these words into my keyboard and try to stay awake, our cover-to-be cruises the Cascades on its way over from Salem Type; nestled in a UPS semi, oblivious to the full midsummer moon.

It's passing strange to write about a cover that doesn't exist yet. If this really works, the result will be beautiful. If not, the cover will probably be blank (except for a "This page intentionally left blank" message), and I'll be on permanent vacation.

Plenty of folks lent a helping hand on this project. Thanks to Dan at Salem Type, and Stan and Patty at Bettis and Associates. But especially to Carol at Micro C: ostensibly our graphic design person, but better known internationally for her early morning sludge (excuse me, coffee).

Well, saddle pals, I didn't get around to discussing flow control operators, recursion, or a host of other PostScript capabilities. Do I smell a sequel? Stay tuned to Issue #50 to find out.

References

PostScript Language Tutorial and Cookbook, by Adobe Systems, Addison-Wesley, 1985, ISBN 0-201-10179-3. (Includes further references.)

PostScript Language Reference Manual, by Adobe Systems, Addison-Wesley, 1985, ISBN 0-201-10174-2.

PostScript Language Program Design, by Adobe Systems, Addison Wesley, 1988, ISBN 0-201-14396-8.

◆ ◆ ◆

By Ron Hicks
819 S. 21st St.
Sheboygan, WI 53081

Programmable Logic Controllers

Industrial Control Is No Longer A Relay Race

In many ways this feels like "Bits From Your Past." Just a few months ago I watched a pressman cussing the relay panel on his press. Actually I didn't know it was a relay panel until I looked inside.

That box contained a hellish mess of color-coded wires (long ago coated a uniform black) and relays clacking and sparking as they connected and disconnected 220 VAC. The hearts of these ancient machines aren't the motors or the fancy rollers; they're the relays, the cussed, nasty, grimy, indispensable relays. Read on for information on new hearts.

You may think this article seems out of place in a micro computer magazine, but it's not. You see, Programmable Controllers are nothing more than micro computers interfaced to "real world" inputs and outputs.

In its simplest form, a programmable controller consists of a central processing unit (CPU) attached to inputs and outputs. The CPU has a program that tells it to energize or de-energize outputs according to the state of its inputs.

PLC Uses

Industry uses programmable logic controllers (PLCs) for a variety of tasks. You'll find small PLCs built into electronic motor controllers, machine control panels, and even vehicles! They oversee current draw, speed, and load factors; they monitor limit switches and operator inputs. The typical small PLC will have from 16 to 200 I/O points.

Larger PLCs can control whole assembly lines or entire factories. Industrial robots and their support equipment (conveyors, tool holders, power units, etc.) are also major users of PLCs.

Not only do they control equipment, but they also report system faults to the central computer so technicians can be dispatched. Many of these units also log data and generate production or maintenance reports. These larger PLCs typically have 200-2400 I/O points.

Of course, PLCs can be networked via mainframes. This way, any machine (or human) on the network shares the I/O and data of all the units. And, software developed on the mainframe can download to one or more controllers.

A Brief History of PLCs

One of the early marvels of electrical engineering was the relay. Using a relay, an engineer could control electrical processes using logic.

Relay logic is nothing more than a hard-wired program. You can control the relay that starts the drill motor by another relay that monitors table position. If the table isn't in position, the motor won't start.

This approach worked fine until machines started becoming more sophisticated. Imagine using relays to also sense drill speed, part positioning, tool positioning, coolant pump operation, motor overload, and spindle position!

Soon the relay panel necessary to operate a machine becomes more complicated than the machine. Where I work, we have panels containing over 500 relays, each! I've spent many hours trying to find bad relays by substituting good ones for the suspects. It isn't much fun.

Computers Enter The Scene

As computer equipment became cheaper, electrical engineers started replacing relay logic with CPU logic. Many of the early PLCs were based on discrete minicomputers.

Modcomp was one of the premier names during the first generation of industrial computers. I have one of their early machines complete with 32K of magnetic core memory. In its heyday, this unit sold for $24,000!

As microprocessors came on the scene, most manufacturers moved on to the 8080 (and later the Z80) and many PLCs still use these chips. The other main chip family used is the INTEL 8088/80188. These machines fall into the second generation.

As with other computers, newer PLCs have vastly increased computing power and much lower prices. Small systems go for $200 and up, while large systems start at about $3,500.

Ladder Diagram Programming

A problem arose when PLCs replaced relay panels. Plant maintenance personnel and electricians were the main installers and operators of this new technology. It was easy enough to follow the installation drawings since they continued to be standard electrical ladder diagrams. Programming was another thing. Many of the early machines were programmed in boolean algebraic equations, hardly the forte of the average maintenance mechanic!

The obvious answer was to use the already familiar ladder diagrams. Ladder diagram programming (LDP) uses graphic images of contacts and coils representing the familiar relay circuits.

Within a ladder diagram, an individual line of code is called a rung.

LDP Rung

```
—[ ]——[/]———o—
  A    B     C
```

The above example shows a normally open contact (A), a normally closed contact (B), and a coil (C). The contacts can be assigned to real inputs so an energized input (A) and a non-energized input (B) will cause coil (C) to energize.

Coil (C) can be assigned to a real output as a motor starter or other electrical signal.

```
——[ ]——+————o-
  A    |     C
       |
——[ ]——+
  B
```

The above rung shows the coil (C) energizing when either Contact A or B energizes. Using these simple elements, service people can generate complex logic statements.

In an LDP program, power always flows from left to right. Although these examples are very simple, LDP programs can have up to 100 elements in each programming screen and programs may have up to 100 screens.

Each PLC's manufacturer has its own proprietary LDP. Some implementations have added features such as subroutines and advanced mathematical features. The best of the LDP languages support a BASIC-like language composed of keywords that can be used in the same networks as the relay contacts. I've shown an example of an advanced ladder language below.

```
-[ ]-[TSEC]-[COND]-[SUB50]-
 A


TSEC = A second timer.
COND = A conditional statement
       ie. TSEC=10
SUB50 = Goto subnetwork number 50
```

When contact A closes, TSEC will start timing. When TSEC equals 10, COND will pass power to SUB50. This will transfer program execution to subnetwork number 50.

Programming And Control Computers

Many different machines are used as programming and control computers. CP/M-based Kaypro machines were very popular for a while, as were VAX computers. However, with the popularity of the IBM PC, most companies have switched to the PC standard for their programming and control computers.

Many companies make "hardened" IBM-PC/AT compatibles for industrial users. They have sealed cases, shock-mounted hard disks, and super heavy-duty power supplies (to support extra expansion cards). In dirty or wet environments they use membrane keyboards. At a recent trade show, one vendor had his AT compatible running in an aquarium with fish swimming around the EGA monitor!

System Setup

Typically, they'll bring in a PLC, connect the I/O, and then use a PC compatible to generate the program. This may seem opposite of normal wisdom (wiring before programming); but once the I/O is connected, you can do all reconfiguration through the LDP program.

Once the program has been downloaded to the controller, you can disconnect the PC. However, for maximum control, you can leave the PC connected and let it serve as a gateway into a network.

Many companies make "hardened" IBM-PC/AT compatibles for industrial users... sealed cases, shock-mounted hard disks, and super heavy-duty power supplies (to support extra expansion cards). In dirty or wet environments they use membrane keyboards.

When the programmable logic controller is running, you can use the IBM-PC to monitor its activities. The IBM-PC can force I/O on or off, which really helps during troubleshooting. It can also examine and change the PLC registers.

PLC Hardware

Let's examine a typical PLC CPU and its method of operation.

The CPU designates a block of memory as the I/O image area. This block holds the status of all the inputs and outputs. The I/O image is checked at the beginning of each program scan.

When the program needs to energize an output, for instance, it loads the new status into a reserved section of memory called the output image area. At the end of the scan, the processor checks output image area and turns on the output.

The PLC's ROM contains the executive operating program. In many ways, this executive can be compared to CP/M or MS-DOS. It handles communications between the user program and the hardware. The executive program also handles math and timer functions, and translates ASCII text.

Digital Inputs

The most common industrial input to a PLC is 120 VAC. Other common inputs are 24 VDC and 5 V TTL. All these inputs are handled the same way, only the components change.

The circuit drawing in Figure 1 shows a common way of interfacing process signals to the PLC bus.

As the 115 VAC signal comes into the card, resistors R1 + R2 lower the voltage. Then the AC signal goes to full wave bridge D1 for conversion to DC. Next R3 and C1 filter the signal and D2, a Zener diode, limits the voltage to the relay.

The operation of the LED confirms the relay coil is energized. A set of contacts on the relay allows a signal to flow to the input of a quad bilateral switch.

A typical input card will have 16 of these inputs. It will have 4 of the quad bilateral gates, which the CPU will scan as a 16 bit word. Energized inputs will read as 1 and unenergized inputs will read as 0.

Output Cards

Take a look at Figure 2. When an output is required, the CPU will send a signal to the card. This signal drives the base of T1 and energizes the relay coil. The LED confirms that the relay coil is energized.

D2 bleeds off the reverse voltage generated when the relay's field collapses. A set of contacts on the relay controls the external device.

Analog Inputs

The card also converts analog signals to binary. In this way the CPU can read an analog input as 16 bits—the same way it read the 16 inputs above. The LDP pro-

Figure 1—PLC Bus/Process Signal Interface.

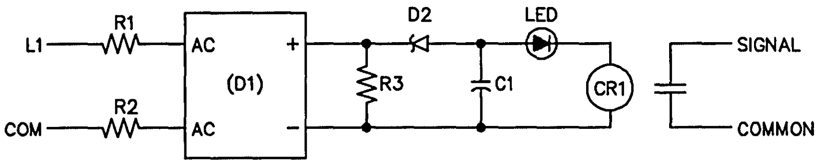


Figure 2—PLC Output Drives.

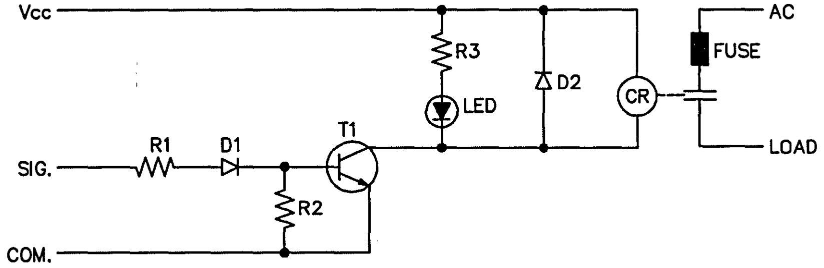
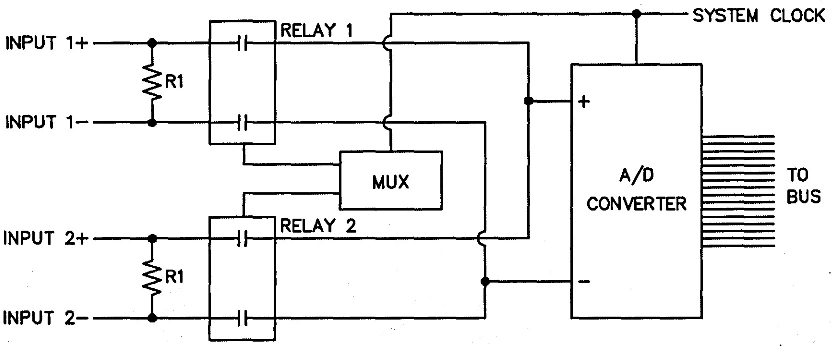
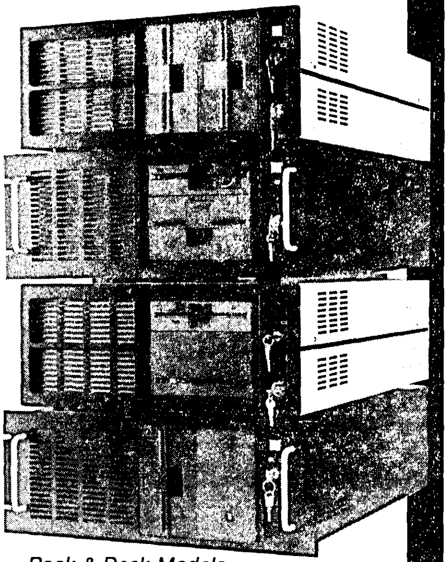
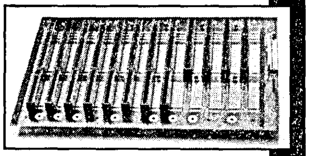


Figure 3—Analog to Digital Conversion.







gram tells the CPU what to do with the data. The program can convert it to a binary coded decimal, integer, or floating point number before storing it in a user register.

In Figure 3, the signal is input through a reed relay. A multiplexer then selects which input to channel to the A/D converter by turning on a relay. The typical analog card will have 8 inputs, each input capable of handling a current or voltage (or thermocouple) input.

Analog Outputs

In Figure 4, the analog output operates in the reverse of the analog input. Each output has a dedicated D/A convertor. This D/A converter drives a current source, thus outputting an analog signal in proportion to the digital information from the CPU.

A register on the D/A converter stores the digital signal so the output will remain constant until the CPU tells it to change or shut up.

Servo Controller Cards

A servo motor controller is one major PLC use. Servo motors are used for robots, part positioning, machine control, and a plethora of other uses. Servo control consists of two separate functions: Sensing motor position and changing motor position.

Servo controller cards have an output driver and an input (connected to servo motor's encoder). The encoder reads the motor's spindle position. (See Figure 5.)

The LDP program decides where the motor should be. The CPU sends a signal to the output section of the controller, which sends the output to the servo motor. The encoder then verifies the motor's new position.

If the motor's shaft hasn't reached the correct position, the controller card detects the error and sends a correction (via the output) to the motor. This way, the CPU doesn't need to check for proper position.

Conclusion

I've just covered the basics of PLC use in industry. There are over one hundred companies making programmable controllers, and every day I get literature from new companies.

Although PLCs have been around for 20 years, it seems 1988 was the year they came into dominance of industrial control. Perhaps the long promised pushbutton world is finally here, thanks to programmable controllers.

Figure 4—Digital to Analog Conversion.

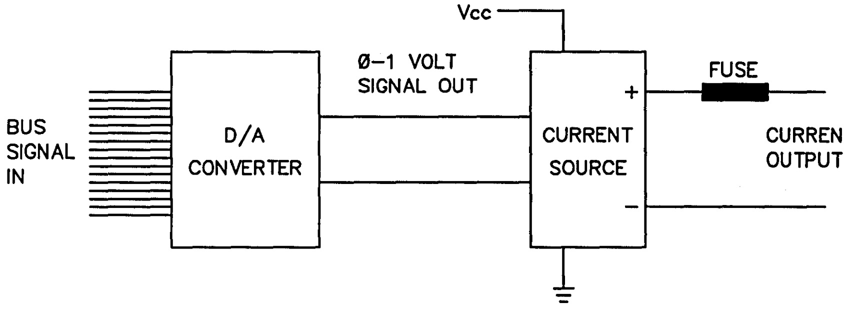
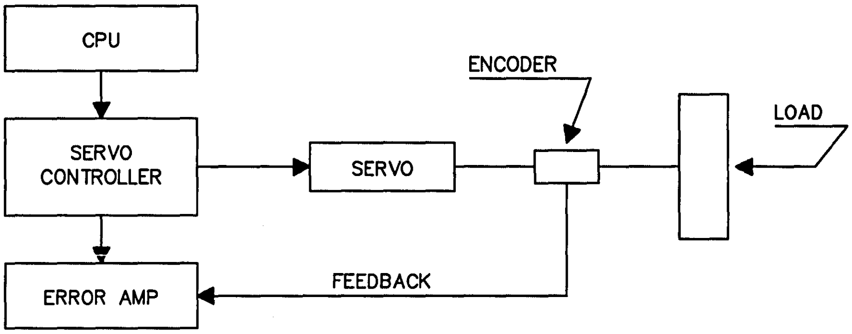


Figure 5—Servo Controller Block Diagram.



Computer Jobs In Industry

Many computer users take their knowledge for granted. We assume that everybody knows how to use a computer. This is *not* the case. In most industrial plants, they look upon computers as a necessary evil. Many senior maintenance men, who think nothing of pulling apart a steam turbine, freeze when they're called upon to touch a computer. Some have never had the chance to use computers, and some just aren't interested. This gives computer users a real opportunity for employment.

Industry requires some skills that seem trivial to us. Simple things like copying a disk or changing a hard drive directory appear insurmountable to an untrained user. More advanced tasks, such as batch file writing or 123 work sheet configuration, are other common problems. I found a user who turned off the computer each time he wanted to change programs. That was the only way he could get back to the root directory!

When you fill out an application at an industrial plant, be sure you include any computer courses you've taken. If you haven't taken any courses, still include computer use in your application. Applications usually have an "other experience" category.

For instance, if you can't think of anything else to write, at least write in "Extensive Computer Experience." This line is enough to pique the interest of an interviewer. However, be sure you're ready with a short, concise explanation of your experience. Include any programming languages you know and the different types of equipment you've used.

One final benefit of industrial employment is training. Industry knows that most industrial equipment is a long way from the computer mainstream. So they look for people who have a basic understanding and the willingness to learn. When they find this kind of person, they'll spare no expense for training.

◆ ◆ ◆

Two great tools.

NEW! SUPPORTS TURBO PASCAL 4.0, 5.0 & 5.5 AND QUICKPASCAL!

SAYWHAT?!
The lightning-fast screen generator.

Whether you're a novice programmer longing for simplicity, or a seasoned pro searching for higher productivity, you owe it to yourself to check out Saywhat. You see, with Saywhat, you can build beautiful, elaborate, color-coded screens in minutes! That's right. Truly *fantastic* screens for menus, data entry, data display, and help-panels that can be displayed with as little as one line of code in *any* language.

Here's what you get:

- Design screens, windows, and *moving bar menus!*
- Easy-to-use, powerful editor lets you create screens in a jiffy.
- Pop up your screens and menus with one line of code in dBASE, all the dBASE compilers, your favorite BASIC, Pascal, *or any other language!*
- Screen Library Manager.
- Generates runtime code.
- No runtime license or royalty fees.
- Comes with a 100 page manual, plus dozens of sample programs and free utilities.

$49.95

TOPAZ
The breakthrough DBMS toolkit for Pascal

If you'd like to combine the raw power and speed of Turbo Pascal or Microsoft's QuickPascal with the simplicity and elegance of dBASE, Topaz is just what you're looking for. That's because Topaz was specially created to let you enjoy the best of *both* worlds. The result? You create complete, truly dazzling applications in a very short time. And no wonder. Topaz is a comprehensive toolkit of dBASE-like commands and functions, designed to help you produce outstanding, polished programs, fast. Check out these powerful features:

- Over 200 routines all with easy-to-use, dBASE-like syntax.
- Data entry routines like SAY, GET, PICTURE, RANGE, color selection, unlimited data validation.
- Open up to 10 DBF files, with up to 7 indexes with USE, SELECT, SKIP, APPEND, PACK, INDEX ON, SET INDEX TO, and FIND.
- No need to buy dBASE. CREATE, BROWSE and REPORT utilities included.
- Easily implement Saywhat and Lotus-style moving bar menus.
- BROWSE any DBF file with just one line of code! Programmable and windowed too.
- Pick from windowed data or filenames with one line of code.
- Comprehensive Time & Date math in 7 international formats.
- Powerful code and report generators included!
- Comes with a complete 250 page manual, plus sample programs to get you started.

$74.95

ORDER NOW.
YOU RISK NOTHING.

Thousands of satisfied customers have already ordered from us. Why not visit your dealer or call toll-free, right now and put Saywhat and Topaz to the test yourself? They're fully guaranteed. You don't risk a penny.

Special limited-time offer! Save $26. Buy Saywhat?! and Topaz together for just $99 (plus $5 shipping and handling).

Visit your nearest dealer
or call toll-free:
800-468-9273
In California: 800-231-7849
International: 415-571-5019

Software Science, Inc.
The Research Group
100 Valley Drive, Brisbane, CA 94005

MONEY BACK GUARANTEE.
If you aren't completely delighted with Saywhat or Topaz, for any reason, return them within 30 days for a prompt, friendly refund.

Dealers: SAYWHAT?! and TOPAZ are available from Kenfil Distribution, and in Europe from ComFood Software, W. Germany 49-2534-7093

Guaranteed!

SOFTWARE SCIENCE INC.

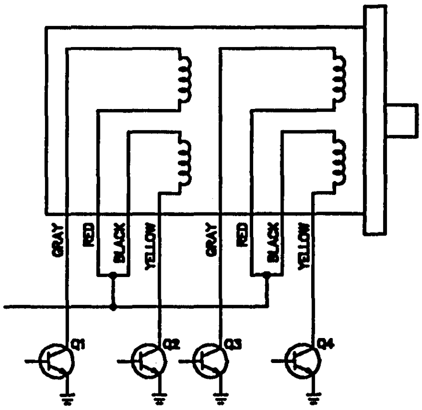
Reader Service Number 129

Driving Stepper Motors

Do It Yourself Stepper Motor Controller

If you're interested in controlling something mechanical, then you're in the right place. Steppers give you a great way to translate the electrical into the physical, and this is a quick (and clean) way to do it.

Figure 1—Unipolar Winding Stepper Motor Sequencing.



NORMAL 4 STEP SEQUENCE — CW ROTATION ↓ / CCW ROTATION ↑

Step	Q1	Q2	Q3	Q4
1	ON	OFF	ON	OFF
2	ON	OFF	OFF	ON
3	OFF	ON	OFF	ON
4	OFF	ON	ON	OFF
1	ON	OFF	ON	OFF

1/2 STEP 8 STEP SEQUENCE — CW ROTATION ↓ / CCW ROTATION ↑

1	ON	OFF	ON	OFF
2	ON	OFF	OFF	OFF
3	ON	OFF	OFF	ON
4	OFF	OFF	OFF	ON
5	OFF	ON	OFF	ON
6	OFF	ON	OFF	OFF
7	OFF	ON	ON	OFF
8	OFF	OFF	ON	OFF
1	ON	OFF	ON	OFF

WAVE DRIVE 4 STEP SEQUENCE — CW ROTATION ↓ / CCW ROTATION ↑

1	ON	OFF	OFF	OFF
2	OFF	OFF	OFF	ON
3	OFF	ON	OFF	OFF
4	OFF	OFF	ON	OFF
1	ON	OFF	OFF	OFF

One day, my artist friend John called and said, "I'm building a computerized wood embossing machine run by my VIC-20."

"That sounds neat," I replied, somewhat bewildered.

"It's going to use stepper motors to X-Y scan a pen and ink drawing optically. Then fire a solenoid to punch holes in the wood when it sees a line."

Now things were getting interesting. I had only the vaguest idea what a stepper motor was, so I said, "Steppers, eh?"

"Yeah. They're great; I can get them from Jerryco for $5 a piece. Can you design me some driver circuits? The ICs for powering these things would cost $20 a shot if I could find them."

Twenty bucks! For some silly reason, I still think ICs should cost less than the socket to plug them into. The next day John mailed me the excellent Airpax stepper application manual. So there I was, a new project.

Another Project Starts!

I had four design goals: (1) low cost; (2) directional control; (3) some degree of direct speed control by the computer; and (4) the use of as few lines from the parallel port as possible.

After a little experimentation, I found I could do the job with cheap, easily available CMOS ICs, though the design is a bit more complex than with dedicated driver chips.

But you save a lot! The driver uses three lines directly from your parallel port. Lines one and two control the speed

By David Metz

Metz Electronics
725 Climer
Muscatine, IA 52761

of the stepper. Line three controls the direction of rotation. The logic runs off the computer's 5-volt supply. I recommend you run the stepper off a separate supply. (Don't torture your computer any more than you have to.)

John used 12-volt steppers on his machine. With the transistors specified, he could have run 5 to 24-volt motors with no problems.

How Do Steppers Work?

A stepper is a permanent magnet motor with two sets of stator (nonmoving) windings. The two windings are mechanically offset by 1/2 a pole pitch—a pole being either the north or south end of one of the rotor's permanent magnets.

Each time a winding energizes, interaction between the winding and the magnet causes movement. The opposite poles attract, and the like poles repel. By controlling the timing (phase) and direction (polarity) of the electricity flowing to the coils, you control both speed and direction of rotation.

Figure 1 shows a unipolar (center tapped) winding motor and the order its windings must be driven. John and I chose the center tapped motors because they required only four driver transistors. Bipolar steppers require eight driver transistors; not good when you're trying to keep costs down.

> John and I chose the center tapped motors because they required only four driver transistors...

Steppers are available in a range of 0.72° to 90° rotation per step. The most common are 7.5° to 18° per step. A 7.5° stepper requires 48 steps per revolution.

The Break Down

The driver consists of four sections:

Clock: Supplies the pulses that determine the speed of the motor.

Phase shifter: Produces the 90° phase shift between the drive coils.

Reversing circuit: Produces a selectable 180° phase reversal on one drive coil to set direction of rotation.

Motor coil drivers: Transistors which take their control from CMOS outputs and in turn control high-current motor coils.

Speed Control

I offer two methods of supplying the clock pulses to the phase shifter. One, a fixed frequency clock that uses 1/2 of the CD4001B NOR gate that you already have (see Figure 2). The other uses the ever popular 555 timer (see Figure 3).

In either case, I need a clock that runs between 10 and 70 Hertz. From that you can infer that Mr. Woodpecker didn't move too fast at first. Later he got much faster, but that's another story.

Your application and the capabilities of the stepper motor you're driving will

Figure 2—CD4001B Clock.

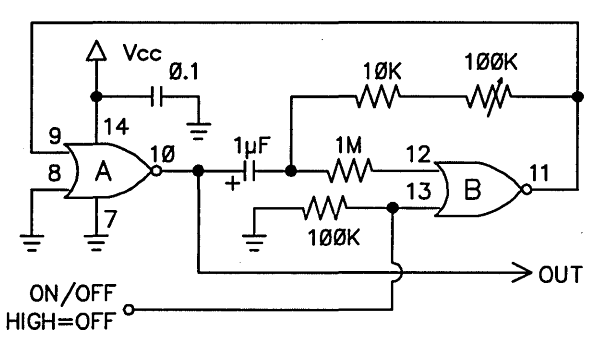
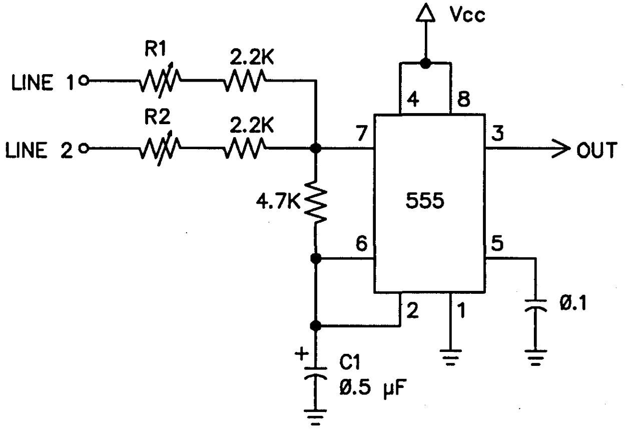


Figure 3—555 Clock.



determine the frequency range you need. Some can move faster than others. The advantage of the 4001 clock is simplicity. The disadvantage is that an on board potentiometer sets the speed. The 555 allows a simple means of software speed control at the cost of a few more parts.

Why speed control? Your robotics project might appear to work just fine at one speed. Ours did until John speeded it up. It turned out that the Woodpecker's mechanical parts did not like being slammed around by sudden starts and stops.

By incorporating one additional bit, I could select three different speeds. Thus the Woodpecker could accelerate or stop smoothly.

The RC time constant of R1, R2, and C1 determines the 555's frequency of oscillation. When C1 reaches ⅔ of V_{cc}, the capacitor discharges and another cycle begins.

Since this fixed internal voltage ratio determines the frequency of oscillation, changes in supply voltage will not affect the clock frequency. Once you determine the values for R1, R2, and C1, the supply voltage can be anything between 5 and 15 volts.

The 555 circuit's speed (frequency) control works by simple parallel resistance. With lines one and two low, no charge current flows to C1 and no oscillation occurs. The stepper sits idle, the current passing through its coils acting as a brake. (Yes, I know what you're thinking, I'll handle that problem later.)

We now make the time-honored principle of parallel resistance work for us. R1 gives one charge rate. R2 gives one that's faster. Thus, if line two is high, the motor will revolve faster than if line one (R1) is high. If both are high, the charge rate is much faster since the combined resistance of R1 and R2 is lower.

For example:

```
R1 (Line one) = 68,000 ohms
R2 (Line two) = 33,000 ohms
Line one high = 34 Hz
Line two high = 62 Hz
Both high     = 84 Hz
```

I would use small ten-turn (screw driver) adjustable pots for R1 and R2. This makes the inevitable after-construction fine-tuning much easier.

The 555s have one other peculiarity that bears watching. When the output pin 3 goes from high to low, the totem pole can short momentarily. This can send a nasty glitch back through the supply line.

There are two ways around this problem. First, try the 7555 CMOS version of the 555 timer (Intersil makes them, Digi-Key sells them, 90¢ each). Or use a large bypass capacitor. I added a 220 µF cap across the power supply pins of the 555. A value that large might be overkill.

Note: if you have several 555s on a board, normally you need only one big bypass.

Watch Your Parts Quality!

There are some components that you just can't goof around with. Timing resistors and capacitors are very important.

The leakage of the capacitor determines the upper value of C. Too large/too leaky capacitors just never seem to charge up! Save yourself some grief. Use a good quality capacitor for C1 like a Panasonic type NHE or HFS if you need a value above 0.22 µF. Below 0.22 µF, use high quality mylar or polystyrene.

Be sure to bypass V_{cc} with good mylar or disk ceramic capacitors in several locations. And keep the leads short! The high current square waves from the stepper drivers could induce spikes in the V_{cc} supply bus.

The Phase Shifter

The phase shifter uses a single CD4013B dual D type flip flop to produce the 90° shift. Note that it divides the clock frequency by four. The input to the D line tells the flip flop what to do. If D is high, clocking makes Q high and *not* Q low.

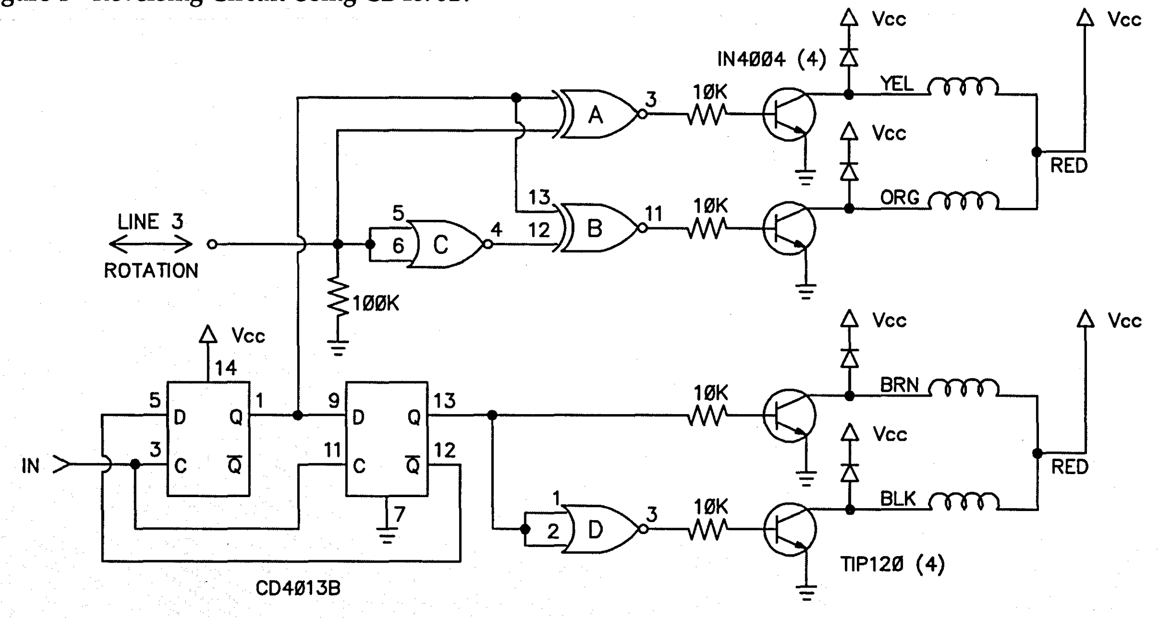
Grounding D does the reverse. Note how the two flip flops are cross connected.

Reversing Circuit

The reverser consists of ½ of a quad CD4070B exclusive NOR gate used as a controllable inverter (see Figure 4).

When line three is high, the gate inverts. When three is low, the gate does not invert. The extra inverter in front of the second section makes it produce the complement of what the first section is doing.

Figure 4—Reversing Circuit Using CD4070B.



Driver Transistors

The 10,000 ohm resistor in the base circuit limits base current to something safe. This value is not critical. The transistor is the Texas Instruments TIP-120 darlington.

A darlington is two transistors in one case with one driving the other. This allows the device to switch large currents despite very low drive currents.

The TIP-120 can switch voltages of up to 60 volts and currents of up to 5 amps. Since in this application they are either on or off, the only heat dissipation comes from their own internal resistance. The 12 volt Airpax steppers we used have 65 ohm coils. These draw only 220 milliamps from a 13.8 volt supply. The TIP-120 transistors don't even get warm.

At less than a dollar a transistor, I saw little point in using smaller or cheaper transistors. If you have the room, use the TIP-120s or equivalent for the reliability an oversized component gives you.

Now, about those diodes across the coils. These are called snubber diodes (or at least, that's what I've always called them). They clip the reverse voltage spike generated when the magnetic field collapses in the stepper coil. Leave these out only at your computer's peril.

Windings

The color code shown here is for the Airpax steppers John used. Others may have different connections. Remember to look for steppers with six leads. They're the bipolar ones that are easy to drive. A few minutes tracing with an ohm meter will give you the connections. Just remember that the center tap has half the resistance of the entire winding. Making a little drawing with the leads' colors and resistances will make the job easy.

Suppliers

Many of the suppliers that advertise in *Micro Cornucopia* have steppers. The ones I've listed may be new to you. I've used them all on a regular basis, and the service has been excellent.

Digi-Key has an excellent selection of new components with very fast service. They have the best selection of computer type components of any mail order firm I know. They send out an updated catalog every two months.

Jerryco often has steppers and some of the darnedest oddball stuff you've ever seen in your entire life. Don't laugh when you check out their catalog. You never know what weird items you'll need to build your dream project.

Small Parts Unlimited is a robotics builder's dream. They have a fantastic collection of structural material and small mechanical components. A great source of gears, shafting, cams, and drive chain.

Questions:

Q. Can I generate the clock pulses in software so I could do away with the clock?

A. Absolutely! I didn't, simply to keep the size of the software down. You can do it with timing loops to synthesize the pulses. Or you could use a programmable divider to divide down the system clock.

Q. Can I shut off the current flowing through the coils when the clock is off to save power on my super battery-operated toy I just designed?

A. Yup! (See Figure 5.) We OR together lines one and two. If both are low, the CD4066B analog switch removes drive from the bases of the driver transistors. I left this out of the Woodpecker in order to get some dynamic braking.

There are two ways of doing this. One uses two diodes with a 100K pull down resistor to OR lines one and two together. A little better way would be to use two sections of a spare CD4001B to first NOR, then invert the NORed signal.

This keeps you from having to add another device (a CD4071B) to the parts list. Leftover gates always seem to have a way of getting used.

Q. Shouldn't I isolate my computer's port from the drivers to protect it?

A. If it lets you sleep nights, by all means do it. Opto-isolators such as the GE H11L2GE work perfectly (see Figure 6). The H11L2 needs a current limiting resistor for the LED inside (about 220 ohms for five volts). Its output comes from an on board schmitt trigger.

It not only isolates and cleans up your data, you can also use it as a level translator. The H11L2 can run from 3 to 16 volts. That means if you have 12 volt steppers, you can run your steppers and logic from the same supply. That would save drain on your computer and eliminate the regulator for the driver logic.

The H11L2 has an open collector transistor for an output. So you have to connect it to an inverter and use a 100K pullup resister. That arrangement will give a logic one output from the inverter with a logic one into the opto-isolator.

Suppliers

DIGI-KEY, Inc.
Box 677
Thief River Falls, MN 56701
(800) 344-4539

JERRYCO, Inc.
601 Linden Place
Evanston, IL 60202
(312) 475-8440

Small Parts Unlimited
6891 N.E. 3rd Ave.
P.O. Box 381966
Miami, FL 33238-1736
(305) 751-0856

AIRPAX, Inc.
A Div. of North America
Philips Controls Corp.
Cheshire Division
Cheshire Industrial Park
Cheshire, CT 06410
(203) 271-6000

Call Airpax for the name of your local distributor and ask for a copy of their stepper motor handbook.

◆ ◆ ◆

Figure 5—Motor Off Switching (No Braking).

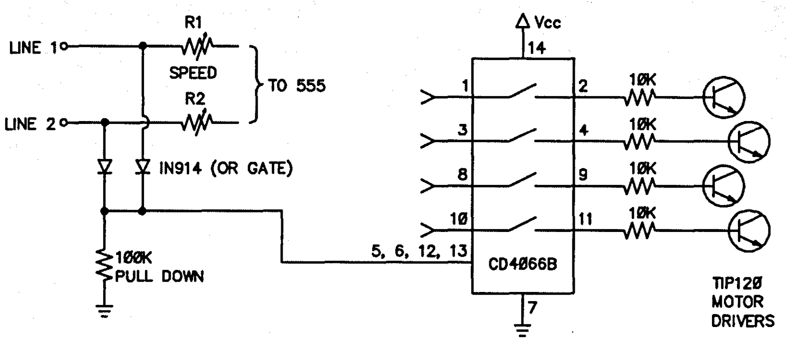
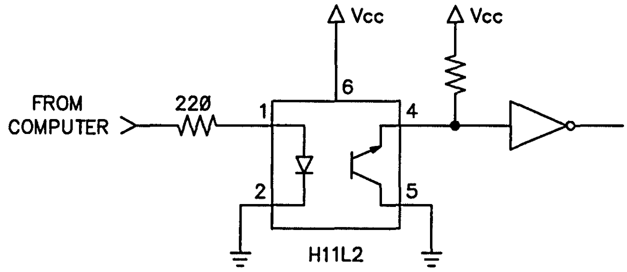


Figure 6—Stepper Motor Interface.



286 or 386SX?

CAN'T MAKE UP YOUR MIND?

WHAT SUITS YOUR NEEDS NOW MAY NOT BE RIGHT FOR YOU IN THE FUTURE. TAKE ADVANTAGE OF THE SPEED AND POWER OF A PT286 SYSTEM AND RETAIN THE OPTION OF UPGRADING LATER TO A 386SX WITH A PLUG IN CPU MODULE. ALL PRODUCTS ARE COMPATIBLE WITH DOS, UNIX, XENIX, OS/2, AND MAY CONTAIN UP TO 5 MEGABYTES OF ON-BOARD MEMORY.

286/386SX KITS ARE AVAILABLE

- PT386SX Assembled board, 16MHZ, 512K RAM, FDC, 2 serial, 1 parallel port **$795.00**
- PT286 Assembled board, 16MHZ, OK RAM, FDC, 2 serial, 1 parallel port **$475.00**
- PT386SX CPU Module Optional upgrade to PT286 Only **$325.00**

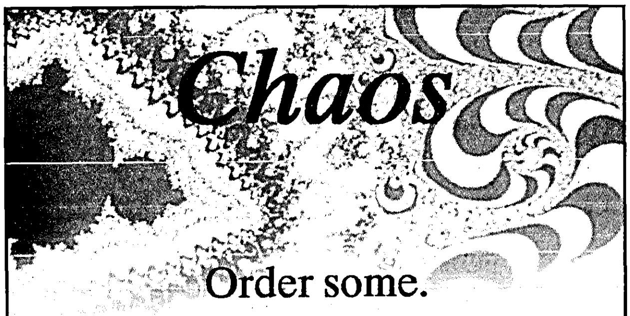
COMPLETE SYSTEMS STARTING AT $999
1 YEAR WARRANTY

CATALOG AVAILABLE UPON REQUEST

PERIPHERAL TECHNOLOGY
1710 CUMBERLAND PT. DR. SUITE 8,
MARIETTA, GEORGIA 30067
404/984-0742/FAX ORDER LINE: 404/984-8248

ALL TRADEMARKS ARE THE PROPERTY OF THEIR RESPECTIVE OWNERS.

Reader Service Number 119



Mandelbrot Explorer 3.0

Fantastic fractal grapnics on 16-color EGA/VGA to 800x600. Magnifies up to 16.5 trillion times. Stop and start at will, save and retrieve, collage, full control over color boundaries, "zoom box," display of periodic orbits, auto-backup, all optimizations for speed including pixel interpolation and 386 integer support. Comes with seven ready-made pictures for immediate gratification.

$30

Peter Garrison
1613 Altivo Way
Los Angeles, CA 90026
213 665 1397

When ordering please specify EGA/VGA and disk format
Overseas orders please add $4

Reader Service Number 112

By Edwin Thall

Wayne General and Technical College
Orrville, OH 44667

Writing TSR Programs

Dr. Edwin Thall, Professor of Chemistry at Wayne General and Technical College of The University of Akron, teaches chemistry and computer programming. (I understand that he has not been terminated, though he is definitely resident.)

IBM and Microsoft ushered in a new approach to writing programs when they introduced the terminate-but-stay-resident (TSR) function call with DOS version 1.1. Software developers quickly caught on, and now they make just about every conceivable utility into a resident program. Preassigned hot keys pop up such applications as calculators, calendars, address books, and notepads.

As the name suggests, resident programs remain in computer memory while other programs execute. But resident programs are also potential disasters since, without warning, they can corrupt data, scramble the screen, or crash the system.

Usually systems crash because DOS doesn't provide the controls needed to maintain peaceful coexistence between two or more TSRs. Users seem to be divided into two camps: those who swear by TSRs and those who swear at TSRs.

The intention of this article is neither to defend nor attack the widespread use of TSRs, but rather to explain thoroughly how they work. I'll present two examples, one well-behaved and the other a "bully." We'll also explore removing a resident program without rebooting the system.

Resident Versus Transient Memory

COMMAND.COM is divided into three components: the resident, initialization, and transient portions. The resident portion is loaded in lower memory immediately following IBMDOS.COM. Interrupt vectors 21-24H point to routines within the resident portion. This portion also contains the bulk of the error recovery messages. Once loaded, the resident portion remains in memory.

The initialization portion of COMMAND.COM loads immediately above the resident portion when the system boots. This portion initially takes control and displays the prompts for the date and time. It also runs the AUTOEXEC batch file, if one is present. The first program from disk overlays this section of memory.

The transient portion loads in the high end of COMMAND.COM and can overlay one interim program with another. As resident connotes permanency, transient implies temporary. The responsibilities of the transient portion include displaying the DOS prompt and reading, as well as executing, commands. The transient portion also contains the routines to load .COM and .EXE files into the appropriate memory location for execution.

When you request execution of a program, the transient portion constructs a program segment prefix directly above the resident portion of COMMAND.COM. It then loads the program immediately following the program segment prefix, sets the exit addresses, and transfers control to your program.

When execution is complete, COMMAND.COM regains control and assigns the same memory locations to your next application. However, if you terminate with one of the TSR functions, the program becomes an extension of the resident portion of COMMAND.COM.

The area of memory occupied by the program is reserved in the same manner as memory is reserved for DOS. Future applications will not overwrite this section. The only way to eliminate such a program, without the benefit of a resident memory manager utility, is to reboot.

Establishing Residency

Let's install a simple program directly above the resident portion of COMMAND.COM. After installation, we'll use the DEBUG utility to locate and examine the program. Let's begin by calling Interrupt 27H to incorporate the message "STAY RESIDENT" in the resident portion of memory.

Interrupt 27H terminates the currently executing program and reserves part or all of its memory so that the next transient program will not overlay it. The maximum quantity of memory this interrupt can reserve is 64K bytes. From DOS, load DEBUG and enter the following program (omit comments):

```
A>DEBUG
-A100
DS:0100  JMP 0110   ;BY-PASS DATA
DS:0102  DB 'STAY RESIDENT ';DATA
DS:0110  MOV DX,0110;PROT TO HERE
DS:0113  INT 27                 ;TSR
DS:0115  <ENTER>
```

The program takes up 21 bytes (offsets 0100-0114H) and can be viewed by typing:

```
-D100,114
DS:0100  EB 0E 53 54 41 59 20 52
        -45 53 49 44 45 4E 54 20
DS:0110  BA 10 01 CD 27
```

Later, you can search resident memory to locate this code. The 256-byte program segment prefix (PSP) precedes the code. The PSP occupies offsets 00-FFH and is made resident along with the code. You can display the PSP with:

```
-D0,FF
```

Here's how the program works. The first instruction (JMP 0100) bypasses the data (DB statement). The next instruction (MOV DX,0110) specifies that the program be protected up to, but not including, offset 0110H. When the program terminates with the last instruction (INT 27H), the PSP (offsets 0000-00FFH) and code (offsets 0100-010FH) become an extension of the resident portion of COMMAND.COM. Save the program as RES27H.COM:

```
-NRES27H.COM
-RCX
CX ????
:0015
-RBX
BX ????
:0000
-W
```

Since DEBUG cannot invoke Interrupt 27H, return to DOS and execute the program.

Reload DEBUG and use the "S" command to search the first 64K bytes of memory for the program's code. When searching the initial 64K bytes of computer memory, you must set the data segment (DS) to zero.

```
A>DEBUG
-RDS
DS ????
:0000
-S 0 L FFFF EB 0E 53 54 41 59 20 52
0000:61E0
0000:8F6F
```

Ignore the last address retieved by the search command. If you're using DOS 2.10, the program's code begins at 0:61E0H with the PSP 100H bytes below at 0:60E0H. To display the PSP and code, enter:

```
-D0:60E0,61EF
```

How is DOS able to keep track of resident memory allocations? The paragraph directly below the PSP (0:60D0H) is a memory control block. Paragraphs originate at an offset ending in zero and are 16 bytes in length. Let's explore the first five locations of this block:

```
-D60D0,60D4
0000:60D0   4D 0E 06 11 00
```

The first location of a memory control block, called the identifier byte, contains either 4DH or 5AH. The value 4DH indicates that the memory directly above belongs to a program or DOS, while 5AH verifies that no more memory control blocks succeed this one. The value 5AH also instructs DOS to install the next TSR here.

Bytes 2 and 3 of the memory control block hold the PSP segment (060EH) of the program to follow. If the PSP segment number is zero, then the memory defined by the memory control block is free. Bytes 4 and 5 specify the size (in paragraphs) of the pending memory block. Notice the value is 0011H, or 17 paragraphs.

Memory control blocks chain from one to the next, and DOS is notified that the next memory control block is 17 plus one, or 18, paragraphs above. Between these memory control blocks are the PSP (16 paragraphs) and the program's code (1 paragraph). Current DOS versions don't use the last 11 bytes in a memory control block so these might contain remnants of other programs.

> Users seem to be divided into two camps: those who swear by TSRs and those who swear at TSRs.

To predict the location of the next resident program, search above the PSP for the memory control block beginning with 5AH. The first occurrence here is at 0:6220H. Look at the initial five bytes of this paragraph:

```
-D6220,6224
0000:6220   5A 23 06 DD 79
```

The information tells us to expect the PSP of the next resident program to load at segment 0623H (address 0000:6230H).

Let's install the same program a second time in resident memory, but this time by invoking the other TSR function. DOS version 2.00 introduced function 31H. IBM prefers this function because it passes a return code and allows more than 64K bytes to remain resident.

When using DOS function 31H, the number of bytes made resident is declared (by paragraph) in the DX register. The value stored in the DX register (0011H) specifies that 17 paragraphs are to be protected. Before entering the program with the DEBUG "A" command, secure a new DS designation by returning to DOS and reloading DEBUG:

```
-Q
A>DEBUG
-A100
DS:0100  JMP  110       ;BY-PASS DATA
DS:0102  DB   'STAY RESIDENT ';DATA
DS:0110  MOV  AH,31              ;TSR
DS:0112  MOV  AL,01     ;RETURN CODE
DS:0114  MOV  DX,0011 ;SAVE 17 PGPH
DS:0117  INT  21            ;CALL DOS
DS:0119  <ENTER>
```

When this program terminates, 16 paragraphs of PSP and one paragraph of code will emerge as part of resident memory. Save the program as RES31H.COM:

```
-NRES31H.COM
-RCX
CX ????
:0019
-RBX
BX ????
:0000
-W
```

Return to DOS and execute RES31H.COM.

To determine the location of the second resident program, search with the Debug "S" command:

```
A>DEBUG
-RDS
DS ????
:0000
-S 0 L FFFF EB 0E 53 54 41 59 20 52
0000:61E0
0000:6330
0000:90BF
```

As anticipated, the new resident code begins at 0:6330H with the PSP at 0:6230H. The memory control block is located at the paragraph preceding the PSP (0:6220H). Display the initial five bytes of this block:

```
-D6220,6224
0000:6220   4D 23 06 11 00
```

The identifier byte (4DH) indicates that a PSP follows at segment 0623H. The length of the PSP/code is 0011H, or 17 paragraphs. Every time you execute a TSR function, the protected memory is "stacked" directly above the previous

Figure 1—First 128 Bytes of DOS 2.10 Interrupt Vector Table.

```
-D0:0
0000:0000  72 30 EB 00 47 01 70 00-C3 E2 00 F0 47 01 70 00
0000:0010  47 01 70 00 54 FF 00 F0-23 FF 00 F0 23 FF 00 F0
0000:0020  A5 FE 00 F0 87 E9 00 F0-23 FF 00 F0 23 FF 00 F0
0000:0030  23 FF 00 F0 55 07 00 C8-57 EF 00 F0 47 01 70 00
0000:0040  65 F0 00 F0 4D F8 00 F0-41 F8 00 F0 51 02 00 C8
0000:0050  39 E7 00 F0 59 F8 00 F0-2E E8 00 F0 D2 EF 00 F0
0000:0060  00 00 00 F6 92 01 00 C8-6E FE 00 F0 40 01 70 00
0000:0070  49 FF 00 F0 A4 F0 00 F0-22 05 00 00 00 00 00 00
```

◆ ◆ ◆

Figure 2—Shift Status Codes

Bit	Specifies	Value
0	right shift	1
1	left shift	2
2	Ctrl key	4
3	Alt key	8
4	Scroll Lock	16
5	Num Lock	32
6	Caps Lock	64
7	Insert state	128

◆ ◆ ◆

resident program. You can continue stacking resident programs until you've filled RAM.

By now, you have probably realized how easy it is to install a program or data in the resident portion of the computer's memory. The not-so-easy part involves activating it after it's resident. This takes us to the subject of interrupt-handlers.

Interrupt Handlers

Interrupts tell the computer's central processing unit to suspend what it's doing and transfer to an interrupt handler program. The handler quickly takes the appropriate action and then returns control to the suspended program. The 8086/8088/80286 family supports up to 256 interrupts, classified as internal hardware, external hardware, or software interrupts.

Certain events encountered during program operation, such as division by zero, generate internal hardware interrupts. Peripherals, such as the keyboard, generate external hardware interrupts. Any program can request a software interrupt.

The initial 1,024 bytes of computer memory are known as the interrupt vector table. Each table entry consists of four bytes pointing to the address of its handler. Figure 1 provides the first 128 bytes of this table for DOS version 2.10.

Each four-byte position in the table corresponds to one interrupt type (00-FFH) and contains the segment and offset of its interrupt handler. To find a particular interrupt in the table, multiply the interrupt type by four. For example, INT 9H is stored in offsets 36-39 (0024-0027H) and points to address F000:E987H in ROM.

Our main programming interest in interrupt vectors is not to read them, but to change them so they point to a new interrupt handling routine.

Writing interrupt handlers has the reputation of being difficult and best left to experts. Actually, the procedure is straightforward. When an interrupt is invoked, the CPU pushes the flag register, the segment register (CS), and the instruction pointer (IP) onto the stack and disables interrupts. It then uses the interrupt number to fetch the address of the handler from the vector table and begins execution at that address.

The handler will enable interrupts, save registers, and process the interrupt.

You can easily activate a resident program by modifying an interrupt vector. Two interrupts especially qualified for this task are the keyboard interrupt (9H) and the user timer interrupt (1CH).

Before you attempt to take control of these interrupts, there are a few things you should know. The handler can call DOS, and DOS will perform the function, but the handler cannot be reentered from DOS. The last instruction of a handler must be IRET (IP, CS, and flag registers popped). When an interrupt is invoked, the trap flag is suspended so that you cannot trace through a handler to locate errors.

Each time you press or release a key, the action reports to the ROM-BIOS via INT 9H. The ROM routine reads port 60H to find out the selected keystroke. The scan code and ASCII value are stored until you press the next key.

When you take control of INT 9H, every keystroke gets routed through your program before being passed onto the ROM. Most programmers select special keystrokes to activate their resident programs.

The special control keys are ideally suited for this role because they do not produce characters of their own, but change the codes generated by other keys. The keyboard I/O interrupt (INT 16H, function 02) returns the status of the eight keys listed in Figure 2.

For example, if you press both shift keys, the AL register returns the value 3. When you press all eight keys simultaneously, it returns the value 255. Testing for a definite value can activate a resident program. Otherwise, keystrokes are treated in the usual manner.

The user timer interrupt is taken 18.2 times per second and is invoked by the timer interrupt (INT 8H). The vector for Interrupt 1CH points to address F000:FF49H in ROM. Use the DEBUG "U" command to look at the first instruction of this handler:

```
-UF000:FF49,FF49
FOOO:FF49   CF    IRET
```

This is a dummy handler which does nothing but execute an interrupt return. However, control of this interrupt allows you to continuously run a procedure in resident memory.

TSR programs typically include an initialization procedure, a portion to redefine the interrupt vector table, and the code that remains resident.

Normally, a program will want to leave only part of itself resident, discarding the initialization code. Therefore, you should organize a TSR so that the resident portion comes at the beginning of the program.

To demonstrate how resident programs work, I've introduced POPUP and RENEGADE. These programs store the current screen in resident memory, but RENEGADE exerts absolute control over the keyboard interrupt. Once the keyboard is in its grasp, RENEGADE does not relinquish control and excludes all other resident programs from using this interrupt.

Introducing POPUP

Figure 3 lists the assembly language source code for POPUP. After its installation in resident memory, activate the program by hitting both shift keys simultaneously. The first entry stores the current screen in resident memory, while subsequent activations pop up the stored screen. This program assembles with MASM.

Here's how POPUP works. The first instruction in the code segment (JMP INIT) bypasses the MAIN procedure and skips to the INIT procedure. The INIT procedure saves the old keyboard vector and then chains INT 9H to the MAIN procedure.

The last instruction of the INIT procedure (INT 27H) makes the ultimate sacrifice by allowing itself to be erased. It terminates the entire operation but protects code from the start of the PSP to the end of the MAIN procedure. Note how the DX register points to the start of INIT (the offset one byte beyond the protected code).

Once installed, the MAIN procedure becomes our interrupt handler. Every keystroke gets intercepted by this handler and redirected to the old INT 9H in ROM for processing. Our handler then calls INT 16H to determine the latest keystroke(s).

If you press both shift keys simultaneously, the processor branches to the routine that stores the current screen. Then, escape returns you to the current program. The screen you just stored will now pop up whenever you press both shift keys. When you finish viewing the stored screen, escape restores the current screen.

To store a new screen, press the Alt and right shift keys. Again, the escape key returns control to the parent program.

POPUP saves two screens: the current application screen you'll return to and the stored screen. Storing a complete screen demands a significant block of computer memory. The storing of a color graphic screen consumes 4000 bytes (as configured, POPUP works with CGA systems), whereas the entire POPUP program requires about 8200 bytes. As you can see, the two stored screens account for 98% of the memory allocated to POPUP.

POPUP is an example of a well-behaved resident program; it does not attempt to sabotage other programs. But what if you install a second resident program, and it also seeks control of the keyboard interrupt? The last program loaded will intercept the keystrokes first, and then chain them to the previously-loaded TSR. A problem would arise if both programs attempted to use the same hot keys.

Figure 3—POPUP.ASM

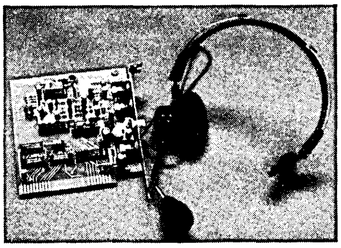
```
;Resident program to save screen.

IVT     SEGMENT AT 0H         ;INTERRUPT TABLE SEGMENT
        ORG 9H*4
KEYBD   DW 2 DUP (?)          ;INT 9H VECTOR
IVT     ENDS
;----------------------------------
CODE    SEGMENT PARA PUBLIC 'CODE'  ;CODE SEGMENT
        ASSUME CS:CODE
        ORG 100H
BEGIN:  JMP  INIT             ;GOTO INITIALIZATION ROUTINE
OLDKB   LABEL DWORD           ;INT 9H ADDRESS IN ROM
OLDKEY  DW 2 DUP (?)          ;STORE OLD INT 9H VECTOR
SCREEN  DB  4000 DUP ('S')    ;STORE ORIGINAL SCREEN
SHIFT   DB  0                 ;SAVE SHIFT STATUS CODE
STATUS  DB  0                 ;CHECK NEW SCREEN STATUS
POPUP   DB  4000 DUP ('P')    ;POP-UP SCREEN
;----------------------------------
;MAIN is made resident and every keystroke routed here.
;----------------------------------
MAIN    PROC  NEAR            ;NEW INT 9H VECTOR POINTS HERE
        STI                   ;ENABLE INTERRUPTS
        PUSH  AX              ;SAVE REGISTERS
        PUSH  BX
        PUSH  CX
        PUSH  DX
        PUSH  SI
        PUSH  DI
        PUSH  DS
        PUSH  ES
        PUSHF                 ;SIMULATE INTERRUPT RETURN
        CALL  OLDKB           ;OLD KEYBD ROUTINE IN ROM
        MOV   AH,2            ;RETURN KEYBD FLAGS
        INT   16H
        MOV   SHIFT,AL        ;SAVE SHIFT STATUS
        AND   AL,3
        CMP   AL,3            ;BOTH SHIFT KEYS PRESSED?
        JE    SAVE            ;IF YES, THEN SAVE SCREEN
        MOV   AL,SHIFT        ;RESTORE SHIFT STATUS
        AND   AL,9            ;ALT/RT. SHIFT KEYS PRESSED?
        CMP   AL,9
        JNE   EXIT            ;IF NO, THEN EXIT HANDLER
        MOV   STATUS,0        ;EXPECT NEW SCREEN
        JMP   SAVE            ;GET NEW SCREEN

EXIT:   POP   ES              ;RESTORE REGISTERS AND EXIT
        POP   DS
        POP   DI
        POP   SI
        POP   DX
        POP   CX
        POP   BX
        POP   AX
        IRET                  ;RETURN TO PARENT PROGRAM

;----------------------------------
;This routine activated when both shift keys
;or Alt/Rt shift pressed.
;----------------------------------
;Save the original cursor position
SAVE:   MOV   AH,3            ;READ CURSOR POSITION
        MOV   BH,0            ;SELECT PAGE 0
        INT   10H
        PUSH  DX              ;SAVE CURSOR POSITION
        PUSH  CX              ;SAVE CURSOR SIZE

;Save the original screen
        MOV     AX,0B800H     ;COLOR GRAPHICS MEMORY
        MOV     DS,AX         ;SOURCE SEGMENT
        MOV     SI,0          ;SOURCE OFFSET
        PUSH    CS
        POP     ES            ;DEST. SEG
        MOV     DI,OFFSET SCREEN  ;DEST. OFFSET
        CLD                   ;CLEAR DIRECTIONAL FLAG
        MOV     CX,4000       ;MOVE 4000 BYTES
        REP     MOVSB         ;FROM VIDEO TO MEMORY

;Determine if new screen
        CMP     STATUS,0FFH   ;CHECK NEW SCREEN STATUS
        JE      READY         ;IF YES, THEN GOTO READY
```

VOICE MASTER KEY®
VOICE RECOGNITION SYSTEM
FOR PC/COMPATIBLES & TANDY 1000 SERIES
A FULL FEATURED VOICE I/O SYSTEM

GIVE A NEW DIMENSION TO PERSONAL COMPUTING. . .The amazing **Voice Master Key System** adds voice recognition to just about any program or application. Voice command up to 256 keyboard macros from within CAD, desktop publishing, word processing, spread sheet, or game programs. Fully TSR and occupies less than 64K. Instant response time and high recognition accuracy. Voice recognition tool-box utilities are included. **A genuine productivity enhancer!**

SPEECH RECORDING SOFTWARE. . .Digitally record your own speech, sound, or music to put into your own software programs. Software provides sampling rate variations, graphics-based editing, and data compression utilities. Create software sound files you can add to macros for voice recognition verification response. **A complete, superior speech and sound development tool.**

SOFTWARE CONVERSION CODES. . .The **Voice Master Key System** operates a growing list of third party talking software titles using synthesized phonetics (text-to-speech) or digitized PCM, ADPCM, and CVSDM encoded sound files. **Voice Master Key System does it all!**



EVERYTHING INCLUDED. . .**Voice Master Key System** consists of a plug-in card, durable lightweight microphone headset, software, and manual. Card fits any available slot. External ports consist of mic inputs and volume controlled output sockets. **High quality throughout, easy and fun to use.**

ONLY $149.95 COMPLETE

ONLY $89.95 FOR TANDY 1000 SL/TL MODELS—SOFTWARE PACKAGE ONLY.
Requires Tandy Brand Electret microphone.

ORDER HOTLINE: (503) 342-1271
Monday-Friday, 8AM to 5PM Pacific Time

Visa/MasterCard, company checks, money orders, CODs (with prior approval) accepted. Personal checks subject to 3 week shipping delay. Specify computer type and disk format (3½" or 5¼") when ordering. Add $5 shipping charge for delivery in USA and Canada. Foreign inquiries contact Covox for C & F quotes. 30 DAY MONEY BACK GUARANTEE IF NOT COMPLETELY SATISFIED. ONE YEAR WARRANTY ON HARDWARE.

CALL OR WRITE FOR **FREE** PRODUCT CATALOG

COVOX INC. 675-D Conger St.
Eugene, Oregon 97402 U.S.A.
TEL: 503-342-1271 • FAX: 503-342-1283
Reader Service Number 143

Introducing RENEGADE

Not all resident programs are as well-mannered as POPUP. Some have been labelled thugs, bullies, and outlaws. What have these programs done to warrant such a reputation? When some commercial programs replace the table address of INT 9H, they exclude all others and do not allow concurrent use of a resident program like POPUP. Let's take POPUP and turn it into a bully.

Figure 4 lists the assembly language source for RENEGADE. The MAIN procedure of this program, not presented in its entirety in the figure, is identical to POPUP. RENEGADE performs the same operations as POPUP but it takes over both INT 1CH and INT 9H.

As mentioned previously, INT 1CH occurs 18.2 times per second and does nothing except return from ROM. You can use this interrupt to monitor the vector table continuously.

The initialization portion of RENEGADE takes control of INT 1CH and directs it to the CHECK procedure in resident memory. You need not save the old INT 1CH vector since it merely executes an interrupt return.

The CHECK procedure, invoked 18.2 times a second, determines whether the keyboard interrupt is chaining directly to the MAIN procedure in resident memory. If another program has taken over the INT 9H vector, the CHECK procedure recaptures it.

RENEGADE is an example of a selfish resident program. It doesn't permit chaining from one resident program to another. The consequence is that it doesn't allow other resident programs to run. Its lack of compatibility diminishes the worth of RENEGADE. This program works only with computers storing their keyboard interrupt routines at address F000:E987H.

To observe RENEGADE in action, load the program and display the vector for INT 9H. Before executing, reboot the system to purge any previously installed resident programs.

```
A>RENEGADE
A>DEBUG
-D0:24,27
0000:0024  4D 20 0E 06
```

The vector points to the MAIN procedure at 060E:204DH. This is the address if RENEGADE is the first TSR loaded and you are using the DOS 2.10 version. Now install POPUP and display the INT 9H vector again:

Continued from page 37

```
;Move the first pop-up screen
        PUSH    CS
        POP     DS              ;SOURCE SEG
        PUSH    CS
        POP     ES              ;DEST. SEG
        MOV     SI,OFFSET SCREEN  ;SOURCE OFFSET
        MOV     DI,OFFSET POPUP   ;DEST. OFFSET
        MOV     CX,4000         ;MOVE 4000 BYTES
        REP     MOVSB           ;FROM SCREEN TO POPUP
        MOV     STATUS,0FFH     ;CHANGE STATUS AFTER FIRST RUN
        JMP     POS

;Save new pop-up screen
READY:  MOV     DI,0            ;DEST. OFFSET
        MOV     SI,OFFSET POPUP   ;SOURCE OFFSET
        PUSH    CS
        POP     DS              ;SOURCE SEG
        MOV     AX,0B800H
        MOV     ES,AX           ;DEST. SEG
        CLD
        MOV     CX,4000         ;MOVE 4000 BYTES
        REP     MOVSB           ;FROM MEMORY TO VIDEO

;Turn cursor off
POS:    MOV   AH,2              ;POSITION CURSOR
        MOV   BH,0
        MOV   DL,0              ;FIRST COLUMN
        MOV   DH,25             ;ROW OFF VIDEO DISPLAY
        INT   10H

;Wait for escape keystroke
WAIT:   MOV   AH,0              ;WAIT FOR KEYSTROKE
        INT   16H
        CMP   AL,27             ;ESCAPE KEY?
        JNZ   WAIT

;Restore original cursor
        POP     CX              ;ORIGINAL CURSOR SIZE
        POP     DX              ;ORIGINAL CURSOR POSITION
        MOV     BH,0
        MOV     AH,2            ;SET CURSOR
        INT     10H

;Restore original screen
        MOV     DI,0            ;DEST. OFFSET
        MOV     SI,OFFSET SCREEN  ;SOURCE OFFSET
        PUSH    CS
        POP     DS              ;SOURCE SEGMENT
        MOV     AX,0B800H
        MOV     ES,AX           ;DEST. SEGMENT
        CLD
        MOV     CX,4000         ;MOVE 4000 BYTES
        REP     MOVSB           ;FROM MEMORY TO VIDEO
        JMP     EXIT            ;HANDLER IS DONE
MAIN    ENDP

;------------------------------------
;INIT is executed once and is not made resident.
;------------------------------------
INIT    PROC NEAR
        MOV   AX,IVT            ;SET DS TO INTERRUPT VECTOR
TABLE
        MOV   DS,AX
        ASSUME  DS:IVT
        MOV   AX,KEYBD          ;SAVE OLD INT 9H VECTOR
        MOV   OLDKEY,AX
        MOV   AX,KEYBD[2]
        MOV   OLDKEY[2],AX
        CLI                     ;DISABLE INTERRUPTS
        MOV   KEYBD,OFFSET MAIN  ;INSTALL NEW VECTOR
        MOV   KEYBD[2],CS
        STI                     ;ENABLE INTERRUPTS
        MOV   DX,OFFSET INIT    ;SAVE TO END OF MAIN PROCEDURE
        INT   27H               ;TSR
INIT    ENDP
;------------------------------------
CODE    ENDS
        END   BEGIN
```

◆ ◆ ◆

Figure 4—RENEGADE.ASM

```
;Same operation as POPUP.ASM but takes over INT 1CH

IVT     SEGMENT AT 0H               ;INTERRUPT TABLE SEGMENT
        ORG 9H*4
KEYBD   DW 2 DUP (?)                ;INT 9H VECTOR
        ORG 1CH*4
TIMER   DW  2 DUP (?)               ;INT 1CH VECTOR
IVT     ENDS
;-------------------------------
CODE    SEGMENT PARA PUBLIC 'CODE'    ;CODE SEGMENT
        ASSUME CS:CODE
        ORG 100H
BEGIN:  JMP  INIT                   ;GOTO INITIALIZATION ROUTINE
OLDKB   LABEL DWORD                 ;INT 9H ADDRESS IN ROM
        DB    87H,0E9H,00H,0F0H     ;STORE INT 9H VECTOR
NEWKEY  DW 2 DUP (?)                ;CURRENT INT 9H VECTOR
SCREEN  DB  4000 DUP ('S')          ;STORE ORIGINAL SCREEN
SHIFT   DB  0                       ;SAVE SHIFT STATUS CODE
STATUS  DB  0                       ;CHECK NEW SCREEN STATUS
POPUP   DB  4000 DUP ('P')          ;POP-UP SCREEN
;-------------------------------
;MAIN is made resident and every keystroke routed here.
;-------------------------------
MAIN    PROC  NEAR                ;NEW INT 9H VECTOR POINTS HERE
;*****   IDENTICAL TO MAIN PROC IN POPUP.ASM   *****
MAIN    ENDP
;-------------------------------
;Once installed, CHECK is invoked 18.2 times/sec.
;INT 1CH vector point here.
;-------------------------------
CHECK   PROC    NEAR
        STI                         ;ENABLE INTERRUPTS
        PUSH    AX                  ;SAVE REGISTERS
        PUSH    BX
        PUSH    CX
        PUSH    DX
        PUSH    SI
        PUSH    DI
        PUSH    DS
        PUSH    ES
        PUSH    CS
        POP     DS                  ;SOURCE SEGMENT
;Determine if INT 9H vector was changed
        MOV     SI,OFFSET NEWKEY ;SOURCE OFFSET
        MOV     AX,0
        MOV     ES,AX               ;DEST. SEGMENT
        MOV     DI,9H*4             ;DEST. OFFSET
        MOV     CX,2                ;COMPARE 2 WORDS
        REPE    CMPSW
        JE      SKIP                ;IF MATCH, EXIT
;Reinstate INT 9H vector
        MOV     AX,IVT
        MOV     DS,AX
        ASSUME  DS:IVT
        CLI                         ;DISABLE INTERRUPTS
        MOV     KEYBD,OFFSET MAIN ;REINSTATE KEYBD VECTOR
        MOV     KEYBD[2],CS
        STI                         ;ENABLE INTERRUPTS
SKIP:   JMP     EXIT
CHECK   ENDP
;-------------------------------
;INIT is executed once and is not made resident.
;-------------------------------
INIT    PROC NEAR
        MOV  AX,IVT             ;SET DS TO INTERRUPT VECTOR TABLE
        MOV  DS,AX
        ASSUME  DS:IVT
        CLI                         ;DISABLE INTERRUPTS
        MOV  KEYBD,OFFSET MAIN      ;INSTALL NEW VECTOR
        MOV  KEYBD[2],CS
        MOV  TIMER,OFFSET CHECK     ;INSTALL NEW TIMER VECTOR
        MOV  TIMER[2],CS
        STI                         ;ENABLE INTERRUPTS
        MOV  DX,OFFSET INIT    ;SAVE TO END OF CHECK PROCEDURE
        INT  27H                    ;TSR
INIT    ENDP
;-------------------------------
CODE    ENDS
        END  BEGIN
```

◆ ◆ ◆

```
-Q
A>POPUP
A>DEBUG
-D0:24,27
0000:0024  4D 20 0E 06
```

Interrupt 9H should still point to 060E:204DH. Once loaded, RENEGADE assures that its handler has first claim to all keystrokes. It not only recaptures the keyboard interrupt vector, but denies POPUP any opportunity for activation.

What happens if two resident programs attempt similar strategies to take over INT 1CH? The last program loaded is the one whose handler is invoked 18.2 times per second. In this game, the program loaded last has the advantage. Now you can appreciate why memory resident utilities such as SideKick, SuperKey, and Prokey insist they must be loaded last.

I tried loading SideKick, an aggressive user of five interrupts, followed by RENEGADE. The result was chaos, with the hot keys giving unpredictable and strange screens, even though, interestingly enough, the system didn't crash.

Evicting RENEGADE

Software packages are available which allow you to manipulate resident programs. These utilities, often referred to as managers or organizers, are designed to remove other programs from resident memory without rebooting the system. I've included a short program (EVICT.COM) to demonstrate how to eliminate RENEGADE's grip on the keyboard interrupt.

Resident programs are distinct from normal ones in two ways: they don't release memory blocks when terminated; and they chain one or more interrupt vectors to themselves. To remove RENEGADE from resident memory, you must restore the interrupt vectors as they existed prior to its installation, and then release all its memory blocks. DOS function 49H releases memory blocks.

The ES register specifies the segment to be released, while bytes 4 and 5 of the memory block already contain the length of the block. Display the following resident memory locations:

```
-D0:60A0,60EF
0000:60A0  4D 0E 06 02 00 05 C8 00-
           A3 BD 0B A1 02 00 8C 1E
0000:60B0  50 41 54 48 3D 00 43 4F-
           4D 53 50 45 43 3D 43 3A
0000:60C0  5C 43 4F 4D 4D 41 4E 44-
           2E 43 4F 4D 00 00 FF FF
```

Commenting Disassembler!

SOURCER™

■ SEE HOW PROGRAMS WORK
■ EASILY MODIFY PROGRAMS

SOURCER™ creates detailed commented source code and listings from memory and executable files. Built in data analyzer and simulator resolves data across multiple segments and provides detailed comments on interrupts and subfunctions, I/O ports and much more. Determines necessary assembler directives for reassembly. Includes a definition file facility to include your own remarks and descriptive labels, force data types, and more. Complete support for 8088/87 through 80286/287 and V20/V30 instruction sets. We welcome comparisons with any other product, because no product comes close to the ease of use and output clarity of SOURCER.

Sourcer is the best disassembler we've ever seen!
—PC Magazine, January 17, 1989, page 101

SAMPLE OUTPUT

Fully automatic

Program header

Assembler directives

Determines data areas and type

Detailed comments

Simulator follows segment changes

Easy to read format

```
     resetprn.lst    ResetPRN v1.01              Sourcer Listing    28-Jun-89   2:34 pm      Page 1

                                      PAGE  60,132
                                      :
                                      :                         RESETPRN
                                      :
                                      :    Created:   15-Apr-88
                                      :    Version:   1.01
                                      :    Passes:    3          Analysis Flags on: H
                                      :

     = 0008                           data_le       equ     8                   ; (0040:0008=378h)

                                      ;------------------------------------------------------------ seg_a -----
                                      seg_a         segment para public
                                                    assume cs:seg_a, ds:seg_a, ss:stack_seg_b

                                      resetprn      proc    far
    658E:0000                         start:
    658E:0000  EB 23                                jmp     short loc_1
    658E:0002  52 65 73 65 74 50                    db      'ResetPRN v1.01', 0Dh
    658E:0008  52 4E 20 76 31 2E
    658E:000E  30 31 0D
    658E:0011  0040                   data_2        dw      40h
    658E:0013  0D 0A 52 65 73 65      data_3        db      0Dh, 0Ah, 'Reset Printer? $'
    658E:0019  74 20 50 72 69 6E
    658E:001F  74 65 72 3F 20 24
    658E:0025                         loc_1:
    658E:0025  0E                                   push    cs
    658E:0026  1F                                   pop .   ds
    658E:0027 .BA 0013                              mov     dx,offset data_3  ; (658E:0013=0Dh)
    658E:002A  B4 09                                mov     ah,9
    658E:002C  CD 21                                int     21h               ; DOS Services  ah=function 09h
                                                                              ;  display char string at ds:dx
    658E:002E  B4 01                                mov     ah,1
    658E:0030  CD 21                                int     21h               ; DOS Services  ah=function 01h
                                                                              ;  get keybd char al, with echo
    658E:0032  3C 79                                cmp     al,79h            ; 'y'
    658E:0034  75 16                                jne     loc_3             ; Jump if not equal
    658E:0036  8E 1E 0011                           mov     ds,data_2         ; (658E:0011=40h)
    658E:003A  8B 16 0008                           mov     dx,ds:data_le     ; (0040:0008=378h)
    658E:003E  83 C2 02                             add     dx,2
    658E:0041  B0 08                                mov     al,8
    658E:0043  EE                                   out     dx,al             ; port 37Ah, printer-2 control
                                                                              ;  al = 8, initialize printer
    658E:0044  B9 8000                              mov     cx,8000h
    658E:0047                         locloop_2:
    658E:0047  E2 FE                                loop    locloop_2         ; Loop if cx > 0
    658E:0049  B0 0C                                mov     al,0Ch
    658E:004B  EE                                   out     dx,al             ; port 37Ah, printer-2 control
                                                                              ;  al = 0Ch, init & strobe off
    658E:004C                         loc_3:
    658E:004C  B4 4C                                mov     ah,4Ch            ; 'L'
    658E:004E  CD 21                                int     21h               ; DOS Services  ah=function 4Ch
                                                                              ;  terminate with al=return code
                                      resetprn      endp
                                      seg_a         ends

                                      ;------------------------------------------------------ stack_seg_b ----
                                      stack_seg_b   segment para stack
    6593:0000  00C0[FF]                             db      192 dup (0FFh)
                                      stack_seg_b   ends

                                                    end     start
```

(Source code output and inline cross reference can also be selected)

BIOS SOURCE

■ CHANGE AND ADD FEATURES
■ CLARIFY INTERFACES

for PS/2, AT, XT, PC, and Clones

The BIOS Pre-Processor™ with SOURCER provides the first means to obtain accurate legal source listings for any BIOS! Identifies entry points with full explanations. Resolves PS/2's multiple jumps for improved clarity. Provides highly descriptive labels such as "video_mode" and much more. Fully automatic.

SOURCER Commenting disassembler	$99.95	**UNPACKER**™ Unpack packed EXE files and more	$39.95
SOURCER with BIOS Pre-Processor	139.95	**ASMtool**™ Assembly source analyzer and flowcharter	89.95

Shipping & Handling: USA $3; Canada/Mexico $10; Other $15; CA Res. add sales tax; PS/2 trademark of IBM Corp.

All our products come with a 30 day money back satisfaction guarantee. Not copy protected. To order or receive additional information just call!

VISA 1-800-662-8266 MasterCard

V COMMUNICATIONS, INC.
3031 Tisch Way, Suite 802, Dept. M3, San Jose, CA 95128 (408) 296-4224

Reader Service Number 62

```
0000:60D0  4D 0E 06 14 02 D3 E8 8C-
           DA 03 C2 A3 1C 0B B8 00
0000:60E0  CD 20 22 08 00 9A F0 FF-
           0D F0 8C 02 42 05 99 02
```

The memory locations beginning at 0:60D0H represent the memory control block for RENEGADE's PSP/code. Another memory control block appears three paragraphs below. It begins at 0:60A0H and also points to RENEGADE's PSP segment at 060EH. This memory control block identifies the environment.

The information stored here includes the path and file name used to load RENEGADE. Removing a resident program requires the release of memory blocks belonging to the environment and the PSP/code.

We're ready to execute EVICT (Figure 5), the program to purge RENEGADE from resident memory. EVICT is an unsophisticated program and its sole purpose is to demonstrate how to remove a single resident program from DOS 2.10.

For EVICT to work properly, RENEGADE must be the only TSR installed. If necessary, reboot the system. Use the DEBUG "A" command to enter EVICT. The program restores the original vectors for Interrupts 9H and 1CH, then releases the two memory blocks. Save the program, execute from DOS, and return to DEBUG to display the two memory control blocks:

```
-NEVICT.COM
-RCX
CX ????
:0036
-RBX
BX ????
:0000
-W
-Q
A>EVICT
A>DEBUG
-D0:60A0,60EF
0000:60A0  4D 0E 06 02 00 05 C8 00-
           A3 BD 0B A1 02 00 8C 1E
0000:60B0  50 41 54 48 3D 00 43 4F-
           4D 53 50 45 43 3D 43 3A
0000:60C0  5C 43 4F 4D 4D 41 4E 44-
           2E 43 4F 4D 00 00 FF FF
0000:60D0  5A 0E 06 F2 79 D3 E8 8C-
           DA 03 C2 A3 1C 0B B8 00
0000:60E0  CD 20 22 08 00 9A F0 FF-
           0D F0 8C 02 42 05 99 02
```

Note the identifier byte at 0:60D0H contains the value 5AH, the designation that the next memory block is free. If you

Figure 5—Assembly Code for EVICT.COM

```
-A100
DS:0100  CLI                  ;DISABLE INTERRUPTS
DS:0101  MOV   AX,0000
DS:0104  MOV   ES,AX          ;SOURCE SEGMENT
DS:0106  MOV   SI,012E        ;SOURCE OFFSET
DS:0109  MOV   DI,0024        ;DEST. OFFSET (INT 9H)
DS:010C  MOV   CX,0004        ;MOVE 4 BYTES
DS:010F  REPZ
DS:0110  MOVSB
DS:0111  MOV   DI,0070        ;DEST. OFFSET (INT 1CH)
DS:0114  MOV   CX,0004        ;MOVE 4 BYTES
DS:0117  REPZ
DS:0118  MOVSB
DS:0119  STI                  ;ENABLE INTERRUPTS
DS:011A  MOV   AX,060E        ;PSP SEGMENT BLOCK
DS:011D  MOV   ES,AX
DS:011F  MOV   AH,49          ;RELEASE PSP BLOCK
DS:0121  INT   21             ;CALL DOS
DS:0123  MOV   AX,060B        ;ENVIRONMENT SEGMENT BLOCK
DS:0126  MOV   ES,AX
DS:0128  MOV   AH,49          ;RELEASE ENVIRONMENT BLOCK
DS:012A  INT   21             ;CALL DOS
DS:012C  INT   20             ;RETURN
DS:012E  DB    87,E9,00,F0    ;ORIGINAL INT 9H  VECTOR
DS:0132  DB    49,FF,00,F0    ;ORIGINAL INT 1CH VECTOR
```

◆ ◆ ◆

install a new TSR, it will be stored at segment 060EH.

EVICT was a very easy program to create. We know exactly where RENEGADE is located in resident memory and which interrupts it controls. For DOS 2.10, EVICT removes the first TSR installed, providing Interrupts 9H and 1CH were the only ones taken over.

The difficult assignment of writing a general utility is to keep track of where each TSR is loaded and the interrupt vector table prior to installation.

Consider the scenario of installing RENEGADE followed by POPUP, and then executing EVICT to remove RENEGADE:

```
A>RENEGADE
A>POPUP
A>EVICT
```

The memory control block defining RENEGADE's PSP is modified and can be viewed with DEBUG:

```
A>DEBUG
-D0000:60D0,60D4
0000:60D0    4D 00 00 14 02
```

The identifier byte prevails as 4DH; however, bytes 2 and 3 are filled with zeros. RENEGADE has been purged from resident memory and its memory blocks released. The next TSR is not stored in this vacated area, but immediately following the memory control block containing 5AH. A "hole" develops in resident memory and all newly-installed TSRs are placed above POPUP.

Software packages are available that can disable, and later enable, a resident program. These utilities are resident programs that save entire interrupt vector tables prior to the installation of every TSR.

To disable a resident program, all interrupt vector tables, except one, are loaded in the same order they were saved. The address of the disabled program will be excluded from the chaining of one handler to the next. Although the code remains in resident memory, the program has no mechanism for activation.

Reenable the dormant TSR by reinstating, in the proper order, the excluded interrupt vector table.

In Closing

As the number of TSRs continues to multiply, a set of guidelines would be helpful. Since IBM and Microsoft are not providing leadership in setting standards, the major software developers should. Some suggestions for guidelines include the assignment of hot keys, designation of interrupts to be taken over, chaining strategies from one handler to another, and procedure for removal of resident programs.

Will it ever work? It seems unlikely. The technology is changing too fast. By the time a set of standards are proposed and accepted, they'll probably be outdated. Until all TSRs can coexist peacefully, the best hope is for all of us to stay informed.

◆ ◆ ◆

Low Cost I/O For The PC

A/D Boards Hit The Cheap Seats

Bruce checks out three inexpensive multipurpose I/O boards to see if they measure up (and if so, how fast they do it). Plus he covers signal conditioning and board testing.

In past issues, I've tried to show you how to build your own hardware from chips, boards, wires, and solder. If no one makes what you need, or if you're trying to save money, or if you want to learn how to build circuits, then "rolling your own" is a great method.

Often, though, it's almost impossible to justify building it yourself. "If they make it in Taiwan," a saying goes, "chances are you can buy the board built and debugged for less than the cost of parts."

Low-cost data acquisition boards for the PC are a recent addition to the "it's cheaper to buy it than build it" phenomenon. Over the years, I've watched the ads for analog-to-digital (A/D) cards, and I've suddenly started seeing products in the $300 range which are too impressive to pass up.

In the first part of this article, I'll discuss some of the features of the boards I've tested. In the second part, I'll complete the demystification of the boards (and the electronics) you'll need to interface them to the real world.

What You Get

Most of these A/D boards begin with, but contain much more than, just an A/D converter. They try to solve everyone's problems, which can be an obnoxious trait in some people, but an ideal feature for a computer. In general, look for boards which provide—

- multiple channels of A/D input (usually a single multiplexed converter),
- several channels of digital-to-analog (D/A) conversion,
- and a number of digital I/O lines.

Often these boards will come with one or more counter/timers. For many situations, you'll only need one of these boards, your PC, and some simple electronics (described later).

The boards I'll talk about represent what's available in the $300 price range. I've tested three different products from four different companies (I'll explain shortly). All three can convert A/D, but beyond that they differ widely.

Real Time Devices, a company in Pennsylvania, builds and markets the AD1000 (along with several other interesting, very low-cost cards). The AD1000 has 8 channels of 12-bit A/D with a 20 microsecond conversion time.

Take the inverse to convert that to speed. A microsecond is 10^{-6}, so 1 divided by a microsecond is 10^{6}, or a megahertz. 20 microseconds translates to $\frac{1}{20}$ megahertz; or 50 kilohertz. The AD1000 contains 24 lines of digital I/O and 3 counter/timers, and comes in faster versions (for more money).

While snooping around for boards, I found two companies (and I'm sure there are more) that market the same board. Bo Ray at Rapid Systems (in Seattle) made no bones about it—"It's a good board," he said, "but the key to customer success isn't just selling the board, it's helping the customer solve the problem."

Advantech (in Taiwan) manufactures the PCL-712 (soon to be the new, improved PCL-812). Both Rapid Systems ($395) and Halted Specialties ($295) sell it.

Rapid Systems has a larger array of products, some they designed and built themselves. And their technical support seems top notch, which might explain the higher price. Halted Specialties, though, seems friendly and competent.

I'd probably opt for Halted if I were counting dollars and building a simple application. For bigger, more complex projects, I'd consider Rapid Systems (for more thorough technical support and slightly better documentation).

The PCL-712 has 16 12-bit A/D channels, two D/A channels, 16 separate digital input lines and 16 output lines, and 3 counter/timer channels (only one of which is available to the user). The other two channels are used for something called a pacer, which isn't explained very well but seems to be a way to trigger A/D conversions automatically. You'll need to talk to someone's tech support if you want to figure out exactly how this works.

The third product I looked at was the Lawson Labs Model 140 15-bit Analog Interface (for $265). This doesn't have all the digital I/O or counter/timers that the other boards do, but it does have a much higher resolution (15 bits vs. 12). It's a differential A/D while the others are single-ended. (I'll explain the terms shortly.) With this board you get more accurate, albeit much slower, readings (7.5 or 15 readings per second, instead of 30,000 or 50,000 readings per second).

A Complaint

In past years, I've worked with several A/D converter boards and I'm always playing sleuth just to figure out how to use the boards. That's because I never get complete documentation. I assume the person who created the documentation is the same engineer who designed the board. "Here are the data sheets, the rest is left to the student."

It's disappointing to see that boards for the PC follow the same traditions as those for more obscure busses. Some packages provide key information from the data sheets for the chips on their boards; some reproduce the data sheets.

By Bruce Eckel

Revolution2
501 N. 36th Street, Suite 163
Seattle, WA 98103

Unfortunately, none of the three boards I'm taking a look at in this article come with a schematic!

All the boards I've worked with in the past had crummy documentation, but at least they had schematics so I could puzzle out what was going on (although I would prefer lucid prose). If these companies provided better documentation with the boards, they'd have more (and happier) customers.

The "best" documentation comes with the Real Time Devices board. Although the printed material only consists of data sheets for the chips (which are hard to use without a schematic), the disk contains quite a bit of useful information. It includes sample programs in BASIC, Pascal and C, and a special programming language based on FORTH. The other boards had drivers or examples in BASIC only.

Board Applications

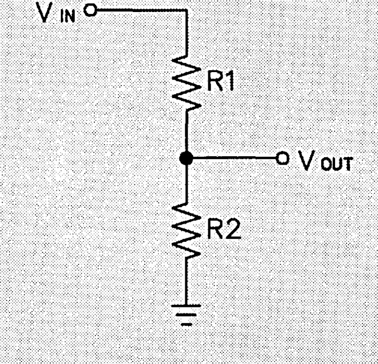
The four sections of the board (A/D, D/A, digital I/O, counter/timer) serve four purposes. The A/D converter accepts a voltage from the outside world and digitizes it (converts it into a number). The D/A converter converts a number into a voltage.

A digital control system uses both A/D and D/A: the A/D measures a voltage which represents something about the device being monitored (for instance, its temperature); the D/A sets a voltage that controls the device. The programmer must write an algorithm which performs the control but doesn't make the system unstable.

A/D and D/A conversion are commonly used alone, without being in a control system. You may simply want to monitor some values in an experiment or process with your A/D. Or you might replace a level control with a D/A so you can change the level with your computer instead of by hand.

You can't just hook a wire between the monitored device and the I/O board.

Figure 1—Voltage Divider.



You can use the counter/timer (as the name suggests) as a counter or as a timer. You'll use a counter (usually) to count some outside event, such as the number of times a beam of light is broken. Use a timer to set up a PC interrupt so that you can do something at predetermined intervals. In a control system, it's usually very important that you take samples or apply the control signals at regular intervals.

Use the digital I/O to read thresholds and to throw switches. If you just want to turn something on or off, or see if a value is above or below a certain level, digital I/O is the way to go.

All the above functions sound great, but a lot of people become confused when they realize they aren't just hooking up a stereo. You can't just hook a wire between the monitored device and the I/O board. You have to perform something called signal conditioning first, so you give the board a value it's capable of measuring (that won't fry a chip).

Digital Inputs

All digital inputs and signals to the counter must be TTL signals unless the documentation specifies otherwise. Roughly, this means that voltages between 0 volts and 0.8 volts will be interpreted as a logical zero. Voltages between 2.4 volts and 5 volts will be interpreted as a logical one. (Between 0.8 and 2.4V is undetermined.) If the inputs exceed 5 volts or drop below 0 volts, you can damage the hardware.

The challenge of conditioning the digital input signals is to take the voltage from the outside world and turn it into something the board can make sense of (and which won't do damage). We can do this easily with a voltage divider (see Figure 1).

A voltage divider is simply two resistors in series. We connect the bottom resistor to ground, and the top to the input voltage. The center point will be somewhere between the input voltage and ground. You can adjust its value by changing the values of the resistors. Either check it with a voltmeter or use the equation:

$$V_{out} = V_{in} * (R2 / (R1 + R2))$$

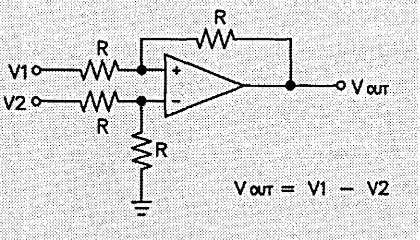
A voltage divider only works if the input voltage doesn't go below zero, since it only reduces the magnitude of the input voltage. The voltage divider doesn't prevent the input voltage from going negative.

If the input signal goes below ground, you can fix it with a simple op-amp circuit called a differencial amplifier (see Figure 2). Op-amps, or operational amplifiers, are inexpensive amplifiers on a chip which are indispensable for all types of signal conditioning. (I've introduced op-amps in my book *Computer Interfacing with Pascal & C*, available from Micro Cornucopia for $30.)

Digital Outputs

The signals from digital output lines are also usually TTL signals. This means that whatever device you turn on and off with the digital output must not only react properly to TTL voltage levels, it must also not require more current than the digital output can provide.

Figure 2—Differential Amplifier.



If you're driving some other TTL or CMOS chip, things will usually work just fine. If you try to power a relay or some other piece of hardware which requires more current or more voltage, you'll have to amplify the output.

You can change both the maximum voltage and the maximum current to be switched by the output line. A common approach uses a Darlington transistor (i.e., two transistors ganged together to provide high current throughput) and a diode (which allows inductive currents to flow in the other direction when the transistor shuts off—see Figure 3).

You can obtain inexpensive Darlington transistors with both transistors in a single package (the TIP120 is common), or with a gang of Darlingtons in a single IC package. Siliconix makes a device to replace the Darlington and the diode called a FETlington, part 2N7000.

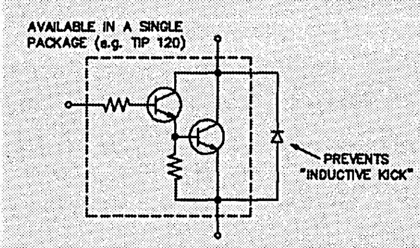
Notice that both Darlingtons and FETlingtons only allow current flow in one direction (which is okay, since you're usually just using DC). If you're switching an AC signal, you'll have to use a relay or a TRIAC or solid-state relay (see Chapter 6 in *Computer Interfacing with Pascal & C*).

Measurements Versus Signals

To understand analog signal conditioning, you must know whether you want to simply take a measurement (for example, getting the temperature) or capture a signal (as in capturing the waveform of someone's voice).

Taking a measurement is a simple process to manage—filter out as much noise as possible from the input signal and then take a measurement whenever you want. The speed of the A/D converter usually isn't critical when you're taking measurements, and the time between measurements may not be too important.

Figure 3—Darlington Pair.



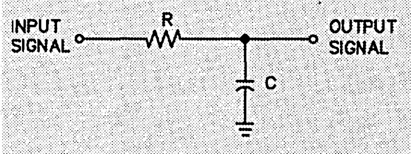
Capturing a waveform is a whole different can of sine waves. For one, you must sample the waveform at regular intervals (thus the need for the timer on the I/O board). Also, the sampling rate must be at least twice the highest signal frequency (this is the "Nyquist frequency"; even higher sampling rates are usually better). This means that for a voice signal, most of which is less than 10 KHz, your sampling rate would be at least 20 KHz.

High-speed A/D converters usually use a "successive-approximation" technique. They test the signal to see if it's above or below the halfway point of the full voltage range (which establishes the most significant bit of the result). If the input is above the halfway point, the value of the halfway point is subtracted and the remainder tested to see if it's above or below the quarter-way point (which establishes the penultimate bit). This goes on until it establishes all the bits, at which time the conversion is complete.

If the input signal changes very slowly, the conversion occurs before there's any appreciable change. However, if the signal changes quickly (for a fast or noisy waveform), it may change between bits, messing things up appreciably. To prevent this, you must: (1) filter the input signal to remove noise; and (2) use a sample-and-hold before your A/D converter. Luckily, both the AD1000 and the PCL-712 (which use successive-approximation) have a built-in sample-and-hold. I'll go into filtering shortly.

The Lawson Labs board uses a dual-slope A/D conversion technique. This charges a capacitor with the input voltage for a known period of time, then discharges it with a known voltage while counting the discharge time. When the capacitor is empty, the discharge time is proportional to the input voltage. This method is also called integrating.

Figure 4—Low-pass Filter.



Because it integrates, it ignores noise and doesn't have the problems associated with the successive-approximation converters. So you don't need a sample-and-hold, and you often don't need a filter. However, a dual-slope converter is slow, though you get much greater resolution (number of bits). The Lawson Labs board is appropriate for taking high-accuracy measurements, but not for capturing waveforms.

Filtering

If you use a successive-approximation converter with a built-in sample-and-hold, you only need to worry about: (1) voltage level adjustments; and (2) filtering out noise and frequencies higher than those you wish to capture.

Common filters are high-pass, which pass frequencies above a certain point, low-pass, which pass frequencies below a certain point, and band-pass, which only pass frequencies within a specified range. Generally we describe the quality of the filter by the "sharpness" of the corner(s) between the pass band and the stop band.

C CODE FOR THE PC

source code, of course

	MS-DOS File Compatibility Package (create, read, & write MS-DOS file systems on non-MS-DOS computers)	$500
	Bluestreak Plus Communications (two ports, programmer's interface, terminal emulation)	$400
NEW!	C+O Data Structures Library (multiple inheritance, messaging, excpetion handling, call-back)	$375
NEW!	dB2c (dBase-to-C translator; includes dB_Files for C anddB_Tools for C)	$325
NEW!	pBase (relational DBMS with debugging calls and sparse table & repeated field support)	$325
	CQL Query System (SQL retrievals on B-trees plus windows)	$325
Updated!	GraphiC 5.0 (high-resolution, DISSPLA-style scientific plots in color & hardcopy)	$325
	PC Curses (Aspen, Software, System V compatible, extensive documentation)	$290
NEW!	Code Base (database manager, dBase and Clipper compatibile indes & data files; Version 4.0)	$260
NEW!	MEWEL (extensible window and event library by Magma Software; message-passing & object-oriented; SAA-compatible; dialog editor)	$250
Updated!	TurboTEX (Release 2.0; HP, PS, dot drivers; CM fonts; LaTEX; MetaFont)	$250
	Greenleaf Data Windows (windows, menus, data entry, interactive form design; specify compiler)	$220
	Greenleaf Communications Library (interrupt mode, modem control, XON-XOFF; specify compiler)	$175
NEW!	QuickGeometry Library (large collection of mathematics, graphics, display & DXF subroutines for CAD/CAM/CAE/CNC)	$170
	Sherlock (C debugging aid)	$170
	CBTree (B+tree ISAM driver, multiple variable-length keys)	$165
	AT BIOS Kit (roll your own BIOS with this complete set of basic input/output functions for ATs)	$160
	Greenleaf Functions (296 useful C functions, all DOS services; specify compiler)	$160
	WKS Library Version 2.0 (C program interface to Lotus 1-2-3, dBase, Supercalc 4, Quatro, & Clipper)	$155
	OS/88 (U••x-like operating system, many tools, cross-development from MS-DOS)	$150
	ME Version 2.1 (programmer's editor with C-like macro language by Magma Software; Version 1.31 still $75)	$140
NEW!	VTEK 4.3 (terminal emulator; VT102/100/52 & Tek 4010/14/15; LIM 4.0; 26 printers; Kermit & XMODEM)	$135
	Vmem/C (virtual memory manager; least-recently used pager; dynamic expansion of swap file)	$140
	Turbo G Graphics Library (all popular adapters, hidden line removal)	$135
	TurboGeometry (library of routines for computational geometry)	$125
Updated!	Install 2.3 (automatic installation program; user-selected partial installation; CRC checking)	$120
	TE Editor Developer's Kit (full screen editor, undo command, multiple windows)	$105
Updated!	Minix Operating System (Version 1.3; U••x-like operating system, includes manual)	$105
	HyperText Viewer (simple hypertext system; multi-file documents; includes Tiny Curses)	$100
Updated!	PC/IP (CMU/MIT TCP/IP for PCs; Ethernet, Appletalk & NETBIOS drivers, RVD, update by Dan Lanciani)	$100
	B-Tree Library & ISAM Driver (file system utilities by Softfocus)	$100
	The Profiler (program execution profile tool)	$100
	QC88 C compiler (ASM output, small model, no longs, floats or bit fields, 80+ function library)	$90
NEW!	Otter 1.0 (beautiful theorem-prover by Bill McCune; includes manual & two books by Wos; complete starter kit)	$80
	C Windows Toolkit (pop-up, pull-down, spreadsheet, CGA/EGA/Hercules)	$80
	JATE Async Terminal Emulator (includes file transfer and menu subsystem)	$80
	Polyglot Lisp-to-C Translator (includes Lisp interpreter, Prolog, and simple calculus prover)	$80
	MultiDOS Plus (DOS-based multitasking, intertask messaging, semaphores)	$80
Updated!	Make (macros, all languages, built-in rules)	$75
	eval() (C function to evaluate ASCII infix expression string; 17 built-in functions)	$75
	XT BIOS Kit (roll your own BIOS with this complete set of basic input/output functions for XTs)	$75
	Professional C Windows (lean & mean window and keyboard handler)	$70
	Heap Expander (use LIM-standard expanded memory as an extension of the heap)	$65
	lp (flexible printer driver; most popular printers supported)	$65
	SYSKIT (rommable or TSR debug/monitor; easily expanded)	$60
	Quincy (interactive C interpreter)	$60
	Symtab (general-purpose symbol table construction and management package)	$60
	PTree (general-purpose parse tree construction and management package)	$60
Updated!	Coder's Prolog (Version 3.0; inference engine for use with C programs)	$60
NEW!	Async-Termio (Unix-compatible general terminal interface for MS-DOS	$55
	Backup & Restore Utility by Blake McBride (multiple volumes, file compression & encryption)	$50
	SuperGrep (exceptionally fast, revolutionary text searching algorithm; also searches sub-directories)	$50
	OBJASM (convert .obj files to .asm files; output is MASM compatible)	$50
	Polyglot TSR Package (includes reminder, bookmark, virus catcher, cache manager, & speech generator)	$50
	Multi-User BBS (chat, mail, menus, sysop displays; does not include modem driver)	$50
NEW!	Fortran-to-C Translator by Polyglot	$40
	Virtual Memory Manager by Blake McBride (LRU pager, dynamic swap file, image save/restore)	$40
	Heap I/O (treat all or part of a disk file as heap storage)	$40
	Biggerstaff's System Tools (multi-tasking window manager kit)	$40
	OOPS (collection of handy C++ classes by Keith Gorlen of NIH; Version 2.2)	$35
	Bison & PREP (YACC workalike parser generator & attribute grammar preprocessor; now includes documentation)	$35
	PC-XINU (Comer's XINU operating system for PC)	$35
	CLIPS (rule-based expert system generator, Version 4.2)	$35
	Tiny Curses (Berkeley curses package)	$35
	Polyglot RAM Disk (change disk size on the fly; includes utilities)	$30
	Clisp (Lisp interpreter with extensive internals documentation)	$30
	Translate Rules to C (YACC-like function generator for rule-based systems)	$30
	6-Pack of Editors (six public domain editors for use, study & hacking)	$30
	Crunch Pack (14 file compression & expansion programs)	$30
	Pascal P-Code Compiler & Interpreter or Pascal-to-C Translator (Wirth standard Pascal)	$25
	ICON (string and list processing language, Version 7.5)	$25
	FLEX (fast lexical analyzer generator; new, improved LEX; Version 1.1)	$25
	LEX (lexical analyzer generator; an oldie but a goodie)	$25
	AutoTrace (program tracer and memory trasher catcher)	$25
	Data Handling Utilities in C (data entry, validation & display; specify Turbo C or Microsoft)	$25
	Arrays for C (macro package to ease handling of arrays)	$25
	A68 (68000 cross-assembler)	$20
	List-Pac (C functions for lists, stacks, and queues)	$20
	XLT Macro Processor (general purpose text translator)	$20
	Data	
	Protein Sequences (over 10,000 sequences; includes demo disk of Pearson FAST/A programs)	$60
	Smithsonian Astronomical Observatory Subset (right ascension, declination, & magnitude of 258,997 stars)	$60
NEW!	Moby Words (500,000 words & phrases, 9,000 stars, 15,000 names)	$55
	U. S. Cities (names & longitude/latitude of 32,000 U.S. cities and 6,000 state boundary points)	$35
	The World Digitized (100,000 longitude/latitude of world country boundaries)	$30
	KST Fonts (13,200 characters in 139 mixed fonts: specify TEX or bitmap format)	$30
	USNO Interactive Computer Ephemeris (high-precision moon, sun, planet & star positions)	$30
	NBS Hershey Fonts (1,377 stroke characters in 14 fonts)	$15
	U. S. Map (15,701 points of state boundaries)	$15

The Austin Code Works
11100 Leafwood Lane
Austin, Texas 78750-3409 USA

acw!info@uunet.uu.net

Voice: (512) 258-0785
BBS: (512) 258-8831
FAX: (512) 258-1342

Free surface shipping for cash in advance For delivery in Texas add 7% MasterCard/VISA

Reader Service Number 4

There are two common ways to build filters. Passive filters consist of resistors, capacitors, and (in some situations) inductors. Simple low-pass filters are cheap and easy to make from resistors and capacitors (see Figure 4), but they don't have very sharp corners and are thus appropriate only when the frequency elements you want to remove are far into the stop band.

For example, if you want to sample a DC signal you'd commonly use a low-pass RC filter to remove 60 Hz noise generated by household AC power. The optional connector boards which come with both the AD1000 and PCL-712 have spaces on the circuit board where you can solder resistors and capacitors to form low-pass filters.

Active filters use op-amps, resistors, and capacitors. Active filters are much more precise and have much sharper corners than passive filters, so the frequency elements you want to remove can be much closer to the frequency elements you want to keep.

Op-amps are fairly cheap, so an active filter isn't much more expensive than a passive filter, although the design of an active filter is slightly more complex. An excellent tutorial on both passive and active filter design is *Introduction to Filter Theory* by Johnson, Hilburn and Johnson.

You can now buy a switched-capacitor filter, which fits on a single chip. Since it's less susceptible to the drift found in ordinary components, it's more accurate over long periods of time.

Single Vs. Double-ended Converters

Both the AD1000 and the PCL-712 have "single-ended" inputs, while the Lawson Labs board has "differential" inputs. Single-ended inputs are signals referenced to a common ground. The Real Time Devices board, for example, has 8 single-ended inputs, which means that all the input signals must have their grounds tied together.

In many situations, this works just fine—you may have a group of signals which already have their grounds connected. Here, you simply take the ground from these signals and tie it to the ground on the I/O board.

In some situations, however, you might be measuring a signal between two points, neither of which is ground. Or, you may not be able to tie grounds together. Here you'll need to use a differential converter, which measures the voltage between two points. A single-ended converter also measures the voltage between two points, but one of those points is always the common ground.

Common-mode Signals

The ideal differential converter will measure the precise voltage between two points.

However, even though neither of the converter's inputs connects to ground, the converter itself has a connection to ground. Let's say that you're measuring a signal that's 6V and 5.8V. The converter will output 0.2V, the difference. But there's another voltage to deal with, the common-mode voltage. It's the average voltage between ground and the two inputs (5.9V here).

Common-mode voltage causes errors in the digital value of the converter. This error is specified (in both differential converters and differential amplifiers) as the common-mode rejection ratio (CMRR). A CMRR of 10^6 means that the device processes one volt of common mode as though it were a differential signal of 1 microvolt at the input.

Often, instrumentation amplifiers (op-amps designed for use in a differential configuration for instrumentation) have a much better CMRR than an A/D converter. In addition, they'll take a differential signal and turn it into a voltage with respect to ground. Thus, with one device you reduce the common-mode error and allow a single-ended board to read a differential signal.

Input Impedance

There's another reason you may want to use an instrumentation amplifier at the front end of your system. An instrumentation amplifier has an especially high input impedance. The input impedance is what the amplifier looks like to the circuit it measures.

In effect, when you measure something, you attach a resistor to ground at the point you're measuring. If the value of the resistor is low, a lot of current will be drawn from the circuit and the behavior of the circuit will change. This means you won't see the real behavior of the circuit while you're measuring it.

It also means that the ideal input impedance of the system you use to perform measurements should be infinite so you don't draw any current from the circuit being measured.

Practically, however, no amplifier has an infinite input impedance, and many circuits aren't affected if you remove a little current. In a circuit where current drawn by the measuring system causes significant errors, you should use an instrumentation amplifier to maximize your input impedance.

Dynamic Range

Dynamic range refers to the largest and smallest values a system can measure. If you have a 12-bit converter and the largest value you want to measure is 100 volts (obviously this signal would have to be conditioned before handing it to the converter), and you want to resolve values down to 0.01 volt, then you have a dynamic range problem.

Your converter will need 10,000 counts to measure up to 100 volts in 0.01 volt increments, and a 12-bit converter only has 2^{12}, or 4096, counts. To solve the dynamic range problem, some boards allow you to change the gain of the input amplifier; you use a lower gain for higher voltages, and increase the gain when the voltage is lower.

Some boards require you to use a jumper to set the gain (which isn't a very flexible solution). Real Time Device's AD100/500 12-bit analog input boards have software-programmable gains so you can change the gain according to the signal. Instrumentation amplifiers often have programmable gains, so if you build your own front end, you can use some of the board's I/O lines to control the amplifier gain.

Grounding & Ground Loops

Digital folk harbor an illusion (actually it's one of several) that "ground is 0 volts." In truth, currents flow through ground wires and traces. Any time current flows, there's an associated voltage drop. In a ground path, this voltage is generally very small. However, if you're measuring tiny voltages, the voltages in the ground path can show up in your measurements and cause errors. We call this problem a ground loop.

Here's what happens. Suppose you have a long ground trace on a circuit board. One end of the trace, which I'll call downstream, dumps into the power supply ground. Somewhere in midstream, some digital chip grounds attach. When these chips switch, they dump current into the ground and raise the voltage at that ground point (and every point upstream) by a fraction of a volt.

Now suppose you attach the ground of an input amplifier upstream from the digital chips. The input amplifier, along with the input voltage, uses the ground to determine the output voltage of the amplifier. Thus, if the ground changes, the output voltage changes. When you're squinting at small input signals, the effect can be significant.

There are several guidelines you can use to eliminate ground loop problems—

(1) All ground traces should be as large as possible.

(2) Analog and digital grounds should be separated by providing completely separate paths to the power supply.

(3) And, if possible, all critical chips should have their own traces to the power supply ground, or to a single point where all the ground traces meet.

Simple Test: Digitize A Battery

You can easily test an A/D converter board by using it to digitize a 1.5 volt flashlight battery. You can hook the battery directly to the inputs of most boards, since 1.5 volts is usually within the range of a typical board.

Then write a very small piece of code to read the value of the A/D converter and print it to the screen. When trying to solve a problem which involves both hardware and software, I find it useful to start with something simple so I can see if my problems are hardware or software. Assuming you hook up the battery to the board correctly and the board is operational, the problem you find with this configuration will always be in hardware.

Since a battery is such a quiet, stable source of input, any noise in the least-significant bits of the conversion will almost always be due to problems in the design of the board.

A Gotcha

Although boards with A/D converters on them have become very cheap, you're in for a surprise when you try to calibrate one. The manufacturer will calibrate the board, but it's best to test them when they arrive. All boards eventually drift out of spec and you need to recalibrate them.

The problem is that calibration requires equipment which is usually unavailable to the average user. For a 12-bit A/D converter, you need a calibrator with overall accuracy better than 60 ppm (parts per million) and a 5 1/2 digit digital voltmeter (DVM). Thus, the board may be under $300 but the equipment necessary to calibrate it can cost thousands. (I'm a novice when it comes to calibration, but it seems like you could get by with the DVM and a couple of standard cells). Some manufacturers will recalibrate the board for you, for a fee, and some commercial laboratories offer a calibration service.

Future Things

Next issue (I/O Yet Again), I plan to apply all the information I've told you about here. The example I have in mind is to capture a voice waveform from a microphone, display the waveform on the screen, and then show the frequency components of the waveform by doing a fast Fourier transform.

Until then, pleasant dreams and good filtering.

Editor's note: I've noticed that everyone does fast Fourier transforms these days. I wonder if they're anything like the (much) slower Fourier transforms we did by hand in engineering lab.

Products Mentioned

Halted Specialties
PCL-712
$295
3500 Ryder Street
Santa Clara, CA 95051
(800) 4-HALTED Outside California
(408) 732-1573 Inside California

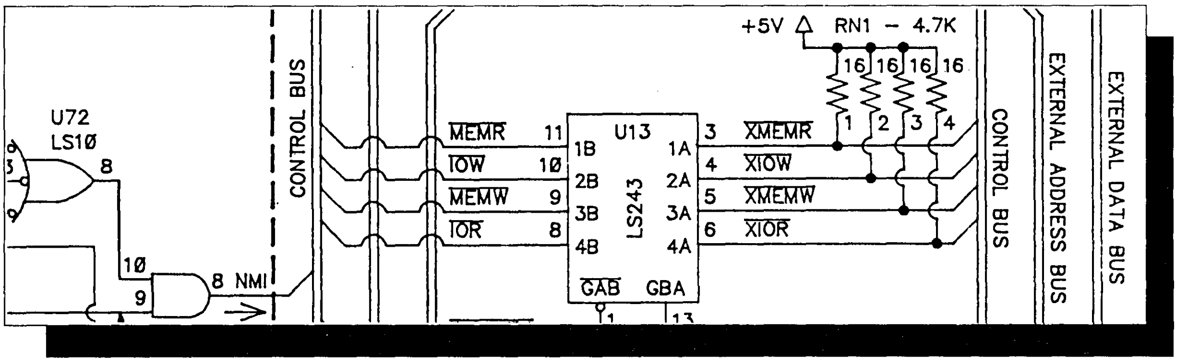
Rapid Systems
PCL-712
$395
433 N. 34th St. Seattle, WA 98103
(206) 547-8311

Real Time Devices, Inc.
AD1000
$295
P.O. Box 906
State College, PA 16804
(814) 234-8087 FAX 234-6864

Lawson Labs Model 140
from Personal Computing Tools
as the ACCUE-140
$265
17419 Farley Road
Los Gatos, CA 95030
(408) 395-6600 FAX 354-4260

◆ ◆ ◆

MICRO CORNUCOPIA **XT SCHEMATIC**



At last you can plumb the mysteries of your computer with this single sheet schematic of the IBM XT's main board. A wealth of information for both True Blue *and* clone owners.

Need to know just how a non-maskable interrupt occurs (and how to mask it)? Is your keyboard dead (or do you just want to know how to disable it)? A trip through our schematic will answer your questions.

Although clones use slightly altered board layouts and different chip location names, they're close enough to the original for this schematic to be very useful. As an example — you have a dead clone. Lil sucker won't even beep. A look at the schematic shows the location of parallel port A. You know that the power-on self test loads a checkpoint number into port A before each test. So now all you have to do is read port A with a logic probe to see how far the system went before it puked.

We include these checkpoints and other trouble shooting information with the schematic.

IBM PC-XT Schematic	**$15.00**

CP/M KAYPRO SCHEMATICS

Of course, we still provide a complete schematic of the processor board in your CP/M Kaypro. It's logically laid out on a single 24'' by 36'' sheet and comes complete with an illustrated theory of operation that's keyed to the schematic. You get detailed information available nowhere else.

For instance, those of you with the 10 and newer 84 systems get a thorough run down of the processor board's video section complete with sample driver routines. All packages contain serial and parallel port details and programming examples. Also coverage of the processor, clock, I/O, and disk controller (information that's not even available in Kaypro's own dealer service manual!).

Kaypro II & IV (pre-84)	**$20.00**
Kaypro 10 (without modem)	**$20.00**
Kaypro 2, 4, and 10 (84 series)	**$20.00**

NOTE: These packages cover *only* the main boards. You're on your own when it comes to disk drives, power supplies, video cards, etc.

Phone Orders: (503) 382-5060 or 1-800-888-8087 Monday-Friday, 9 AM - 5 PM PST

ERAC CO.

8280 Clairemont Mesa Blvd., Suite 117
San Diego, California 92111
(619) 569-1864

AT/BABY AT

8 Meg CPU Board
Zero Wait State
8 Expansion Slots
640K RAM On-Board
Math Co-processor Option
Phoenix Bios
200 Watt Power Supply
Hercules Compat. Video Bd.
Parallel Port
2 Serial Ports Active
Game Port
Clock/Calendar
Hard Disk & Floppy Controller
20M Hard Drive
1.2M 5¼" Floppy Drive
360K 5¼" Floppy Drive
5061 Keyboard
Case with Turbo & Reset,
Hard Drive Light and
Keyboard Disable Switch
Amber Graphics Monitor

$1299

EGA ADD $400
40M HD ADD $150
10 MHz ADD $50

XT/TURBO

Motherboard
5 & 8 MHz Switchable
8088 — V20 Optional
Optional Co-processor
8 Expansion Slots
ERSO or Bison Bios
640K RAM
150 Watt Power Supply
Hercules Compat. Video Bd.
Parallel Port
2 Serial Ports Active
Game Port
Clock/Calendar
Hard Disk and
Floppy Controller
20M 5¼"Hard Drive
2 ea. 360K 5¼" Floppy Drive
AT Style Keyboard
Standard Slide Case
Amber Graphics Monitor

★ ★

$949

EGA ADD $400
40M HD ADD $150
5 & 10 MHz ADD $21

ELGAR

UNINTERRUPTIBLE POWER SUPPLIES

400 Watt MODEL IPS400+
$650

Power distribution center and sine-wave UPS. Only 2" high.

560 Watt MODEL IPS560
$350

Sinewave, 560W complete with batteries.

400 Watt MODEL SPR401
$180

Supplies may have minor cosmetic damage, but are electrically sound. Squarewave output. Run on internal or external 24VDC battery when line goes down. Typical transfer time = 12MS. Battery supplied. For AT, XT & Kaypro.

★

NEW 24V INTERNAL BATTERY $75

NiCds

AA Cells .6ah	$1.00
12V Pack AA Cells .6ah	6.50
Sub-C Cells 1.5ah	1.50
12V Pack Sub-C	10.00
Double D Cell 2.5V 4ah unused	8.00
C Cells	1.75
7.2V RC-Pack 1.2ah	18.00

GEL CELLS

6V 8ah	$6.00
12V 20ah	25.00
12V 15ah	15.00
12V 2.5ah	8.50
D Cell 2.5ah	2.00

ROBOTICS

5V DC Gear Motor w/Tach 1"x2"	$7.50
Joystick, 4 switches, 1" knob	5.00
Z80 Controller with 8-Bit A/D	15.00
Brushless 12VDC 3" Fan	7.50
12V Gear Motor 30 RPM	7.50
Cable: DB9M-DB9F 1 ft. length	2.00
High Voltage Power Supply Input: 15-30V DC Output: 100V 400V 16KV	6.50

KAYPRO EQUIPMENT BARGAINS *We Repair CPM Kaypros*

9" Green Monitor — 83	$50
9" Green Monitor — 84, K16	60
9" Amber CRT	$45
Keyboard	75
PRO-8 Mod. to your board	149
Host Interface Board	15
Replacement Power Supply	$50
Drivetek 2.6M FD	75

CPM COMPUTERS

K4-83	$350	K2-84	$400
K4-84	425	K4X	425

IC'S

81-189 Video Pal	$15.00
81-194 RAM Pal	15.00
81-Series Char. Gen. ROMs	10.00
81-Series Monitor ROMs	10.00

TEST EQUIPMENT

OSCILLOSCOPES

TEK 7403N/7A18N/7B50A 60 MHz	$650
TEK 465B Dual Trace 100 MHz	1395
TEK 2215A Dual Trace 60 MHz	850
Scope Probe x1, x10 100MHz	25
USM338 50MHz Dual Trace Delayed Sweep (as is)	145

ANALYZERS

TEK 491 10MHz - 40 GHz	$4000
Biomation 805 Waveform Rcrdr	195
Biomation 8100 2-Channel Waveform Recorder	495
HP1600A Logic Analyzer	395
HP1600A/1607A Logic Anlyzr	595

MISC.

Optronics 550 MHz Freq Cntr	$95
Data Royal Function Gen F210A	195

HOURS: Mon. - Fri. 9 - 6 — Sat. 10 - 4
MINIMUM ORDER — $15.00
TERMS: VISA, MasterCard, Certified Checks, Money Order, NO COD. Visa and MasterCard add 3%. *Personal checks must clear BEFORE we ship.* Include shipping charges. California residents add 7% Sales Tax. For more information please call.

CPU & RAM & MISC.

41256-12	$7.50	41256-15	$6.00
4164-10	2.50	4164-12	2.10
4164-15	2.00	4164-20	1.25
MK48Z02B-20	10.00		
Dallas D1220Y	10.00		
SIP DRAM 256-12	7.00		
2716	3.50	2732	3.75
2764	4.00	27128	6.50
27256	5.25	27512	7.00
MC68000-8 CPU	8.00		
Z80 CPU	.75	Z80A CPU	1.50
Z80 CTC	1.50		
Z80A PIO	2.00	Z80A SIO	5.00
8089-3	6.50		
80C85A	4.50	8088	6.50
8212	2.00		
8251	1.50	8253-5	1.50
8255-2	3.50	8255-5	2.50
D8284A	2.50		
D8749	7.00		
6845	5.00		
1793	6.00	1797	7.00

★ SPECIAL ★

External 3½" Floppy Drive, 720K, Top Loading, 5V Only, w/Docs **$55**

SWITCHERS

5V/9.5A, 12V/3.8A, -12V/.8A	$39.00
5V/3A, 12V/2A, -12V/.4A	19.50
5V/6A, 12V/2A, -12V/1A	29.00
5V/6A, 24V/1¼ A, 12V/.6A, -12V/.6A	29.00
5V/10A	19.00
5V/20A	24.00
5V/30A	39.00
5V 100A	100.00
5V 120A	110.00

★ SPECIAL ★

IBM PS2 Model 25. Freight damaged. Works perfect. **$550**

★ SPECIAL ★

AT 80286-6 CPU BOARD
with reset and mono/color switch
Connector for KB, Battery & SPKR
Phoenix Bios
(tested with Award 3.03)
6MHz, can be upgraded to 8 or 10MHz
Used with backplane, add memory board, I/O board, etc.

***ONLY* $99**

Reader Service Number 93

C'ING CLEARLY

Computer Life

Creating Mutations On Your Very Own Computer

By Scott Robert Ladd
705 West Virginia
Gunnison, CO 81230
(303) 641-6438

Will there be life after computers? Who knows. There certainly was life before computers, even though it wasn't worth watching.

It's late May as I write this, and spring is happening all around. It's the time of flowering in these parts; bright petals of color cover the forest floor. On the 17th of April, our daughter Elora was born—a perfect child, if I might say so myself. She is quite a joy, day and night.

My wife and I took our new child on her first wilderness hike when she was seven weeks old. We took the Mill-Castle Trail along Mill Creek into the West Elk Wilderness. Two miles in, we stopped when the trail crossed Mill Creek (primarily because the baby couldn't swim if she fell in) and had lunch in an Aspen grove.

There wasn't a human in sight; the nearest was probably several miles away. In the shadow of those jagged cliffs, we could feel a presence. We would have stayed later, but a fast-moving mountain thunderstorm rolled down the valley, chasing us back to our Jeep. We got soaked, but enjoyed every minute of it. The planet we live on is fantastic.

I suppose I'd better start talking about computer programming. After all, that's what this column is *supposed* to be about.

TOOLKIT

The craft of producing computer programs has undergone a massive change during the last decade. For those of us who have been a part of this industry since its earliest days, it's like walking in the Emerald City after fighting off the Wicked Witch. Software development tools of amazing power abound for hardware platforms of dazzling capability. Unfortunately, we often take it all for granted.

With the powerful hardware and fantastic software development environments, you'd think applications would be equally wonderful. Alas, that is not so. Major companies release buggy versions of their software so that they can meet "marketing windows." Many programs are hard to use, or are missing key features.

As programmers, we need to use these advanced software development tools to create the next generation of software. As consumers, we need to put our foot down and say, "I'm not going to take it any longer!" Buy software which works, support vendors who support you—and raise the devil with those companies who can't or won't do the job. If we continue accepting the crud, we'll continue getting it.

The quality of public domain software is one of my barometers of programming practice. When affordable microcomputers first appeared, public domain software flourished. Everyone wanted to share, knowing that their combined knowledge would make computer software more powerful. There was a sense of community.

Today, microcomputers are big business. There's less public domain and more shareware and crippleware. Legal battles and license agreements have replaced the spirit of community. Yuck. What would have happened if Ward Christianson had made XModem shareware?

I don't mean to say that I don't like shareware; some of it (PC-Write, PC-Outline, Procomm, FST Modula-2, etc.) is excellent and worth more than the authors charge. But 99% of the shareware I've seen is crap—pure and simple.

Somebody writes a program and distributes it with a request for money. Often, the program is badly documented, buggy, or just downright useless. Why distribute it if it isn't any good? Don't people have pride in their work? Or are they just interested in a quick buck? Somehow, I think the latter attitude is becoming more and more common as time passes.

That's why all the source code which appears in this column is public domain. I'd appreciate credit for it if you use it, merely as a courtesy. If I'm going to publish it, though, I expect people to be able to use it if they want to.

Life

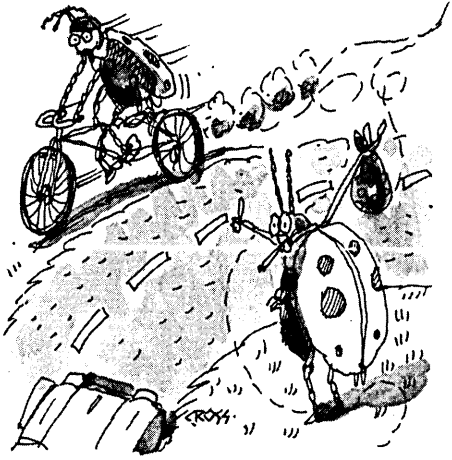
This part of the column is going to eat space again.... As usual, something came up which changed my plans for this issue. I *was* going to talk about infinite-precision math and methods of calculating pi to an astounding number of digits. That idea will have to wait. I'm building it as a C++ class library, and it will use some very sophisticated algorithms. Also, I need to do a bit of brushing-up on calculus. So, I came up with something different—and much more interesting.

Computers have been used for years to simulate the real world. One of the most popular simulations is Life, an elementary imitation of a colony of cells. John Conway of the University of Cambridge (England) developed the system in the late 1960s.

Cells live, die, and are born according to a simple set of rules. An empty cell with exactly three live neighbors gets born. When a live cell has four or more live neighbors, it dies from overcrowding. Any live cell with two or three live neighbors simply survives.

From these three rules came a fascinating pastime for many people. The computer is given a beginning pattern of live cells and then begins processing it according to these rules. Fluctuating and oscillating patterns emerge and have been given names such as "floaters" and "guns."

You have even more fun when you change or add to the rules. People have become quite creative in enhancing Life, adding different universes, modifying rules, and mutating cells. It's fascinating to watch the patterns as they change; even people who hate computers (are there such cretins? <grin>) will spend hours watching Life run its course.



Making Life More Interesting

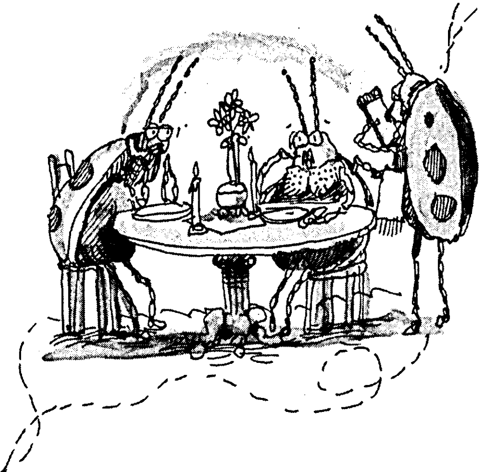
Even within its extended forms, however, Life is not truly lifelike. It is a system for modifying a pattern according to a set of very simple rules. The rules don't change once the pattern is running.

However, real life includes complicated interactions between organisms and their environment. Living creatures evolve as their environment changes. Is it possible to create evolving creatures in a computer?

Yes! And it's fun! (Oh no! Not fun, we can't have *that* around here....)

I first came upon this idea via A. K. Dewdney's "Computer Recreations" column in *Scientific American.* Every month, Dewdney covers subjects ranging from prime numbers to fractals. In the last year-and-a-half, he has covered computer-based life at least three times; in the December 1987, February 1988, and May 1989 issues.

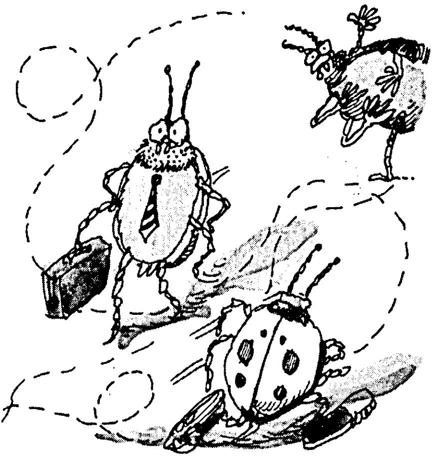
An interesting facet of Dewdney's column is that he does not publish any source code, or even pseudocode. Dewdney's column gives you an excellent idea of where the cutting edge of computer technology is. He discusses algorithms and ideas—and then leaves the implementation up to the reader. This is a great way to stretch your programming- and algorithm-development skills.



In the May issue, Dewdney looks at a program called "Simulated Evolution," written by a California high school teacher named Michael Palmiter. Palmiter's "bugs" live in a "dish" which contains "bacteria," the bugs' primary food.

The bugs begin by randomly moving about in the dish, eating any bacteria they encounter. The bacteria provide energy for the bugs to continue moving, and when the bugs are old enough and have sufficient energy, they reproduce by breaking into two new bugs. All this takes place right on the screen.

What makes Palmiter's ideas interesting is that he has built a small amount of "genetic" information into each bug. Each bug has a set of genes which control its movement. Lottery determines movement; every turn picks a random direction. However, as the bugs reproduce, they mutate. The mutation increases the probability that a bug will go in a certain direction. Where the first bugs tend to jitter randomly, later bugs take on a more directed pattern.



Those bugs which find food live long enough to reproduce, passing on their characteristics to a new generation. Jitter-bugs have a problem when the food in their local area becomes scarce, because they tend not to travel far from their origin point. Cruiser-bugs, which travel primarily in one or two directions, are much more likely to find new sources of food. Zigzag-cruisers move in an erratic pattern which tends to lead them in one direction, but with more area coverage.

Palmiter's program is simple, yet elegant. As time goes on, the bugs seem to develop behavior patterns—even if they are a bit random in nature. But is it real "life on a computer screen?" No; real life, even for single-cell organisms, is far more complicated than Palmiter's simulated bugs. Nevertheless, he has made a major step forward in the creation of real artificial life.

I've taken the ideas developed by Palmiter and expanded upon them. My first effort is the program shown in Figure 1. CRITTERS.C is written for Microsoft's C 5.10 and QuickC 2.0; an executable version is available on the Micro C BBS or on the issue #49 disk.

The program requires an IBM-PC compatible with an EGA card and at least 256K of RAM. I will create versions compatible with Turbo C 2.0, Zortech C, and WATCOM C 7.0; I may even write an object-oriented version of the program in C++. Making modifications for non-Microsoft C compilers consists of changing the graphics calls to match each compiler's library. It isn't very difficult to do.

The version of CRITTERS presented here has several features of my own design. Bugs are known as critters, and I refer to the bacterium generically as food. The critters store in a doubly-linked list and are dynamically allocated. You define a critter by a structure which contains the current location, age, energy level, and pointers to adjacent critters in the list. Food is merely a different-colored pixel.

Critters move either up, down, left, or right. Initially, they have an equal chance of moving in any direction. A critter gains energy when it occupies the same position as a food pixel.

When the critter is mature (having reached a specified age), and it has sufficient energy, it creates a new bug—and has its age reset to zero. New bugs and their parents each get half of the energy the parent had when the birth took place. New critters have an increased chance of moving in a particular direction.

If a critter reaches a certain age, and has not yet reproduced, it dies of old age. Critters also die from starvation (when they have a zero energy level).

Details

Take a look at the constants I've defined at the beginning of the code. These 20 values specify the program's behavior, and changing them can radically alter the lives of the critters. Try changing just one value, to see what happens. Changes in the values for food_value, move_cost, max_mutant, and max_age make significant variations in the way critters live and die.

If you're so inclined, you can change the value of dead_are_food to a nonzero value. This makes dead critters turn into food, and radically changes the environment.

After initialization, the program will enter a four-part loop containing: reproduction (which also includes dying); movement (which includes eating); food

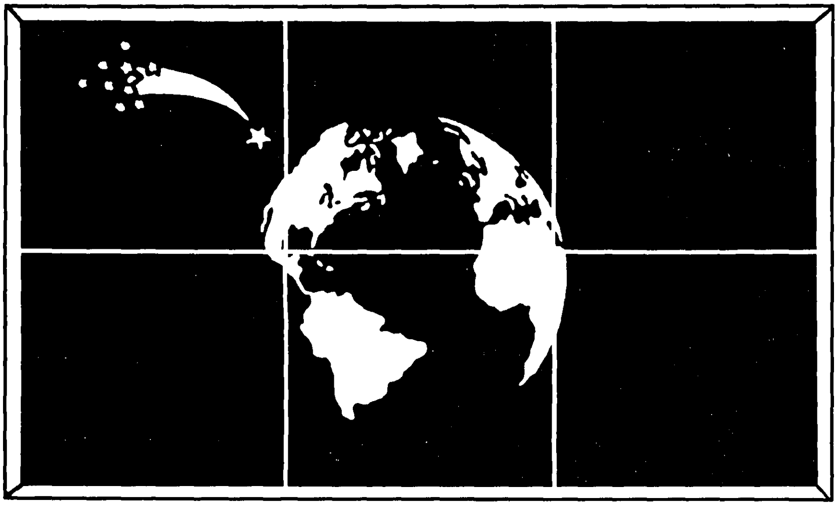
Continued on page 55

Figure 1—Critters Code

```
/*  Program:   Critters
    Version:   1.00    02-Jun-1989
    Language:  Microsoft QuickC v2.0
    Purpose:   Simulates simple life forms which evolve

    Written by Scott Robert Ladd. No rights reserved. */

#include "graph.h"
#include "stddef.h"
#include "stdio.h"
#include "stdlib.h"
#include "string.h"
#include "conio.h"
#include "time.h"

/* structure which defines a critter */
struct critter
    {
    int move_gene[4];
    int energy_level;
    int posx;
    int posy;
    long int age;
    struct critter * next;
    struct critter * prev;
    };

/* constants which define the program environment */
const int start_crit   =   25; /* initial number of critters */
const int start_food   = 2500; /* initial food supply */
const int max_x        =  600; /* maximum x dimension of "world" */
const int max_y        =  300; /*    "    y     "      "    "    */
const int max_energy   = 1000; /* max energy a critter can have */
const int max_food     = 4000; /* max amount of food in existence */
const int max_age      =  200; /* max age w/o reproducing */
const int max_mutant   =    4; /* max magnitude of a mutation */
const int food_value   =   25; /* energy value of a piece of food */
const int move_cost    =    1; /* energy cost to move once */
const int repro_energy =  500; /* min energy for critter to repr */
const int repro_age    =  100; /* min age for critter to reproduce */
const int start_energy =  500; /* initial energy level for critters */
const int food_growth  =    2; /* amt food created each generation */

/* boolean values for program behavior */
const int old_age_kills = 1; /* true */
const int mutate_parent = 1; /* true */
const int dead_are_food = 0; /* false */

/* colors of critters and food */
const short food_color =  7; /* grey */
const short crit_color = 15; /* white */

/* movement definition table */
const int delta_table[4][2] = { {0, -3}, {+3, 0,}, {0, +3}, {-3, 0} };

/* root and tail of critter linked list */
struct critter * first_critter = NULL;
struct critter * last_critter  = NULL;

/* informational display counts */
long int critter_count = 0;
long int food_supply   = 0;
long int generation    = 0;

/* macro to get a random value between 1 and x */
#define randval(x) ((rand() % x) + 1)

/* function prototypes */
void main(void);        /* program control and logic */
void setup(void);       /* initial set-up of program */
void reproduce(void);   /* critter reproduction/death cycle */
void movement(void);    /* critter motion */
void make_food(void);   /* adds new food to world */
void show_stats(void);  /* displays current information on the world */
int  cancel(void);      /* checks for a keyboard entry */

void main(void)
    {
    setup();
    do  {
        reproduce();
        movement();
```

Continued on page 54

You asked to see more...



And **Tele**™ *created windows to your world*

Berry Computers presents The Tele Toolkit — a complete Operating Systems Kernel

Fast pop-up windows. No change in programming model. Transparent use. Those are just some of the advantages Tele windows will give you.

Tele windows work like this. Your application program may create any number of virtual display screens. Physically, a virtual display is simply an array in memory. **Virtual displays may have any number of rows and columns - whatever is appropriate for your application.**

Once created, your program need only select a particular window and then write to it using operations similar to those provided by either the BIOS or MS-DOS for console output. **Tele windows are compatible with graphics.**

Operations on virtual windows are fast, but not visible. To make output visible, rectangles are mapped from your virtual displays to the physical display. Your application may define how the mapping is done, or you may do it from a higher level of software. **Windows overlay each other in the order they are mapped,** unless marked to stay on top (as a diagnostic screen might require).

The mappings are processed periodically to make your virtual displays visible. Tele updates the screen only as fast as the eye can perceive. **A standard PC clone using Tele windows is 5 times faster than when using the BIOS and BIOS doesn't do windows.**

And there are no software side effects. In a multitasking environment **all Tele windows may be active simultaneously.** "Foreground" and "background" only describe how your windows are overlayed; you control task priority with the scheduler.

Thorough documentation and complete C and assembly source code is provided, including demonstration programs. Tele windows is the CD component of the Tele Operating System Toolkit. **Tele works on all PC's, based on any member of the 8086 family.**

Demo Diskette	**$ 5**	(refundable with purchase)
SK system kernel	**$50**	(multitasking)
CD console display	**$40**	(windows, requires SK)
FS file system	**$40**	(MS-DOS media, requires SK)
OS core	**$130**	(SK, CD, and FS)

Telephone support is freely available.

The Tele Toolkit is available from:

Crosby Associates
P.O. Box 248
Sutter Creek, California
95685

CALL NOW TO ORDER:
(209) 267-0362 FAX (209) 267-0115

Visa, Mastercard, American Express & Discover Card accepted.

MS-DOS is a trademark of Microsoft Corporation.

Reader Service Number 147

```
        make_food();
        show_stats();
        }
    while (!cancel());
    _setvideomode(_DEFAULTMODE);
    }

void setup(void)
    {
    int i, j, res, x, y, pval, working;
    unsigned seed;
    struct critter * temp_critter;

    /* initialize random number generator */
    seed = (unsigned)time(NULL);
    srand(seed);
    /* initialize graphics to EGA 16 color mode */
    res = _setvideomode(_ERESCOLOR);
    if (!res)
        {
        _setvideomode(_DEFAULTMODE);
        exit(1);
        }
    /* initialize linked list of critters */
    for (i = 0; i < start_crit; ++i)
        {
        temp_critter = malloc(sizeof(struct critter));
        if (first_critter == NULL)
            {
            temp_critter->prev = NULL;
            first_critter = temp_critter;
            }
        else
            {
            last_critter->next = temp_critter;
            temp_critter->prev = last_critter;
            }
        last_critter = temp_critter;
        temp_critter->next = NULL;
        temp_critter->energy_level = start_energy;
        for (j = 0; j < 4; ++j)
            temp_critter->move_gene[j] = j + 1;
        temp_critter->posx = randval(max_x);
        temp_critter->posy = randval(max_y);
        temp_critter->age = 0;
        }
    /* place initial food supply */
    _setcolor(food_color);
    for (i = 0; i < start_food; ++i)
        {
        working = 1;
        do  {
            x = randval(max_x);
            y = randval(max_y);
            pval = _getpixel(x,y);
            if (pval != food_color)
                {
                _setpixel(x,y);
                working = 0;
                }
            ++food_supply;
            }
        while (working);
        }


    /* set information counts */
    critter_count = start_crit;
    food_supply   = start_food;
    }

void reproduce(void)
    {
    int i, x, y;
    struct critter * temp_critter;
    struct critter * new_critter;

    ++generation;
    temp_critter = first_critter;
    while (temp_critter != NULL)
        {
        /* the critter gets older */
        ++temp_critter->age;
```

```
        /* if critter can reproduce, it does */
        if ((temp_critter->energy_level >= repro_energy)
        && (temp_critter->age >= repro_age))
            {
            /* allocate a new critter */
            new_critter = malloc(sizeof(struct critter));
            if (new_critter == NULL)
                {
                _setvideomode(_DEFAULTMODE);
                exit(3);
                }
            /* add new critter to end of critter list */
            new_critter->prev  = last_critter;
            new_critter->next  = NULL;
            last_critter->next = new_critter;
            last_critter       = new_critter;
            /* new energy levels for parent and child */
            temp_critter->energy_level =
              temp_critter->energy_level/2;
            new_critter->energy_level  =
              temp_critter->energy_level;
            /* inherit the old genes */
            for (i = 0; i <= 3; ++i)
                 new_critter->move_gene[i] =
                   temp_critter->move_gene[i];
            /* select gene which is mutatated! */
            for (i = randval(4) - 1; i <= 3; ++i)
                 new_critter->move_gene[i] +=
                   randval(max_mutant);
            /* if parent mutates... */
            if (mutate_parent)
                {
                for (i = randval(4) - 1; i <= 3; ++i)
                     temp_critter->move_gene[i] +=
                       randval(max_mutant);
                }
            /* put new critter in same place as parent */
            new_critter->posx = temp_critter->posx;
            new_critter->posy = temp_critter->posy;
            ++critter_count;
            new_critter->age  = 0;
            temp_critter->age = 0;
            }
        /* check to see if the critter dies */
        else if ((temp_critter->energy_level == 0)
             || ((old_age_kills)
             && (temp_critter->age > max_age)))
            {
            --critter_count;
            /* if all critters are dead, end program */
            if (critter_count <= 0)
                {
                show_stats();
                while (!cancel())
                    /* empty */ ;
                _setvideomode(_DEFAULTMODE);
                exit(2);
                }
            /* delete character from linked list */
            if (temp_critter->prev == NULL)
                {
                first_critter = temp_critter->next;
                first_critter->prev = NULL;
                }
            else if (temp_critter->next == NULL)
                {
                last_critter = temp_critter->prev;
                last_critter->next = NULL;
                }
            else
                {
                temp_critter->prev->next=
                    temp_critter->next;
                temp_critter->next->prev=
                    temp_critter->prev;
                }
            /* remove body of dead critter */
            for (x = temp_critter->posx - 1;
                 x <= temp_critter->posx + 1; ++x)
                {
                for (y = temp_critter->posy - 1;
                     y<=temp_critter->posy+1; ++y)
                    {
```

Continued on next page

```
                    if (dead_are_food)
                        _setcolor(food_color);
                    else
                        _setcolor(0);
                    _setpixel(x,y);
                    }
                }
            if (dead_are_food)
                food_supply += 9;
            free(temp_critter);
            }
        temp_critter = temp_critter->next;
        }
    }

void movement(void)
    {
    int i, x, y;
    int pval, move;
    struct critter * temp_critter;

    temp_critter = first_critter;
    while (temp_critter != NULL)
        {
        /* erase critter from old position */
        for (x = temp_critter->posx - 1;
             x <= temp_critter->posx + 1; ++x)
            {
            for (y = temp_critter->posy - 1;
                 y <= temp_critter->posy + 1; ++y)
                {
                _setcolor(0);
                _setpixel(x,y);
                }
            }
        /* select movement */
        move = randval(temp_critter->move_gene[3]);
        for (i=0; move>temp_critter->move_gene[i]; ++i)
            /*empty*/;

        /* adjust critters position */
        temp_critter->posx += delta_table[i][0];
        temp_critter->posy += delta_table[i][1];
        /* wrap around edges of the world if needed */
        if (temp_critter->posx < 1)
            temp_critter->posx = max_x;
        if (temp_critter->posy < 1)
            temp_critter->posy = max_y;
        if (temp_critter->posx > max_x)
            temp_critter->posx = 1;
        if (temp_critter->posy > max_y)
            temp_critter->posy = 1;
        /* assess movement cost */
        temp_critter->energy_level -= move_cost;
        /* redraw critter at new position */
        for (x = temp_critter->posx - 1;
             x <= temp_critter->posx + 1; ++x)
            {
            for (y = temp_critter->posy - 1;
                 y <= temp_critter->posy + 1; ++y)
                {
                _setcolor(crit_color);
                pval = _setpixel(x,y);
                /* if pixel covered is food, eat it! */
                if (pval == food_color)
                  {
                  if(temp_critter->energy_level<
                    max_energy)
                     temp_critter->energy_level+=
                       food_value;
                  --food_supply;
                  }
                }
            }
        temp_critter = temp_critter->next;
        }
    }

void make_food(void)
    {
    int i, x, y, pval, working;

    if (food_supply >= max_food)
```

Continued on next page

growth; and statistics calculations. Note that the critter's world is circular; when they move off one side of the map, they reappear on the opposite side.

As you watch the program run, you'll notice that the behavior of the bugs changes. The initial set of critters jitter around without much direction. Later, you can see certain types of critters—called cruisers—which tend to travel in a single direction. Often, a group of cruisers (critter packs) will form; they'll move across the screen, devouring all the food in their path.

The size of the population will ebb and flow. Sometimes it will grow to large numbers, and at other times it will drop to very few members. It's interesting to see how the life cycles work, since significant die-offs often generate new populations based on the genetics of the few survivors of the earlier colony. Eventually, the population will die out completely, although this usually happens only after several thousand generations.

The longest-lived group I've seen yet made 29,384 generations before giving up. Changing the factors listed in the beginning of CRITTERS.C can have a dramatic affect on the way the population evolves.

This project is by no means finished. I'm adding genetics to critters for the ability to sense food, to change their metabolic rate (i.e., a move costs less), and to change other behaviors.

What would happen if critters suddenly turned carnivorous? (Depends partly on what constitutes meat, I suppose.) Does changing the distribution of food make for different behaviors in different parts of the critter's world? And, of course, the program needs a setup mode with which you change the program constants at run-time, instead of having to recompile every time.

The possibilities are endless. As I add new features to the program, you'll hear about them here in this column.

NEWS AND REVIEWS

There isn't much going on in C these days. The biggest news is in the Pascal arena. While this column ostensibly covers C, the battle over Pascal between Microsoft and Borland has several implications for the future of those companies' C language products.

Microsoft has entered Borland's territory with QuickPascal, a clone of Borland Turbo Pascal 5.0 with the addition

Continued from page 55

```
        return;
    _setcolor(food_color);
    /* drop pieces of food in blank spots */
    for (i = 0; i < food_growth; ++i)
        {
        working = 1;
        do  {
            x = randval(max_x);
            y = randval(max_y);
            pval = _getpixel(x,y);
            if ((pval != crit_color)
               && (pval != food_color))
                {
                _setpixel(x,y);
                working = 0;
                }
            ++food_supply;
            }
        while (working);
        }
    }

void show_stats(void)
    {
    char buffer[78];

    _settextcolor(crit_color);
    sprintf(buffer,"generation: %6ld, critters: %5ld,\
            food: %5ld", generation, critter_count,
            food_supply);
    _settextposition(24,1);
    _outtext(buffer);
    }

int cancel(void)
    {
    if (kbhit())
        {
        if (!getch()) getch();
        return 1;
        }
    return 0;
    }
```

◆ ◆ ◆

of compatibility with Apple & Wirth's object-oriented Object Pascal language. Borland, meanwhile, released Turbo Pascal 5.5, which uses Borland's own support for objects.

Before I make any qualitative statements, let me make a disclaimer: I am writing a book called *Learning Object-Oriented Programming with QuickPascal* for Microsoft Press. Some people may immediately discount my opinions on Pascal (and possibly other products) because of this. However, I like to make my affiliations public so that people know where I'm coming from.

QuickPascal is a good, solid clone of Turbo Pascal. It compiled every Turbo Pascal program I've tried very quickly (of course). I like its environment better than Borland's, since it allows you to edit multiple files simultaneously. Also, QuickPascal has the capability to animate your program, showing the source code you are executing while the program runs in slow motion.

QuickPascal is cheaper than Turbo Pascal; it can be found in some places for under $50. However, Microsoft has continued their recent practice of putting much of the documentation for Quick products into on-line help, rather than paper manuals.

If it weren't for the object-oriented extensions, I don't think Microsoft did enough extra to lure many old-timers away from Turbo Pascal. But, of course, it is the object-oriented extensions which make these products interesting.

Microsoft followed the existing standard for Object Pascal, while Borland created their own object-oriented dialect of Pascal. Both programs are very similar in capability and function—in spite of what the propaganda says.

Borland has made some changes which enhance the speed of object-oriented programs.

Turbo Pascal 5.5 also shows some signs of incomplete thinking by Borland; their implementation of constructors and destructors (borrowed from C++) is weak. QuickPascal's object-oriented extensions are easy to learn and follow a standard used in Macintosh Pascals, but they produce programs which are 5% slower than their Borland counterparts.

Both companies are looking to the future; they are developing enhanced versions of the programs as you read this. I see a similar fight occurring in regard to C.

Microsoft has publicly announced plans to develop an object-oriented C by the first quarter of 1990; they will probably be implementing an extended C++.

Borland is being coy, saying that they are moving toward object-oriented C, but that they haven't made a decision on how to go about it. While Borland can define their own standards in the limited world of PC Pascal, I don't think they can do the same for the larger universe of C.

Resources

I highly recommend A.K. Dewdney's column (mentioned above) in *Scientific American*. If you're interested in expanding your mind, this column will do it for you.

Editor's note: Of course, if you're not interested in expanding your mind, you can continue reading Micro C.

I hope Dewdney will put out a collection of his columns for those people who are not *Scientific American* subscribers. Heck, I recommend *Scientific American* all by itself; it's a solid scientific journal for the educated non-scientist.

I'm Outa Here....

I'm putting together a collection of computer-based life programs which you can soon find on the Micro C BBS and issue disk. Included will be programs for the Borland, WATCOM, and Zortech compilers. In addition, I'll be putting in a new version of CRITTERS, which will incorporate some of the extensions I mentioned earlier.

Please feel free to take these programs and experiment with them; I'd love to hear back about your observations. As always, write me at the above address, or send to me via FidoNet NETMAIL at 1:104/45.2.

◆ ◆ ◆

86
WORLD

Raiders Of The Lost Editor

By Laine Stump
% Redhouse Press
Merkez PK 142
34432 Sirkeci
Istanbul, Turkey

Laine discovers Brief; but that doesn't mean he's short, just Brief as only Laine can be Brief.

I guess just about everybody knows of my lifelong search for the perfect programming editor. Several different editors, including my own (which my brother and I wrote about five or six years ago), have led me down the road. Every time I run into a new editor that has some features I've been looking for, it always seems to have just enough minor (or major) flaws to justify not switching over completely.

Because of this dissatisfaction with any one editor, my machine usually contains three or four of the beasties.

For example, if I need to look at several files at once, I use Logitech's Point. It has multiple, overlapping windows and can support up to 50-line displays. If I need to do some massive, repetitive changes to a file, or edit a single file with very quick response, I'll call up Express (my homegrown editor), for its small size, speed, and simple keystroke macros. On the other hand, if I need to cut and paste columns of text, I haul out Microsoft Word (the other two can't do this.)

All this schizophrenia causes some confusion at 5:30 in the morning. For example, Point uses F5 to begin marking a block of text and F6 to finish. Word uses F6 to begin and finish marking a block, while F5 turns replace mode on/off.

I've managed to get rid of the glaring differences (like cursor movement keys) by writing a TSR program that traps keyboard input and translates my favorite keys into the appropriate ones for a particular program. But it's still a royal pain in the butt to keep jumping from one editor to another, doing twenty seconds of work, saving, exiting, and then jumping back....

Enter Brief

During my last two winters at PC Tech, I sat across a desk from Earl Hinrichs. Whenever my eyes got screwed up from staring too hard at my own screen, I would look up at Earl, bapping away at his keyboard like there was no tomorrow. Windows flashing by in blinks, file menus shooting out and disappearing a split second later. He was using Brief (Basic Reconfigurable Interactive Editing Facility).

We talked about editors a few times. After all, we both spend probably more than half our waking lives working with an editor. I would tell him a few things about my current editor, and he would reply with a counter point about similar capabilities in Brief. It was like "Dueling Cursor Keys," and Earl was the local, furiously picking away on his banjo while my fingers became impossibly tangled in my guitar strings.

Even so, the first year I wasn't intrigued enough to spend any time learning yet another editor. After all, I had just spent a week or so figuring out how to use Point. It did most everything I wanted (I thought); Express covered the rest (I thought). I let it slide.

This spring I was finally ready for a change. After using Point for a year, I decided that it was just too slow. Even on a 10 MHz AT, if I had multiple views of a window on the screen and hadn't saved in a while, I could type faster than it could accept characters. It looked nice, cursor movement was fast, and the price was right (it came free with Logitech Modula and older Logitech Mice). But I was ready for something new.

Also, I had decided to get a new PC Tech Mono II board and double page display. This display can support at least 160 columns of text. While no other editor I knew of could handle that size of display, the ads for Brief claimed that it could drive displays up to 255 columns by 127 lines. That was enough for me to dig in and try it.

What Is It?

How about a brief overview? Brief is a text editor with a very powerful macro language. It can display multiple windows on the screen at one time, each showing a different (or possibly the same) file. It is especially well suited for programming, with things like automatic indenting and automatic compilation and error display (it can make any compiler into a "Turbo" compiler).

Although Brief has tons of features and is very fast, its most important aspect is the macro language. This is not just a simple keystroke recorder. The Brief macro language is full enough that many parts of Brief itself were written as Brief macros. You can write your own macros to extend Brief, or modify existing parts of Brief. (It includes the source code to the parts of Brief written in the macro language.)

> Although Brief has tons of features and is very fast, its most important aspect is the macro language.

Review?

I could go through a standard magazine review form and tell you how many seconds it takes to save the "standard" file, what the maximum line length is, how many windows it can display, and all that meaningless bull. I will give you a bit of that, in fact. What I really want to do is show you a few macros that might give you a feel for the flavor of Brief, and maybe lead you to thoughts about some of the possible uses of Brief and its macro language over and above just plain text editing.

But first, a few praises and whines.

Intelligent Indenting

I always thought that an intelligent indenting editor would be either too intelligent or too stupid for me. Of course, nearly every editor these days has an indent mode where hitting "enter" places you on the next line at the position of the first letter on the previous line. Somehow it just seemed like a program would either not recognize every case, or would force my programming style into something I didn't like.

Brief's intelligent indenting package for the C language mildly surprised me. There was only one case where it wouldn't match my own preferences for indenting. It was able to recognize all the constructs I gave it and placed me in the proper position when I hit enter.

For example, when I typed:

```
if (junk==25) <enter>
   _
```

it automatically placed the cursor at the next tab stop in from the position of "if" on the previous line (as shown by the underscore). If I then typed a single statement, followed by a semicolon:

```
if (junk==25)
   ct++; <enter>
_
```

the cursor would step back out one tab stop for the next line. If, on the other hand, I typed a bracket, followed by enter, it would go down to the next line, leaving the indent at the same place (until I typed the closing brace).

The smart indenting package (which is written entirely in the Brief macro language) can recognize the four most common forms of C indenting style:

```
if (junk==25)
   {
   ...
   }

if (junk==25) {
   ...
   }

if (junk==25)
{
   ...
}

if (junk==25) {
   ...
}
```

It handled the first two automatically and supported the third and fourth styles if you set an environment variable in the DOS environment.

The one construct that it would not handle properly, by the way, was the definition of a function. I like to have the braces that start and end a function (and all the lines in the function) indented one tab from the left margin:

```
void twist(int schillings)
   {
   ...
   }   /* end of twist */
```

When I typed enter after the closing parenthesis of the function definition, the cursor went to the first column of the next line. I typed the brace there and typed enter, expecting the cursor to go to the first column again. Instead it went in one tab stop:

```
void twist(int schillings)
{
   _
```

Although the position of the opening and closing braces of if() while(), etc., is selectable, the position of the braces around a function is not.

Another feature that goes along with smart indenting is the special function of tab and shift-tab. If you select a block of text, tab and shift-tab will move all the selected lines in or out by one level of indent, rather than moving just the current line.

Unfortunately for programmers in other languages, smart indenting is only available for C (and for dBase if you buy a package called dBrief, sold by Global Technologies). But Brief does have "regular" indenting (the kind found in most other editors). You could always use the C smart indenting package as an example to write your own indenting package for another language. Maybe you could even give me a copy. Or sell it. Or both.

Column Blocks

This is another nice feature that the Brief macro language (source code included, of course) implemented. When you mark a block of text using a "Column Mark" instead of a regular mark, the Copy, Cut, and Delete macros recognize it automatically. Then, instead of deleting everything between the start mark and end mark, they go into a loop that appends the proper portions of each line into the "scrap buffer," separated by a special marker.

A subsequent Insert command will recognize that the scrap contains a columnar block and go through a similar loop that inserts each line fragment at the column of the cursor, starting at the current line and continuing down until using up all the line fragments.

I was doing some work with Xenix mapchan (translates from one character set standard to another on a specified I/O channel) last week. Columnar cut and paste saved me at least half an hour of retyping columnar data in a different order.

(No, I don't have Brief for Xenix. I do most of my work under DOS and then use the Xenix doscp command to move it onto the Xenix partition).

Other Nice Things

Brief does all kinds of other nice things, too. So many I can't mention them all here. A few examples: Brief can swap itself out to EMS, RAMDisk, or disk when you escape to DOS to execute a DOS command. This frees up all but 3K of your RAM for use by the command. It also allows you to specify which directory to put backup files in so that, for example, all backups go into a single directory for ease of erasing.

Finally, you can specify a macro package to load for files with different extensions (that's how it handles the C indenting). It does this by setting an environment variable called BPACKAGES. The keyboard assignments of these macros are kept local to the window that the file is in.

For example, you can have a .C file in one window which is using smart indenting, while a .DOC file in another window uses word wrap.

Bad Things (BOO!! HISSSSS!!)

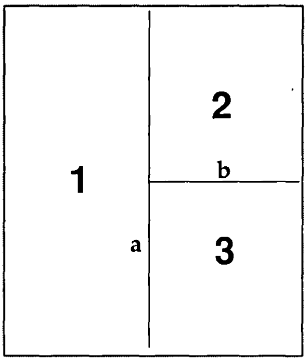
The main deficiency of Brief that kept me away for such a long time was its tiled windows (a window not allowed to partially conceal another window). Even though I know that overlapped windows don't magically make the screen capable of holding more characters, it does *seem* to do that. When you add to this the fact that you can't move a window border which touches on more than two windows (see Figure 1), the windowing capability of Brief appears seriously brain damaged.

I can kind of understand the decision to make windows tiled instead of overlapped; it makes for much less complexity in the screen update algorithms. But not allowing changes to the position of a window border touched by more than two windows looks to me like just plain laziness. Surely it shouldn't take more than a simple loop cycling through the windows, readjusting.

Brief does have overlapped windows, by the way (I even use them in one of the examples below), but they are very restricted. For example, once you open an overlapped window, you can't switch to any other window until you close it. Mostly, it uses overlapped windows for menu displays.

Another window complaint is that it's impossible to retrieve enough information about the windows to reconstruct a display successfully once it's torn down. This would be very useful for the included "restore" macro package, which causes Brief to reload all the files you were editing during the previous session automatically.

Figure 1—Window Borders in Brief.



This Brief screen has 3 windows and 2 edges (a & b). Brief allows moving edge b (thus resizing windows 2 and 3), but won't allow you to move edge a. The only way to move a is to first delete b.

◆ ◆ ◆

No Mouse Support

Not only is mouse support not included with Brief, it would be very difficult to do many of the things I like having a mouse for. An example is scrolling and searching.

In Point, clicking the left or right button on the left border of a window scrolls forward or backward through the file. Clicking on the right border searches for the previous or next occurrence of the current selection.

Since the right side of a Brief window is also the left side of the window to its right, this scheme would be impossible. There is no way to decide whether the desire is to scroll the right hand window, or search in the left hand window.

Maybe this is just my problem. I'm trying to make Brief be everything I've worked with. Maybe there's a much more convenient way of searching that I haven't thought of yet.

A Fixable Problem

The final complaint in my notes is the inability of the BPACKAGES facility to recognize wildcards (t??, 01?) in filename extensions. I encountered this problem when trying to set up BPACKAGES to turn on the word processing package, wp, automatically whenever I edited a text file.

The problem is that I use the extension of text filenames to indicate a sequence number. For example, letters to my brother are given the names CECIL.001, CECIL.002, etc. I tried to make BPACKAGES recognize this with the command:

```
SET BPACKAGES=0??:wp
```

which *should* mean "whenever you see a file extension with a 0 at the beginning, load the wp macro package." Unfortunately, it didn't work. If I wanted to, I could do it by specifying every possible extension (all 999 of them), but....

The best solution is to rewrite the BPACKAGES stuff to recognize wildcards. Even better would be to rewrite BPACKAGES to accept regular expressions for file extensions. Then I could do this:

```
SET BPACKAGES=[0-9]??:wp
```

That's one nice thing about Brief. If a facility isn't there, the tools to implement it are.

Figure 2—MYSTUFF.M

```
;**
;**    mystuff.m - an example of a simple brief macro to
;**                add a couple key definitions to brief.
;**
;**    Laine Stump 6/10/89
;**

(macro _init
  (
    (autoload "wp" "reform") ;** tells brief where reform macro is

    ;** first some simple redefinitions of existing commands
    (assign_to_key "<Ctrl-O>" "reform")    ;** reformat paragraph
    (assign_to_key "<Alt-D>"  "dos")       ;** escape to dos shell
    (assign_to_key "<Right>"  "next_char") ;** go to next char in buf
    (assign_to_key "<Left>"   "prev_char") ;** go to prev char in buf

    ;** now define a key to call our own new macro
    (assign_to_key "<Alt-Z>"  "zoom_window")

  )
)


(extern display_file_name)

(macro zoom_window   ;** zoom the current window to fill the screen
  (
    (int lines cols zbuffer)

    (inq_screen_size lines cols)      ;** see how big we can get
    (= zbuffer (inq_buffer))          ;** this is what's in the window
    (create_window 0 (- lines 3) (- cols 2) 0
                  "ZOOM MODE (alt+Z for WINDOW MODE)")
    (set_buffer    zbuffer)           ;** make the window and attach
    (attach_buffer zbuffer)           ;** current buffer
    (refresh)                         ;** update the display
    (assign_to_key "<Alt-Z>" "exit")  ;** provide a sure way out
    (display_file_name)
    (process)                         ;** recursively call brief
    (delete_window)                   ;** remove the zoom window
    (set_buffer    zbuffer)           ;** reattach original window
    (attach_buffer zbuffer)           ;** in case we switched
    (display_file_name)
    (assign_to_key "<Alt-Z>" "zoom_window")
  )
)   ;** end of zoom_window
```

◆ ◆ ◆

Support

Since I seldom ask for support from Istanbul, I don't know anything firsthand about support from Solution Systems. The one thing I do know is that Earl sent several letters to them asking for information on upgrading to version 2.1, and they never replied. I guess they didn't want his money.

However, they do have an 800 number which reportedly gives toll-free support (in case your Briefs need extra support).

Price

The final bad thing about Brief is the price. $195!??!? Of course, if you believe what Patrick O. Conley said in "On Your Own" in issue #48, it makes sense. Solution Systems is just keeping out the geeks and nonserious users by pricing their product above the "Well, hyuck, that looks like a nifty program. Maybe I'll just buy it even though I'm stupid. I can always call them up when I can't figure out how to save a file."

Aside from the appearance that Solution Systems advertises way too much for its own good (everywhere except *Micro C*), giving them undoubtedly *huge* operating expenses, I think the price of Brief sends a message to prospective buyers: "Think before you buy!"

Brief is not for the casual Space Invaders player. Not that it's difficult to use. While Brief can do many things a $30 editor cannot do, there are many things that a $30 editor could do just as well as Brief. If you only need the $30 features, it would be just plain stupidity to buy Brief and use it as a $30 editor.

If you're a *real* professional, you can justify the expense. Especially if you buy it from Programmer's Connection or Programmer's Paradise, or some other discount software place.

Taste Test

So, enough praising and whining. Now I'll show you a few little chunks of code that I wrote in the Brief macro language. They should give you an idea of some of the possibilities. Remember though: I'm a Brief neophyte. I wrote my first Brief program this morning. What I can do is *nothing* compared to what is possible.

Figure 3—SQUARES.M

```
(macro squares    ;** add the square of the first n integers
   (
      (int last ct total)

      (get_parm 0 last)
      (= total 0)
      (= ct 1)
      (while (<= ct last)
         (
            (+= total (* ct ct))
            (++ ct)
         )
      )
      (returns total)
   )
)

(macro square_10  ;** an example of calling squares
   (
      (message "The sum of first ten squares is: %d" (squares 10))
   )
)
```

◆ ◆ ◆

Reassigning Keys

Probably the first thing anyone will want to do when they get Brief is to change the command key assignments to match what they're accustomed to. The program MYSTUFF.M in Figure 2 shows how to do this. To begin using this program, just type it in with Brief, then type alt+F10 to compile and load it.

In subsequent sessions, start Brief with the "-mmystuff" option. The -m tells Brief to load the macro file mystuff. When loading a macro, Brief searches for a function called _init and, if found, executes it.

You're probably wondering about all those parentheses in there. The Brief language has many similarities to LISP. One similarity is that every statement is surrounded by parentheses (instead of being separated by commas, like C).

It also uses prefix notation instead of the standard infix. This means that if you want to add 3 and 4, you use the statement:

```
(+ 3 4)
```

If you want to call the routine "square" on the result of 3*4+2 and put the result of square into the variable "result, " you would use:

```
(= result (square (+ (* 3 4) 2)))
```

Kind of like a backwards HP calculator.

A Zoom Window

Also in Figure 2 is an assignment of the alt+Z key combination to a macro called zoom_window. This macro isn't included with Brief; I wrote it myself.

I wrote it to mimic a command in Point that zooms the current window to fill the entire screen until I press alt+Z again, when it restores the original state of windows.

zoom_window uses a Brief overlapped window to display a file over the top of the normal windows on the screen. I was forced to do it this way because, in order to make a tiled window fill the screen, I would have to tear down the current window structure of the screen. I have already mentioned that there is no way to get enough information about windows on the screen to reconstruct a window configuration once it's destroyed.

First, zoom_window gets the screen size. It uses that information to make the zoom window as large as possible. Next it asks which buffer (file, in practical terms) is in the current window. It then creates a new window to fill the entire screen and the "current" buffer is attached to the new window.

After setting up the window to edit, but before editing, I reassign the alt+Z key to prevent attempting to zoom an already zoomed window. This wouldn't be dangerous (you can create up to three overlapped windows simultaneously, although the only way to switch to a previous window is to delete the current one), but there's no reason to zoom an already zoomed window.

To allow entering keystrokes and editing the file in the zoomed window, I call process, which is a recursive call to Brief itself. process inputs and processes keystrokes until it calls the macro, exit.

After process is finished, it's time to delete the zoom window, then reattach the original buffer to the original window. Finally, I reassign the zoom key.

There are some problems with zoom_macro that I haven't bothered to figure out. For example, if you move the cursor while zoomed, the cursor in the window display doesn't reflect that movement when you return. Also, the word wrap package seems to get lost now and then if I switch the displaying file in the zoom window.

The solution to the first problem is to inquire about cursor position in the zoomed window before deleting it, and set the tiled window to the same location. The solution to losing word wrap when switching files in the zoomed window is to set up a "local keyboard" for the zoomed window which has the "switch file" keys disabled. (Each window can have its own set of commands.)

Those are the famous "exercises for the reader," though. I may fix them later, but meanwhile you can fix them for yourself.

Squares

The final program, Figure 3, is just a little something to show that the Brief macro language really is a general purpose programming language. The function squares accepts an integer as a parameter, and returns the sum of the squares of all numbers from 1 to that number, inclusive. square_10 shows how you would call squares.

Again you'll notice the strange parenthesis syntax of the language. You'll also notice that the definition of squares doesn't seem to indicate anywhere that it accepts a parameter. The function get_parm handles that.

By not forcing you to hardcode the parameter list for a function, Brief allows you to have the same function accept violently different parameter lists based on decisions made in the function. It also allows the function to either supply a default value, or prompt the user for the value, when a parameter isn't given.

Figure 4—Corrected NewIntTable

```
IntInfo   STRUC
  IntNo      DB   ?
  IntVector  DW   ?
IntInfo   ENDS
;
;  table of new interrupt vectors to install

NewIntTable \
  IntInfo    < 0,Div0Trap  >
  IntInfo    < 6,InvOpTrap >    ;Intel Invalid Opcode exception
  IntInfo    < 0,0 >            ;END OF TABLE
```

◆ ◆ ◆

Onward

You can do many other things using the Brief macro programming language. There's a complete macro package included for creating menus, as well as integer to string and string to integer routines.

There are deficiencies, though. For example, arrays must be emulated by using a text buffer, and complex data types cannot be constructed. Also, there is no floating point type (big deal!).

Still, you can easily solve many problems having to do with manipulating large amounts of text by using Brief macro programs. Most important, you can modify your editing environment to be very close to "exactly how I want it."

Epilogue

I began using Brief two months ago. At first I just used it when I was bored. Later, as I came up against things I couldn't do with my other editors, I used Brief more and more. As of today, I haven't used any editor except Brief in five days. A couple of times I accidentally called up Point from force of habit, and immediately quit.

Even though it isn't "everything I've always asked for in an editor," it's slowly becoming "what I've always asked for in an editor more than anything else."

Corrections

There were a couple of mistakes in the listing of TRAP in my column of issue 48:

Mistake 1 was my fault. I changed the operation of the routine CRASHIT (bottom of page 46), but didn't change the comments at the beginning. Remove the reference to "string to print in CS:SI" and "interrupt # in DX." These two parameters don't exist in this version of TRAP

Mistake 2 is the fault of the stupid programs that Micro C uses for printing listings. (It seems that somehow my listing got sent through Ventura again).

Editor's note: We've Ventura'd every listing Laine has sent us for the past 2 1/2 years. But we forgot to double up one set of <>'s so they'd print—mainly because Laine's article arrived late, as usual.

The problem here is in the table NewIntTable (bottom of page 48). Micro C's program ate the brackets ("<>") around the table entries, and the interrupt number, which is the first member of the structure. A correct printing of NewIntTable is in Figure 4.

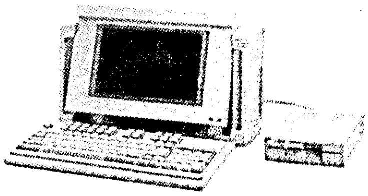
References

Solution Systems
541 Main Street, Suite 410
South Weymouth, MA 02190
(800) 821-2492 Ext. 351
(617) 337-6963 MA and outside U.S.

◆ ◆ ◆

RABBIT

GAS PLASMA PORTABLE



★LCD PORTABLE ALSO AVAILABLE

- 80286-12 CPU
- 640K MEMORY, EXPANDABLE TO 4MB
- DUAL SPEED 6/12 MHZ, 0 WAIT STATE
- LANDMARK 16 MHZ
- 5 SLOTS, EXTERNAL 5¼" 25 PIN DRIVE PORT
- CGA/MGA/EGA 640x400, 4 GRAY SCALE
- 101-KEY ENHANCED KEYBOARD
- GAS PLASMA DISPLAY
- 1 PARALLEL, 1 SERIAL
- 200WT AC 110/220 AUTOSWITCHABLE
- EGA/MGA MONITOR PORT 9 PIN
- HARD DISK/FLOPPY DISK CONTROLLER
- 1.44MB FLOPPY DRIVE AND 20MG HARD DISK 40MS
- 9.45"x16"x8.27" 19.8 LB.
- CARRYING BAG w/SHOULDER STRAP
- ONE YEAR LIMITED WARRANTY

GAS PLASMA $2199

LCD PORTABLE $1729

CHICONY

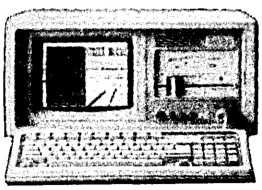
LAP-TOP

- 80286-16 CPU (0 WAIT STATE)
- NEAT, TURBO PAGE MODE SPEED UP TO 21.4 MHZ
- 1MB ON BOARD, EXPANDABLE TO 5MB (EMS V 4.0)
- CGA/MDA/EGA, 640x400, 4 GRAY LARGE GAS PLASMA DISPLAY
- 1.2MB FLOPPY AND 40MB HARD DISK
- "84 + FN" ENHANCED KEYBOARD
- 1P/2S (D-9 and D-25), REAL TIME CLOCK
- 100W, AC 110/220V SWITCHABLE
- 14.8" x 13.4" x 3.7", 15.4 LBS.
- CARRYING BAG w/SHOULDER STRAP
- ONE YEAR WARRANTY

.... $3400

McTEK

EGA PORTABLE



- 286-12 CPU, LANDMARK = 16MHZ
- 640K DRAM (EXPANDABLE TO 4MB)
- 0 WAIT STATE, DUAL SPEED 6/12 MHZ
- DUAL FLOPPY AND HARD DISK CONTROLLER (1=1 INTERLEAVE)
- ONE 1.2MB FLOPPY DRIVE
- ONE 20MB HARD DISK (40MB)
- TRUE OS/2, XENIX, MS DOS AND NOVELL COMPATIBLE
- 80287 MATH CO-PROCESSOR SOCKET
- ENHANCED 101 KEYBOARD w/TACTILE FEELING
- 2 SERIAL PORTS, 1 PARALLEL PORT AND 1 GAME
- LED INDICATORS
- 200W UL POWER SUPPLY
- ONE YEAR WARRANTY

.... $2299

McTEK 286/12MHZ

Assembled & Tested IBM® AT Compatible
MS-DOS® OS/2® Compatible

- ■ 80286 12/6 MHz
- ■ Phoenix BIOS
- ■ 640K of RAM Expandable to 4MB
- ■ 0 Wait State
- ■ 200W Power Supply
- ■ 1.2MB Floppy Drive
- ■ Ports: 1 Serial, 1 Parallel, 1 Game
- ■ Dual Floppy/Dual H.D. Controller
- ■ Monochrome Graphic Card
- ■ 101 Key Enhanced Keyboard
- ■ TTL Monitor 12"
- ■ 20MB Hard Disk (40MS)

Options:................................ Call

$1,199^{00}

McTEK 386/165X

Assembled & Tested IBM® AT Compatible
MS-DOS® OS/2® & UNIX® Compatible

- ■ 80386 16/8 MHz Norton V4.0 SI 17.6
- ■ Phoenix BIOS
- ■ 1MB expandable to 16MB RAM
- ■ 80387 Coprocessor Socket
- ■ 200W Power Supply
- ■ 1.2MB Floppy Drive
- ■ Ports: 1 Serial, 1 Parallel, 1 Game
- ■ Dual Floppy/Dual H.D. Controller
- ■ Monochrome Graphic Card
- ■ 101 Key Enhanced Keyboard
- ■ TTL Monitor 12"
- ■ 20MB Hard Disk (40MS)

Options:............................. Call

$1,495^{00}

McTEK 386-20MHZ

Assembled & Tested IBM® AT Compatible
MS-DOS® OS/2® & UNIX® Compatible

- ■ 80386 20/8 MHz Norton V4.0 SI 22
- ■ Phoenix BIOS
- ■ 1MB expandable to 16MB RAM
- ■ 80387 Coprocessor Socket
- ■ 220W Power Supply
- ■ 1.2MB Floppy Drive
- ■ Ports: 2 Serial, 1 Parallel, 1 Game
- ■ Dual Floppy/Dual H.D. Controller
- ■ Monochrome Graphic Card
- ■ 101 Key Enhanced Keyboard
- ■ TTL Monitor 12"
- ■ 40MB Hard Disk (28MS)

Options:............................. Call

$1,995^{00}

DISK DRIVES

Fujitsu 360k	$69
Fujitsu 1.2MB	$89
Teac	$75
Teac 1.2MB	$89
Teac 3½" 1.4MB	$89
20MB Hard Disk Kit	$279
30MB Hard Disk Kit	$309
KL320 (40MS)	$215
8425 Miniscribe	$249
8438 30MB Miniscribe	$259
3650 40MB Miniscribe	$349
3675 60MB Miniscribe	$379
ST-157 49MB 3½"	$479

PRINTERS

Citizen CD 120	$149
Citizen CD 180	$189
HPLASAR Serial2	$1699
Epson LX-800	$219
Epson LQ-500	$379
Toshiba 321 XL	$519
NEC P2000	$369

Call for prices of other brands

MODEMS

300/1200	$79
2400 external	$139
2400 internal	$99

MONITORS

Samsung amber	$79
Samsung EGA color	$359
Samsung RGB color	$259
NEC Multisync	$559
Sony Multiscan	$619
HGC-compat.mono card	$49
Color graphic card	$49
EGA Paradise 480	$149
VGA Paradise	$279
Genoa Super VGA	$299

MOUSE

Logimouse C7	$69
Logimouse H1 Res	$99

PC/XT

640k TurboMothrbrd	$80
10MHz TurboMothrbrd	$85
Multi I/O w/disk contrir	$59
640k RAM card	$39
2MB Expansion card	$89
RS232 2-port card	$35
4-serial port card	$79
Game I/O card	$15
384k Multifunction card	$69
FCC-app. slide XTcase	$29
150W power supply	$49
XT keyboard	$42
Clock Card	$19
Floppy Controller	$19

PC/AT

McTek 286-20MHz	$359
McTek 286-12	$259
McTek 386-16SX	$429
McTek 386-20MHz	$699
McTek 386-24MHz	$899
Locking slide case	$59
200W power supply	$65
Enhanced keyboard	$59
WD FD/HDC	$129
DTC FDC/HDC 1:1	$189
3MB EMS (0K)	$99

DESKTOP

MISC.

Kingtech CRT Portable Kits: XT/AT (power supply, case keyboard, monitor)	$380/$410
Eprom burner 4-socket	$139
LCD Portable	$799
Plasma Portable Kits	$1499
AC power strips	$15
Diskette file box	$9
Printer or serial cable	$8
Archive Tape Backup 40MB	$339
XT 10MHz 640k 2 Drive System	$759

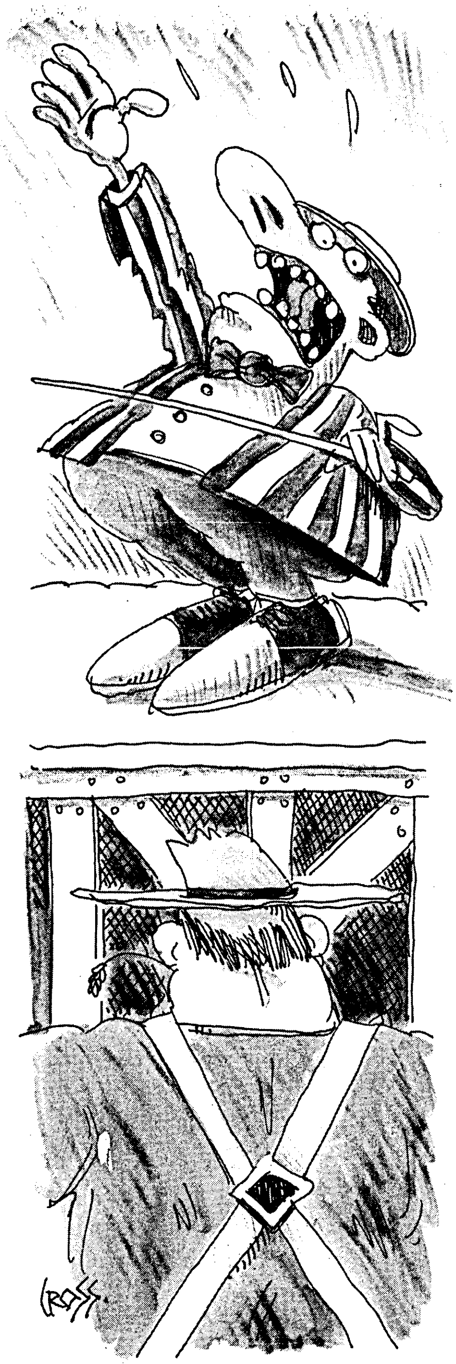
McTek Systems, Inc. • 1521 San Pablo Avenue • Berkeley, CA 94702 • 415-525-5129

Reader Service Number 42

CULTURE
CORNER

By David Thompson
Micro C Staff

The Culture Corner



It was one of those classic county faires, you know, the blare of ferris wheel music (with or without the ferris wheel) and the barkers:

"Hey big fella, win this stuffed panda for your little sweetheart."

"Hey big fella, win this little sweetheart for your stuffed panda."

"Walk away with these prizes, we're giving them away today, we can't take them with us."

"Three balls for just a quarter."

There were also the smells: caramel corn, corn dogs bubbling in rancid oil, fresh sawdust, and Queen Anne's Lace.

Great place to hide from the battles of hi-tech. Grab a snow cone (grape) and feel like 12 again.

"Electronic shredder, get your electronic shredder here. Step right up, this is the kind of file protection you can't be without!" The barker wore a pair of the brightest green pants I'd ever seen.

My first reaction was to flee, after all how many 12-year-olds are into shredders? Probably none, and certainly not more than once. But the adult side of me was curious how this slicker was going to sell a computer product to farm folks. As a small crowd of overalls gathered, I edged in next to a large fellow wearing huge red suspenders.

"This is the latest, greatest invention. Keeps the neighbors, the bank, the wife, the IRS, even the government out of your old data files. With this dramatic breakthrough you don't just throw away information, you shred it into electronic history."

"Lemme see the shavin's," said red suspenders. "The wife's real handy with them jigsaws that has lots of sky."

"That's the best part, there's nothing to throw away. Nothing to haul out to the hogs. This is technology at its finest."

"So what's the use of shredding if you don't get something to do at the dinner table during January? And no mulch? What about the little lady's garden?"

Mr. Green Jeans looked puzzled for a moment. "Let me explain—this removes information from a computer, removes it permanently, by writing new information over what's already there. By the way, are any of you fine folks considering buying your first computer?"

No one raised a finger.

"What kind of information does it write into the computer?" asked suspenders.

"I'm not really sure, just different information, unimportant information. The most unimportant information available. It's information that wouldn't be interesting to the kind of person who might otherwise be interested."

"How does anyone know what's not interesting?"

I noticed that Red Suspenders' drawl had disappeared and the grass stalk he'd been chewing was getting quite short.

"Well take my word for it, no one's been interested yet."

"What does it run on?"

The barker's eyes brightened. "Macs, Apples, Commodores, Ataris, Amigas, and for any of you doing serious computing, the PC family. It'll run on anything."

"How about Sun workstations?"

"What's that?"

"How about my Sun workstations?"

"Who makes them?"

At that point the crowd began to disperse, smiling to themselves. I overheard a couple comments about problems running DOS applications under UNIX.

The last I saw of suspenders, he had his face totally imbedded in a fresh wad of cotton candy. For a moment I had the impression the cotton candy was hiding a huge grin—the grin of an unrepentant 12-year-old.

◆ ◆ ◆

ARE YOU FAMILIAR WITH THE MICRO CORNUCOPIA NETWORK?

The network is a growing, international group of over 20,000 engineers, designers, programmers, publishers, and consultants: practitioners who seek out and use micro computer products and make recommendations about their use. Over half have advanced degrees and nearly three quarters moonlight as computer consultants (in addition to their 9 to 5 jobs in the computer industry).

The network is the developers, hardware and software manufacturers, and retailers who provide the tools: the libraries, systems, and hardware. Our advertisers are key technical resources providing both tools and expertise to this fascinating group. Ads in Micro C are read as thoroughly as the articles.

At the heart of the network is MICRO CORNUCOPIA, the Micro Technical Journal, published bimonthly in a light, often humorous, tell-it-like-it-is style. Our writers are experts, providing in-depth discussions of utilities, libraries, graphic interfaces, clone boards, language interfaces, chaos theory, fractals, micro controllers, and much more. Ninety-five percent of our subscribers keep back issues in their libraries. In addition to subscriptions, Micro C is sold in major newsstands including B. Dalton and Waldenbook.

INTRODUCING THE MICRO CORNUCOPIA TECHNICAL REVIEW SERVICE

Having reviewed thousands of products, services, and startup ventures, we've discovered that most share common challenges. To assist you in your venture we've developed our technical review service. Here's how it works.

STEP 1. You'll start by ordering and completing our questionnaire. It has over 75 important questions about you and your venture. We ask about your product. We ask about your manufacturing, marketing, and administrative tasks and expenses. You'll also receive a personal evaluation form to help you assess your own strengths and weaknesses.

All together, the questions help you explore your market and explore the expertise you and your team bring together. Honest answers to these questions should give you a good perspective about your venture and its chances for success. As an introductory offer, there will be no charge for questionnaires requested prior to October 31, 1989. To receive your free questionnaire write #159 on the reader service card and send it in.

STEP 2. If filling out the questionaire leaves more questions than answers, return it and $150.00 to Micro C. (Add $50.00 for each personal evaluation form beyond the first). We'll review your questionnaire and return our assessment of product feasibility, including places where the idea and/or your group seem particularly strong and where they seem particularly weak. If we see an opportunity for expanding the idea or taking it in an entirely new direction, we'll include that.

Once you receive the written assessment, you're welcome to schedule a one-hour conference call with the evaluators. This will give you a chance to flesh out old ideas or bang around new ones. We don't guarantee to be right (we don't even promise to agree with each other) but we do guarantee to give you our thoughtful opinions and we'll give them to you straight.

WOULD YOU LIKE TO CONNECT WITH THE NETWORK?

Getting started is easy.

Simply return the appropriate response card to subscribe to Micro C, to receive our media kit with everything you need to prepare and submit your advertising, or to receive your technical review questionnaire. If you know others who could benefit by connecting with the network please share this information with them.

SHARE
WARE

Play It Again, DOS

Anthony Barcellos
P.O. Box 2249
Davis, CA 95617-2249
Voice: (916) 756-4866
Data: (916) 758-1002

Anarkey and ARC-rivals; if Tony doesn't watch out, he may have Webster's camped on his doorstep.

Computers are quite literal. Muff one keystroke and your PC comes back at you with "Bad command or file name." Until computers routinely offer DWIM capability ("Do What I *Mean*"), we'll have to watch what we type.

DOS is not particularly helpful here. Sure, you can reissue or edit your last command; but unless you use DOSEDIT or CED, all the prior commands are history.

Steven Calwas of Modern Software has entered this arena with his own command-line editor, Anarkey. Now at version 2.00, Anarkey boasts a mix of features.

The Up and Down arrow keys sequentially retrieve command lines from the command buffer, which is a standard approach among command-line editors. However, Anarkey also lets you enter the initial characters of the command you want to retrieve; it will search the history buffer to retrieve the command you started to enter. The command-line history buffer can be written to, or read from, disk.

The command completion feature of Anarkey extends to filenames as well, matching on a file as soon as you have entered enough characters to get a unique match.

You may stack commands on a single line.

Your input strings can have up to 255 characters.

You may use aliases (including the standard replaceable DOS parameters, %1 through %9).

If these features are not enough, you may enjoy using Anarkey to reconfigure your keyboard and retrieve environment variables for editing.

Anarkey can be reconfigured, too. Calwas provides a utility to edit Anarkey's defaults. Its standard history buffer is only 500 characters long, but you can expand it. In standard configuration, Anarkey requires less than 7400 bytes of memory.

The 50-page user's manual contains lots of examples. In it, Calwas explains the standard Anarkey functions, the use of the AKA (or alias) feature, applications of environment variables, and the default-editing utility.

According to the program's order blank, you owe a modest $25 if you plan to continue using the program. (If you use Anarkey for more than a month without registering, you are in violation of the license agreement. If you truly appreciate shareware and want to encourage authors to keep distributing software in this way, you won't flout the honor system that keeps it going.) There is a special registration schedule for commercial institutions.

Steven Calwas has also arranged for accepting new registrations by credit card via the Public (Software) Library at their toll-free number, (800) 2424-PSL. Visa or MasterCard.

Anarkey, version 2.00
Personal registration, $25
Update for registered users, $6
(California residents add 7% state sales tax.)

Steven Calwas
Moderne Software
P.O. Box 3638
Santa Clara, CA 95055-3638

The ARC War Peace Talks

"You can't fight in here, gentlemen! This is the war room!" –*Dr. Strangelove*

The unlikely happened in Bethesda, Maryland, on June 7, 1989. Phil Katz of PKWare and Thom Henderson of Software Enhancement Associates shared a platform in a joint presentation to a users group. It was a meeting of the Communications SIG of the Capital PC Users Group. SIG chairman Walter Nissen's diplomatic skills proved equal to the formidable task of arranging the simultaneous appearance of the ARC-rivals.

Nissen had carefully enjoined both speakers from attempting to rehash the *SEA v. PKWare* lawsuit at the user forum. Katz and Henderson accepted his strictures.

I called Nissen immediately after the conference was held.

"There were only the most remote references to the lawsuit," he said, "although many in the audience seemed curious about it."

The two combatants apparently spent most of their time discussing the future of archiving and compression utilities.

According to Nissen, Katz and Henderson agreed that there would probably be some movement toward compatibility among archiving programs. But, there were no signs that either party was prepared to move in that direction.

Katz, of course, is bound by the terms of the lawsuit settlement to refrain from producing programs that work with ARC-format archives (unless he and SEA were to agree on a license arrangement). Although Katz has placed his new ZIP format in the public domain, Henderson did not specifically address it.

Nissen thought compatibility didn't interest the speakers as much as it did the members of the audience. From the speakers' perspective, it's easy to understand, he pointed out. Compatibility becomes an issue only at extraction time.

To create an archive you can use either ARC or ZIP. Of course, each vendor hopes its package will become the standard. Each is willing to let third-party developers address compatibility with extraction utilities that understand both formats.

It turns out that many people use archiving programs for data backup; squeezing large amounts of hard disk information into a much smaller space on floppies. That raises a fundamental question about archiving priorities: size versus security. In one sense, the best archive is the smallest archive, and that's one major area of competition among compression programs. File integrity, however, depends on data redundancy.

"The point of archiving is to reduce the number of bits used," said Nissen. "Redundancy runs counter to this."

Vendors will have to decide whether to aim at size or security, since users are clearly interested in both file integrity and compression. Katz and Henderson both said that they were not expecting to replace backup utilities with their programs, but people are already using ZIP and ARC for data storage.

The two competitors further agreed that size is a tradeoff against speed and security. It takes time to figure out the most efficient compression. PKWare managed to beat ARC in the early going by coding the algorithms in assembly language while SEA used C.

The archiving competition is by no means limited to DOS. SEA is already working in other environments and PKWare appears to be doing the same.

SEA appears to have an edge in the corporate market, but several bulletin boards have converted from ARC to PKWare's ZIP.

The competition could also branch out into commercial software, where many developers are interested in embedded routines that would let their programs de-archive files.

Although Katz and Henderson shared similar views on the future of archiving (bright) and its applications (broad), there was no consensus on whether future algorithms would be more effective at compressing files.

However, the LHARC program now making the rounds of the bulletin boards definitely offers better compression. Unfortunately, it's not as fast as ARC or ZIP.

A Capital Group

The Capital PC User Group, host of the forum, is one of the most important and influential users groups in the country. To learn more about the group, call their 24-hr. recorded message number, (301) 762-6775, or write to:

Capital PC User Group
51 Monroe Street Plaza East Two
Rockville, MD 20850

A one-year membership is $35, which entitles you to a subscription to the Capital PC *Monitor*, their fabulous monthly newsletter. The *Monitor* contains a regular column by shareware enthusiast Bill Fowler.

Trojan Viruses

By now everyone knows that Trojan Horse programs are time bombs that eventually blow up and destroy both themselves and everything else on your hard disk. Viruses, however, are more sophisticated, since they attempt to propagate. Viruses generally bide their time until they determine that their progeny have infected enough targets to carry on the strain.

Thanks to continuing advances by sociopathic computer users, now we have reports that archives can be used as Trojan Horses to transport viruses past your watchful eyes. There may be something to it, though uninformed panic has probably caused some of the concern.

Here's how it should work: During archive extraction, comments are displayed that a deranged mind has altered to contain nonstandard escape codes. (PKZIP and LHARC both permit comments in their file formats.)

The escape codes then redefine your keys so that the next time you try to use the DIR command, you actually invoke a FORMAT of your hard disk. (Plus, self-extracting archives could do immediate damage.)

Slick. Not much of a danger, however, unless you use ANSI.SYS as your console driver. Otherwise the escape codes cannot redefine your keyboard. Furthermore, BBS operators have already started to report precautionary steps that users can take to counter the slim dangers that such archives may offer.

The simplest way is to examine the archive before extracting it, searching with a utility for dangerous command strings. Of course, you could just yank the "device=ansi.sys" line from your CONFIG.SYS file. (Are you really using it for anything?)

Why Wait?

ButtonWare has issued version 1.1 of PC-File:dB. Why?

As marketing representative Suzanne Faith tells it, it was the enthusiastic response of users to version 1.0 and their encouraging feedback: "Our users immediately had several excellent ideas and suggestions regarding what they would like to see in the new program. Some of these suggestions were so good, we decided not to wait until our next regular update, late in 1989."

Now you'll be able choose your own names for index files, preventing possible conflicts. Memo fields are wider, the import function is faster, IF calculations have been refined, and relational lookup support supposedly has been improved.

For more information, contact ButtonWare.

PC-File:dB, version 1.1
ButtonWare, Inc.
P.O. Box 96058
Bellevue, WA 98009-4469
(206) 454-0479
(800) JBUTTON (orders only)

◆ ◆ ◆

ON
YOUR OWN

Modem Operated Switch

Ya Gotta Have MOX

By Jay C. Bowden
Epoch Data Devices
P.O. Box 1093
Cardiff, CA 92007

What do you do after you've fought your way through the development cycle, lined up someone to handle distribution, sent out releases, finagled reviews, and placed ads? You wait for orders.

This story begins with the unsolicited article I sent in to *Micro Cornucopia* back in July of '87. "Phone Your Own Clone" described how a person could build the hardware to turn on his PC when the Carrier Detect (CD, also called RLSD) signal in his external modem became active.

A few lines in the AUTOEXEC, a program to interrogate the state of CD via a BIOS call, and suddenly your PC was smart enough to *know* if you were sitting in front of the screen turning the power on, or calling it over the modem. If modem, then CTTY COM1 turns control of the PC over to the phone line!

You probably won't remember the article, because it didn't appear. I had even offered to sell the PC boards and parts kit, with just enough markup to cover the cost of the PC layout. I was *convinced* that this was something that people would *want* to do.

Why I Designed The Product

I had recently worked at two separate locations: one north of home and one equally far south. After a full day in the south, I didn't feel like driving north, so I arranged for the installation of a modem-operated switch at the northern office, which only had one phone line.

I also put a simple appliance timer on the modem only. Turning it on from 7 p.m. to 7 a.m. gave me access to the office PC, and let the one phone line do double duty: daytime voice, nighttime data. It was great!

This was not revolutionary technology, but I was sure that if it was easy and *convenient*, more people would want to take advantage of it. Besides, there seemed to be no one else who had done it right.

Black Box Catalog sold something called a Line Operated Relay for $100, but it was made for a teletype machine. You had to wire it yourself, and it had no override for the times you were sitting at the console. Then there were the sophisticated widgets that had the brains to make you enter a password before they would turn on the PC, but they sold for $300 or more.

Enter MR. MOX

I answered the convenience issue by using an RS-232 jumper box (like a gender changer, only it doesn't change the gender) to intercept the CD signal and run it to a jack on the face of the MR. MOX. (Oh, and that name: like VOX is to voice operated switch, MOX is to modem.)

Then, a small switch in a tiny box at the end of the wire for the override. You could stick this switch anywhere it was convenient near the *front* of the PC, and keep the power cables in the back. It really did install in seconds. I shoe-horned the circuitry inside the case of a 6-outlet power strip, and considered it productized.

Manufacturing—What Can Go Wrong....

There were trade-offs made in manufacturing. Everything you buy has its own peculiar cost vs. quantity curve, but the thing they all have in common is the more you buy, the cheaper the unit cost.

My general plan was not to sweat for absolute lowest cost at the front end. I knew I wouldn't make any money until sales got into the hundreds of units, anyway. So I would concentrate on low manufacturing cost later, if MR. MOX took off.

But costs weren't the major problem. For example, I located the perfect power strip in which to install MR. MOX. Holding it in my hand, I ordered that part, *by Manufacturer's Model Number.* What I got (with the same model number) was a power strip from hell: half the size of my example, single outlets instead of duplex pairs, and *it was riveted together!* This was just the beginning. Relays that I could get routinely in five days suddenly became three-month special orders from Japan. Of course, my PC boards would accept only this brand.

With naïve dedication, I checked every trace

You are not reading a success story here. The facts are true. I have omitted the names to protect the guilty.

the PC layout person should have put down *before* the boards were fabbed. I might as well have gone fishing—sure they fixed the errors that I found, but ¾ of the fixes were done wrong!

I even wound up special ordering the cardboard boxes after I found out they weren't available locally. As usual, it's the small things that cause the most frustration.

Finally I was held up for an entire month by a total flake of a graphic artist. I finally accepted trash she'd produced just to get rid of her.

Eventually, though, the only hurdle left was marketing my product.

Only? Here my dream turned into something of a nightmare. Those of you who want to market your own product—*Beware!* You are not reading a success story here. The facts are true. I have omitted the names to protect the guilty.

Got Those Marketing Blues

The greatest product in the world won't sell itself. Advertising rates in magazines are high—a 5"x3" box among hundreds of other such boxes (among hundreds of pages of bigger boxes) can cost $400 or more. Of course, you have to recover the cost of advertising out of your "profit," but you can't price your product too high.

I determined that the best way to get started was by getting "free" advertising in the form of "new product announcements." I bought a mailing list of computer magazines, and another one of computer users groups, and began mailing MR. MOX press releases.

For various reasons, all of which made sense to me at the time, I proposed a partnership with the proprietor of a dormant one-product hardware business. His well-known (to me, anyway) logo would appear at the bottom of MR. MOX ads. He would handle all advertising and sales. I would supply the product. We would coordinate ads to appear along with the press releases which I would submit. It sounded great.

Time passed. I finally (in frustration) placed an ad myself in *Micro C* (Jan-Feb '89—take a look), since this "partnership" resulted in *no* advertising. MR. MOX, to put it mildly, was not the center of my partner's life.

A Little Exposure

However, I was actually having some success with press releases—that is, they were getting published. *BYTE* magazine (June '88) ran one. *LINK-UP* ran a paraphrase of my data sheet and a photo. *PCM*, a Tandy magazine, gave MR. MOX a mention (June '88). A note in a Q and A column in *Personal Computing* (Sept. '88) continued my media barrage.

Two problems, however. First, the *Personal Computing* author suggested that it would be better to buy an $8.79 piece of software and an appliance timer for the PC that would turn it on when you were likely to call in. Arrrgh! The author had obviously never tried to use a PC with an appliance timer (2 prongs vs. 3). Plus, the PC would crank up every day whether you were going to use it or not.

Second, even when they simply printed the releases, the free advertising didn't work. There was no flood of orders for MR. MOX—calling it a trickle would have been generous.

Then came a break, I thought. *LINK-UP* wanted to evaluate MR. MOX in an article. I sent off a unit and waited expectantly. And waited. And waited. I wish now I were still waiting.

My first indication that the article was out came by phone. A modem manufacturer called to say that their unit didn't have the problem described in the article. Uh-oh. What did the article say? I asked the guy on the other end.

"Well, it said that your product doesn't work. They rated it a poor buy."

I had to call *LINK-UP* to get a copy of the article. The author was a fellow whose name I had seen mostly on book reviews. The article was fairly complimentary, up until the point when the author became frustrated that he could not get his MR. MOX to work. What happened?

He seemed totally convinced that if he set up the modem *the way the modem has to be set up to work with MR. MOX,* the modem would not work with the computer. He did not *try* to set it up the way it has to be set up. He didn't have to. He *knew* it wouldn't work.

He didn't call. He just gave up, ending his article with, "It makes one wonder how the engineers thought through the concept and tested it."

Arrrgh.

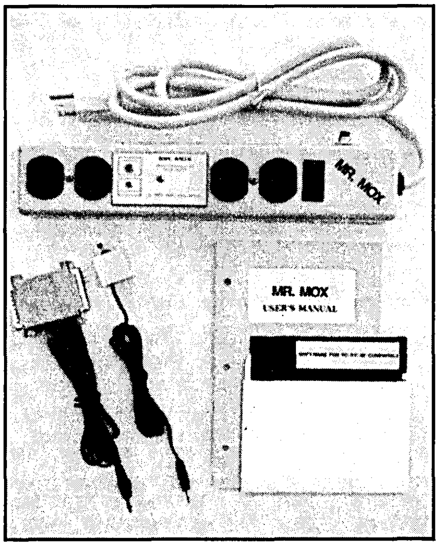
To their credit, they published my rebuttal in their next issue. The author apologized, in print, "for an unintentionally misleading review." The editor apologized on the phone, and they upgraded MR. MOX to a "Good Buy." (Darn hard to use the condemning article/rebuttal/apology in your marketing literature, though!)

Did this undo the damage of the original article? Dave Thompson, your friendly *Micro C* editor, mentioned that usually more people read the "letters" columns than the articles. However, if sales are any indication, it doesn't appear that people read either in *LINK-UP.*

Well, there are always retail sales, right? Not so fast! Big or small, retailers pointed out that customers had not been coming into their stores asking for MR. MOX. Now my point of view was, "If only my product were on display next to XXXX, then people would realize how handy it would be to have this feature."

Sorry. The most shelf space goes to the products that have generated the most sales; zero sales gets you zero shelf space. Welcome to the world of catch-22.

Mr. Mox in the Flesh.



There Must Be A Lesson Here Somewhere....

Manufacturing and marketing a product isn't glamorous. It's hard work. The rewards are possible; the effort a certainty.

First, if you should start down this road, it won't take you long to discover that everyone and their brother (and your brother) will offer you endless advice. If success doesn't beat a path to your door, they'll tell you exactly what you're doing wrong. Of course, they've never tried doing anything similar.

Plan for delays and hitches in materials. If work is done for you, check it. Where possible, don't give anyone more money up front than you're willing to walk away from (my experience with the graphic artist nailed that lesson home).

When sending out press releases, be aware that there is much duplication on mailing lists. Just pick out titles you recognize, and perhaps others that seem particularly oriented toward your product. Be persistent. Be aware that there's a lot of competition for a New Products editor's attention.

Editor's note: If you want an eye opener, show up any Monday morning and you can help Tammy sort through the basket-full of releases.

I know I could have done better with the pictures. I sent B&W prints to all my targets. If you are the artistic editor of a slick mag, and have a choice between techno-surrealistic, visually interesting photos from HP, Compaq, and AST, and a dull but functional B&W pic, which would you choose?

Editor's note: I'm thinking....

Don't count on someone else doing your work for you. No agent or rep or retailer is as interested in your product as you are. Perhaps it seems obvious, but it's so very appealing to think someone else will do the job you don't want.

While the odds are against your generating a retail market, it's not impossible. If you have an "in" with a retailer, give it a try. But prepare yourself to encounter the sort of attitude I found while pushing Floppy Pockets. (Another invention of mine that attached to existing hanging folders so you could keep your floppy discs.... Oh, never mind. That's another story.)

Anyway, this stationery store buyer described to me his recollection of a product that a large, respectable company tried to foist off on him. He *knew* it wouldn't sell.

The company was 3M. The product? Some silly little sticky pads, and they only came in yellow.

Stay Tuned

Has all this discouraged me? Yep.

It looks as though MR. MOX is a bit of a sleeper; a niche product where the cost of finding buyers exceeds the price of the product. (It's still available, make no mistake about that!)

In spite of it all, I find myself thinking.... What if there were an attachment to the phone line that could detect a single isolated ring, and then turn on the power to the PC? If you called back immediately, the PC would answer. This way you could share the line with an answering machine.

What if the PC would turn *itself* on? Then, at 3 a.m., click, beep, PC dials, connects to remote computer, download (upload?), hangup, print maybe, turn off, silence.

What if you called it LATER-ON, since the PC programs itself to come on later? What if this time the picture was in *color*? Hook it up to your printer port, but it doesn't affect printing. A few lines in the AUTOEXEC and the PC knows if it's turning on for an automatic telecom database inquiry, or for a user sitting at the console.

New applications for realtors, stockbrokers, BBS addicts, football players, housewives. MR. MOX—the next generation! Clearly there is an opportunity here!

Maybe just one more release....

◆ ◆ ◆

Around the Bend

Continued from page 4

I gave you the details about the problem, but people had questions so I'll give 'em again.

First, you'll have to understand that getting valid service information on hard drives is about as easy as swiping memos from the CIA. So I make do with a rumor here, an upset customer there.

A while back I started hearing grumbling about hard drive reformatters. The common complaint was that drives didn't seem to work as well after nondestructive format. I questioned the commenters.

"What kind of software?"

"What kind of drive?"

"What kind of controller?"

"Did you have new problems after reformatting and, if so, what kind and how long did it take for them to show up?"

"Were there problems before reformatting?"

They were using SpinRite, 5 1/4" half-height Seagates (usually the 225s), and Western Digital controllers (those who knew which controller they had). The new problems showed up within a few weeks of the format, and most folks were seeing a bad sector or two before they began the reformat. (Bear in mind, this is a *tiny* survey, involving only half a dozen drives.)

By the way, if you've been using any hard drive reformat packages that save the data, by all means drop me a post card with answers to the above questions (to Micro C, P.O. Box 223, Bend OR 97709) or leave a message to the SYSOP on our BBS (503-382-7643). That way I'll have a bigger survey.

Despite the tiny size of the sample, I suspected something might be connecting all this together. I'd noticed that our XT machines took hours for a reformat while ATs took only minutes. Plus, the poor XT drives were seeking endlessly (especially as the format progressed) while the ATs were just stepping merrily along track by track.

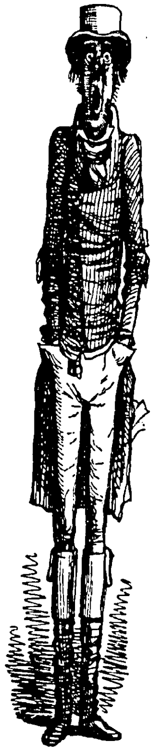
It was then I discovered that the older XT controllers automatically home the head (the move toward home is very slow), then count their way out, when they're asked to do a single track format. (The controller designers assumed, I supposed, there was no other way to guarantee head position. When you're formatting you can't count on reading an already formatted track to check position. Only track 0, or -1 in the case of the 225, is independently verifiable.)

The packages I've seen (SpinRite, Disk Technician, and Optune) are smart enough to spot a cludgy controller (by measuring the time it takes to format the first couple tracks). They speed up the process quite a bit by doing a seek to track 0 before issuing the track reformat command. But the head still has to move all the way home and all the way back before formatting each track.

What does all this activity have to do with errors? Very good question. It should make no difference how far the head traveled, it should settle at exactly the same spot.

Hard drives that dedicate one surface to servo (position) data have no problem positioning their heads precisely (these drives have an odd number of read/write heads—you'll find that full-height Seagates and Priams have odd heads).

But the little half-heights insist on cramming data onto every platter; so they're not going to dedicate a surface for position, they just count stepper motor pulses. (At least this is the case for the stepper based assemblies, voice coil positioners with optical position sensors are probably fine.)



Anyway, the heads can drift, both with age and with temperature. That's the main reason you purchase a reformatter. The heads drift away from the original tracks. When you reformat, you rewrite the tracks where the heads are. However, the heads may be in one place after a seek from 0 and in a slightly different place after stepping just once.

What began as a fix for slight misalignment could actually make the problem worse. At least that's how I read it.

So.... If you have a half-height drive with a stepper motor for head position, an even number of heads, running on a Western Digital controller, and you find it takes an hour or more to do a nondestructive reformat.... Well, I think I'd back off all the data and follow up with a low level reformat and copy all the data back.

Might be enough reason to get an AT, eh?

Games

Okay, enough serious stuff, how about a light game of chess? (An oxymoron for sure.) I've tried several chess programs, but the Chessmaster 2100 has the best graphics, the best teaching mode, and it has a database of classic games.

Want to step back through one of your old games to see where you went wrong? Fine. Want the computer to suggest the next move? Fine, just tell it how much time it has to explore the possibilities. Want to know which positions can be attacked by the opponent? Want to see your legal moves?

You can use a mouse or the keyboard to select and move pieces, and you do it just like you would on a real chessboard. You can select a top-down view of the pieces or an oblique view, both easy to understand.

Does it play a strong game of chess? I don't know. When I put it into beginner mode, I can beat it. That's all that matters.

Chessmaster 2100
$49.95
The Software Toolworks
19808 Nordhoff Place
Chatsworth, CA 91311
(818) 885-9000

Empire

"Empire is a wonderful game. Jerry Pournelle rated it the top game, said he wound up playing it until 2 a.m.," my brother commented as I threaded my way among computers.

Interesting. Jerry might get off on a silly game, but I have a column to write. Besides, I've sworn off trivial pursuits like war games. (I tried a board game called Battleship once—found it about as exciting as cleaning the cat box.)

"Try my copy, you'll like it."

I glanced through the manual, inside the back there were pictures of the authors. Walter Bright? Our Walter? Author of Zortech C++? Speaker at SOG? He wrote a silly game?

I borrowed Don's copy.

That very evening, about midnight, as I finished editing some articles, I remembered the game. Popped in the disk and:

"Empire, Wargame Of The Century"

Let's see, I start with one city and that city can produce an army (every 6 turns) or a plane (every 17 turns) or one of a half dozen ships.

Armies capture cities, troop ships haul armies, destroyers map new territory, cruisers kill destroyers.... You play against the computer, against another player, or both. How addictive is it?

I called Walter and asked (not at midnight, you understand, I'd be interrupting his work day. I waited until the following noon).

He said he'd written the game on a PDP 11 while a student at Cal Tech. When he went to Cal Tech, there were no restrictions on use of the 11; when he left, games were restricted to graveyard shift—midnight to six a.m.

"So we renamed 'empire' to 'link,' or something like that. That was the only way we could play it during the day.

"I no longer play it; after thousands of hours, I'm playing with other things like C++. If I started playing it again I'd want to add new things, like making the computer more competitive, and I wouldn't get anything else done.

"You know, it's hard to make the computer play a really intelligent game. Most game writers create a dumb algorithm but give the computer some kind of advantage, such as producing men faster. I worked hard on the game algorithm, so the computer could play by the rules and still be competitive.

"Anyway, after Cal Tech, I knew it was a good game; I tried to sell it on my own. It sold okay in spite of my efforts, but mail order consumed too much time—filling orders, answering letters. Just a lot of details that don't interest me. I didn't want to be involved in that end of things.

"So I collect royalties on the game and on C++. The thing with royalties is that you work and work and work and it's hard to keep in mind that eventually you're going to get paid for it. Most people won't work for royalties because of the lag.

"Anyway, Interstel takes care of sales and promotion. In fact, Empire was just reviewed in *Penthouse*."

But I'm getting ahead of myself—let's see, where was I. Yeah, I start with one city, explore about, I capture more cities with my armies, make more armies, expand my territory, capture more cities. Hey, there's an enemy troop ship! Where's one of my destroyers? Geeze, it's getting light outside. Can't be, it was midnight...just minutes ago.

Empire
$49.95
Interstel
P.O. Box 57825
Webster, TX 77598
(713) 486-4163

ASMFLOW

I haven't done any assembly language programming lately, mostly I work in Pascal (occasionally those efforts are perforated by a C filter). However, Larry has been disassembling some things with Sourcer, and Sourcer generates assembly language (for some reason no one has come up with a disassembler that regenerates the original C, or whatever).

So, I dug out the office copy of ASMFLOW and attacked the output of Sourcer. Boxes and code and arrows, pretty fancy. Makes it a lot easier to see the structure of the program. The program also gives you an estimated time (and clock cycles) per routine for the 8088, 80286, and 80386.

It was interesting to watch the number of cycles it estimated. For instance, on Larry's screen update routine (we're talking fractals here), it specified 486 clocks for the 8088, 273 for the 80286, and 156 for the 80386. In fact, all the code I checked came out with about the same ratio.

Boy, shows how gullible I am. I believed it when the second source of 286s announced that a 20 MHz 80286 is as fast as a 20 MHz 80386. It might be if they carefully select instructions, but not for any of the code I've seen.

Anyway, I remember writing assembly for a Z80. I would have traded my second 8" drive for a tool like this. After seeing what ASMFLOW displays, 8088 code looks almost easy. Shucks, if Turbo Pascal 5.5 hadn't just appeared....

ASMFLOW
$99.95 + $3.00 S&H
Quantasm Corporation
(formerly Quantum Software)
19855 Stevens Creek Blvd., Suite 154
Cupertino, CA 95014
(408) 244-6826

Turbo Pascal

For the past nine months, I've been working on a hardware project. You know, it's one of those design jobs that you can't talk about with your closest compatriots. However, into every fun project a little software falls.

This is one of those I/O things where I prod lots of A/Ds, MUXs, relays, etc. Ten years ago I'd have done it in BASIC. Eight years ago I'd have written it in Pascal (although I'm not sure how, there wasn't much hardware support in the old Microsoft Pascal). Six years ago I'd have done it in FORTH. Five years ago I'd have gone nuts trying to read what I'd FORTHed. Four years ago I'd have written it in C. Three years ago I'd have written it in Modula.

Last year I started writing it in Pascal. Turbo Pascal 4.0.

Units make perfect libraries. I write them, compile them, and use them. I write the low level stuff once. When it works, it works. Of course C has libraries, but I don't C as well as I Pascal. When I C I usually use my old copy of Aztec. That way when the compiler complains or the code doesn't run, I know it's my fault.

Now Borland's released 5.5. If objects in Pascal are as good as objects in C++, we may find that software development is no object. Or it is an object. Or people will no longer object to writing software. (Sorry, I'm not being objective.)

Anyway this is beginning to sound an awful lot like something I'd put in the editorial.

Speaking Of Objects

With Borland's release of objects, can Microsoft be far behind? Nope. In fact Redmond, Washington, came out with Pascalian objects before Borland.

Microsoft's Quick Pascal is just the first shot. I think Microsoft would like nothing more than to invade Borland's home court, the Pascal market.

The next year should be very interesting. Microsoft is smarting from reviews, which conclude that Turbo debugger is outperforming CodeView—no question MS would like to depose Borland in the Pascal market and they're counting on objects to do it.

Are objects really that important? Look at AI. Prolog started out with fireworks and then.... (By the way, what did happen to Prolog? Maybe Gary remembers Prolog.) However, I was talking to Bruce Eckel the other day and I mentioned some of Blaise's Pascal libraries (most useful).

ICs PROMPT DELIVERY!!!
SAME DAY SHIPPING (USUALLY)
QUANTITY ONE PRICES SHOWN for JUNE 25, 1989

OUTSIDE OKLAHOMA: NO SALES TAX

DYNAMIC RAM			
SIMM (1)	256Kx36	80 ns	$400.00
SIMM (2)	1Mx9	80 ns	220.00
SIMM (2)	1Mx9	85 ns	195.00
SIMM (2)	256Kx9	80 ns	95.00
1Mbit	1Mx1	100 ns	16.25
41256	256Kx1	60 ns	8.50
41256	256Kx1	80 ns	7.35
41256	256Kx1	100 ns	5.95
41256	256Kx1	120 ns	5.75
4464	64Kx4	120 ns	8.50
41264 (3)	64Kx4	120 ns	12.50
EPROM			
27C1000	128Kx8	200 ns	$27.50
27512	64Kx8	200 ns	9.95
27256	32Kx8	150 ns	7.25
27128	16Kx8	250 ns	4.50
STATIC RAM			
62256P-10	32Kx8	100 ns	$26.50
6264P-12	8Kx8	120 ns	7.95
6116AP-12	2Kx8	120 ns	4.95

(1) PS2/70 OR PS2/80 1Mbyte
(2) FOR: AST AT&T COMPAQ DELL MAC'S MYLEX PS/2 Tandy WYSE WD
(3) 2-PORT VIDEO RAM

80387-16 80387-20
80387-25 80387-33
80287-8 $215.00
8087-2 $135.00

OPEN 6½ DAYS, 7:30 AM-10 PM: SHIP VIA FED-EX ON SAT.

SAT DELIVERY INCLUDED ON FED-EX ORDERS RECEIVED BY:
Th: Std Air $6/3 lb
Fr: P-1 $12.25/1 lb

MasterCard/VISA or UPS CASH COD
MICROPROCESSORS UNLIMITED, INC.
24,000 S. Peoria Ave.,
BEGGS, OK. 74421
(918) 267-4961
No minimum order. Please note: prices subject to change!
Shipping, insurance extra, up to $1 for packing materials.

Reader Service Number 37

PC COMPATIBLE ENGINEERING

Annabooks gives you the **hardware, software, and firmware** information you need to design PC-compatible systems faster and better. And you have control of your design from the ground up -- our firmware and software products include **source code!** Plus all the utilities you need.
Do hardware design? **Doctor Design's** 1M DRAM SuperSpec is the first of a series of hardware books you won't want to miss. And a PC Bus timing book is on the way! Start by getting these books:

AT BiosKit: an AT Bios with source code you can modify. With setup & debug. 380 pages with disk, $199
XT BiosKit: Includes a debug. 270 pages with disk, $99
Intel Wildcard Supplement for XT BiosKit: Includes ASIC setup, turbo speeds, 60 pages with disk, $49
1M DRAM SuperSpec: Design your memory to all mfg's specs at once! Lots of timing diagrams & tables, $79
PromKit: Puts anything in Eprom or SRAM; DOS, your code, data, you name it! With source on disk, $179
SysKit: Here's a debug/monitor you can use even with a brand X Bios. Includes source, of course. $69
XT-AT Handbook: The famous pocket-sized book jam-packed with hardware & software info. $9.95 ea. or 5 or more for $5 each.

Software tools: You need MS C & MASM 5.1 for modifying the Kit products.

FREE Mention this ad when you order and get a **free XT-AT Handbook by Choisser & Foster!** Hurry before we come to our senses and change our minds.

Annabooks
12145 Alta Carmel Ct Suite 250-262
San Diego, California 92128
(619) 271-9526 Money-back guarantee

Reader Service Number 160

Around the Bend

"Are they object oriented?"

"No, not the versions I have."

"Then I have real trouble getting interested. Too much trouble to use a library that's not designed around objects."

He has a point. The less you need to know about a library routine, the easier it is to use. Objective libraries should be very easy to get close to. (We're talking libraries here, not librarians.) Speaking of librarians....

How Fast Is Fast?

I spend most of my life plodding along behind an 8 MHz 80186 system while out in the other room lies our latest acquisition, a 20 MHz 80386 screamer. It's usually busy—grinding out fractals.

When I hit return on my 186 things pick right up, and it isn't very long before something has happened. (I do have to clean up around the processor socket after rust flakes off the accumulator.) By the time I've finished hitting return on the 386, however, things are fairly well completed.

I understand we'll soon see a new 60 MHz 486. (How soon is open to speculation.) There's a good chance Intel will run into timing problems, after all the chip will be doing a fair number of things in a short amount of time. I know a lot of silicon engineers (hard rock miners) who'd be hard pressed to do that much stuff that quickly.

Don't misunderstand, I don't have a pick to grind. Intel can make chips fly fast and furious, but I reserve the right to finish my return before the damned computer finishes the answer. Kinda keeps things in perspective.

Silence

The reason my little Toshiba 1000 has become my favorite writing computer isn't so much what it has, it's what it hasn't. It hasn't any, uhm, noise.

There's no hard drive to whine, no power supply fan to buzz, no monitor to whistle, no nothin'. The only thing I hear is the chunk, chunk, whirr of the floppy drive when I'm loading or saving a file. But then it shuts up so I can work.

Someone should come up with standard 150 and 200 watt PC power supplies with *quiet* fans (slow turning with special blades). There's no reason that moving air has to be so noisy.

Since we're now moving (more or less successfully) to 3 1/2" drives, it seems like some of those should be both reliable and *quiet*. Maybe there is one, though. Maybe I just haven't heard about it.

Silently yours,

David Thompson
Editor and....

Continued from page 6

practical articles about what you can do with what you have: articles on disk formats, what's a FAT, using CONFIG.SYS, etc. So many magazines have articles that are just hype for new hardware and software that I can't afford anyway. Besides that, your non-glossy pages are easier to read than the glossy ones that have so much glare.

However, I subscribe to your magazine for the computer content. A few remarks once in a while, about life in general or your SOG adventures, are refreshing. And I enjoy your wry humor. But your July/August issue went overboard on your metaphysical views. That's not what I subscribe for.

Peter E. Walberg
390 Loma Dr. Box 57
Forsyth, IL 62535-0057

Editor's note: Thanks for the comments Peter, I take them to heart. So far, we've received seven letters, calls, and BBS messages opposed to my mentioning the metaphysical in the editorial (two of those were subscription cancellations).

On the other side, seven people have been very supportive. "Where have you been all these years?" has been the general comment. I particularly thank those of you who sent reading material (e.g., The Tao of Pooh, The Mystic Path to Cosmic Power, *and, curiously,* Humor Suddenly Returns*).*

I've been slowed a bit on the books however. Turbo 5.5 just arrived, Hitachi sent a whole box-full of controller data books, and I've got to finish responding to these letters. Isn't life a kick?

LIMBO's Distance Sensor

Since he shows us neither mathematical proof nor empirical data, I don't know how Bob Nansel ("The LIMBO Project," Issue #47) came to the astounding conclusion that the intensity of reflected light is inversely proportional to the fourth power of distance. But he is clearly in error in this assumption.

Mr. Nansel contends that since the intensity of both the incident wave and the reflected wave are subject to the inverse square law, then somehow these two functions should be multiplied, resulting in an inverse fourth power law. I'm curious as to what happens if, after leaving the source, the light is reflected twice instead of just once before it reaches the detector. Do we then have an inverse sixth power, or an inverse eighth power relationship?

The power detected is actually proportional to—

$$(d_1 + d_2)^{-2}$$

where d_1 is the source-to-reflector distance and d_2 is the reflector-to-detector distance. No product, just the inverse square of the sum of the distances.

As Mr. Nansel has pointed out, we are measuring power per unit area. The dimensional analysis must be consistent with these units. The square of d (a length) is consistent with area, but the fourth power of d is not.

Neither of the two parts of the article published so far have made it clear just what the upper and lower range limits are. So it isn't possible for the reader to estimate the limits of sensor input and thus determine whether he needs a log, rather than a linear, amplifier.

I hope, upon reflection, Mr. Nansel sees the light (puns intended).

Don E. Sweet
2161 Snowberry Road
Tustin, CA 92680

Editor's note: When I read Bob's article for the first time, my physics antennae went up, too. So I gave him a call to set him straight on inverse square fall off. Neither of us could convince the other how wrong he really was, so I headed off to the library to check out Bob's reference to something called the radar equation.

I dusted off an ancient tome on radar, and there it was—an inverse fourth power relationship. Now whether this equation is empirical or founded in theory, I can't say. And it still feels wrong to me. Anyone have anything to add to this powerful controversy? (The airlines are certainly hanging on this one.) (LCF)

To List Or Not To List

I have enjoyed your magazine very much. Especially those articles on computer hardware (e.g., controlling the outside world through the parallel port) and the Pascal columns (more, please).

My only complaint is about what I seem to get for my subscription fee. Since you only publish six times per year, I would like to see fewer *National Geographic* type articles. For example, the trips through Turkey are interesting; but if you were the size, cost, and periodicity of, say, *PC Computing*, those articles would be a refreshing insert rather than a squandering of valuable space.

Judging from your gushing (pun intended) comments on the SOG, there doesn't appear to be a shortage of computer-related material. Would it be possible to include some of the SOG discussions in future issues?

A few times in past issues, great articles were diminished by statements to the effect of, "Here's a partial program listing. Call our BBS for the complete program." Since your BBS isn't PC-Pursuitable, the long distance charges make the call cost-prohibitive. Please don't print only partial listings.

Product reviews are okay, but I enjoy *Micro Cornucopia* more as a nuts-and-bolts type learning tool on the how-to and why-of the microprocessor world. Because of this, in the past I have recommended your magazine to others as a valuable information source.

Ron DesGroseilliers, Jr.
209 Cecil Ave.
Springlake, NC 28390

Editor's note: Boy. I know what you mean. I like complete listings. I also enjoy the fun stuff that helps me understand the person who's behind the technical information. This issue Larry was struggling with this question. Much of his current fractal code is a simple repeat of code we've run in

Continued from page 75

recent issues. So he's limited it to the PostScript additions. That way we have room to run one more article in this issue.

Making Custom PC Cards

If you took to heart the piece in Issue #43 by Bruce Eckel on designing your own cards for your PC, you may wish to make something less prototype and more production than the prototype cards described. If you have OrCAD PCB, log on to OrCAD's BBS, (503) 640-5002. Look for IBMPCB.ARC in the PCB Download area, among the Customer Contributed Models.

You won't need to be a registered user to download it, but you will need OrCAD PCB to make use of it. It has the outlines and connectors already drawn for half- and full-length cards for both XTs and ATs. There is a useful .DOC file as well. When you've done the prototype the way Bruce suggests, and you're ready to launch your creation on the world, this is the way to take the next step.

John Innes
120 MacPherson St.
Cremorne, NSW 2090
Australia

Free PCB Autorouter

I've enjoyed *Micro C* for many years as a newsstand purchase, and recently (finally!) ordered a subscription. I read with interest the articles in Issue #45 about PCB layout systems.

Micro C readers might like to hear of the best bargain available in this type of software: autorouting PCB layout software that runs on EGA systems, outputs to an HP laser printer, and comes with full C and assembly source code. The price: FREE!

Interested people can contact the program author—

Randy Nevin
1731 211th Pl. NE
Redmond, WA 98053

Just send a formatted floppy and a self-addressed mailer with sufficient postage. The system doesn't have a flashy interface, but it works and is suitable for two-sided board production. The author invites feedback on the product so he can improve it.

David Gwillim
159 Woodbury Road
Hicksville, NY 11801-3030

Down Under Order

I would be grateful if you would send the Kaypro IV schematic to me.

Would you please tell your Mr. David J. Thompson that in the interest of good international relations, I typed this request whilst upside down. The Kaypro did not seem to mind, but holding up the keyboard while typing was a pain.

A. W. Bastian
Box 501 P.O.
Campbelltown, NSW 2560
Australia

Rebuttal Rebuttal

What did I say? What did I say? Quick, pass me that half-eaten bag of CCCCs (crispy chocolate chip cookies) from your editorial in Issue #45. What was it that I said in my letter (in Issue #42) that so upset John Mulligan ("Letters," Issue #45), a self-confessed computer store person?

I turn to my pile of magazines and dig out the edition to read my letter again. Ah, there it is. One passing reference to computer store sales staff (CSSS) and the insinuation that many of their ranks are pretenders.

Not such a damning statement methinks. A mere pinprick in the bloodletting of business, I would have thought. Certainly defensible most would say, I think. (A thought-provoking letter, this.)

Well it appears that John was provoked into thought, too. Several hundred words worth in fact. All in defense of himself. Some were dripping with sarcasm and vitriol. Others I have not heard of; what do "buffalo chip slinger" and "dishwashing jarhead" mean?

I suspect that the former equates to an Arthur Daley used car salesman and the latter to a pre-stardom Sylvester Stallone doing night jobs (maybe post Rocky X and divorces, as well). And if money be the measure of success then I suppose I should be flattered with the comparisons, but I guess I'm indifferent.

I understand "intolerant," but I don't think it applies. After all, I am answering John's letter. And from the passion of it, I'm glad I don't live in Syracuse. Author I am not, but I hope John doesn't have the ear of Khomeni....

Honest, John, I was not speaking specifically of you when I said CSSSes are less expert than they claim to be. During the last ten years, I have computer shopped in many places around the world (including Bend...by telephone).

Unfortunately, I have found many staff who don't understand their product range, a lot who believe that they do, some who make an effort to learn, and some who have been very good. Perhaps the good ones get promoted to managers. Salesmanship is more important than technical expertise on the shop floor. Anyway, my original comment was aimed at the pretenders.

To other matters. I see that DJT of Micro C (the tall skinny guy with the startled expression) is suggesting that the answer to the Ultimate Question of Life, the Universe, and Everything is a variable! (See "Culture Corner" Issue #45.) A daring contention. Just when the world was getting comfortable with 42.

Still, the idea does have consistency with the Uncertainty Principle. Unless, of course, with your blossoming confidence you may task Larry to look at Schrödinger's cat and declare that the Uncertainty Principle is not a sure thing. Avant-garde stuff. Obviously, Micro C thinks it can make waves in the mainstream of contemporary publications. But be mindful of the puddles you are leaving behind.

Speaking of changes, I am glad the full frontal is back to the front so to speak. Thanks for the CCCCs, but one didn't taste too good. Could it have been a Buffalo chip cookie?

Having reread this letter, I hope there are no sensitive Sylvester Stallone fans out there. I may be running the risk of another nasty-gram.

Bevin J. Pettitt
84 Sun Valley Road
Valley Heights, NSW 2777
Australia

◆ ◆ ◆

UNITS &
MODULES

Keeping Order, Generically

By Michael S. Hunt
2313 N. 20th
Boise, ID 83702
(208) 233-7539

Are you out of sorts? Are your friends out of sorts? Well, here's the sort for all sorts, the ultimate generic sort. (You can sort all your generics this way.)

There are many ways to sort data. Quicksort, shellsort, heapsort, bubblesort, insertionsort, and binsort to name a few. Although each of these sorting methods has its pros and cons, all accomplish the same end (sorted data, we hope) by different means. Usually they're implemented for a specific data structure.

Each time the data structure changes, we face the task of modifying the sort routine. In keeping with the spirit of reusable tools, how about a sort routine that handles any data structure.

As I mentioned last issue, one of my new toys at work is a VAX cluster running VMS. VMS has many, many, many built-in routines for the programmer. The SOR routines make up one set.

The SOR routines let you sort or merge records and files with a few simple system calls. The unit GenSort (see Figure 1) only supports sorting of records. A full emulation of the VMS SOR routines would require about three to four times the source code.

I hope to make clear how the routines work so you can use them and adapt them to your specific programming style and needs. I'll cover them in the order of use.

GenSrtBegin

This routine initializes the sort by passing key information and sort options. An array of words contains the key information. The first word is the number of keys. Each four-word group after that specifies the data type, order (ascending or descending), offset from the beginning of the record (first byte is 1, not 0), and the length of the key.

GenSrtBegin returns a sort id that uniquely identifies the sort, and a status code. It's the first routine to call for a sort.

GenSrtRelease

The GenSort routines don't receive all the data at once. A call to GenSrtRelease passes each record to the sort. It builds the key for each record according to the key information stored in srtKeyArr by GenSrtBegin.

GetMem allocates storage space for the data and key because the standard procedure New allocates a specific number of bytes according to the variable's type. Our variable type is largely unknown and the storage requirements vary between sorts.

The key is built a byte at a time. By using variable type casting to access the address offset, it increments the pointer to the current byte location. Use move to transfer the actual record information from the calling routine's variable to the space allocated. Then make a call to GenBinInsert to insert the information in the correct location in the binary tree. (GenBinInsert is part of the GenBinTree unit. See Figure 2.)

GenSrtDoSrt

Nothing really happens at this stage of the sort. The sort status changes to prevent further entry of records. If you used a sort method other than the binary tree insertion, this would be the appropriate procedure to initiate the actual sort.

I choose the binary tree insertion sort because it works naturally with the record-at-a-time loading of SOR. It's also a natural lead into next issue's topic.

GenSrtRetrieve

Use this to get your data back from the black hole you've dumped it into. This retrieves the data from the binary tree and moves it back into the calling routine's variable. The procedure GenBinRetDelSmRec stands for "return and then delete the smallest record from the binary tree."

GenSrtEnd

This deletes any records remaining in the tree and returns the srtId to a "not valid" status, making it available to another sort.

GenSrtStat

This routine returns the status record for the passed srtId. It's great for debugging.

GenSrtMsg

This procedure provides a text interpretation of the error messages and sort states.

Miscellaneous

GenSort allows up to MAX_SRTS sorts at one time. I've found this feature very useful when reading data from several files and preparing several reports. GenSort does little error checking. The addition of more data types such as integer and real is necessary for this to be a really useful tool.

As always, the Turbo Pascal 5.0 and Modula-2 code for the two units, along with sample programs, are available on issue disk #49 ($6) or the Micro C BBS.

Next Time

One of the problems with using the GenBinTree (see Figure 2) unit is that if you insert the data into the tree in a somewhat sorted order (either ascending or descending), the tree begins to resemble a linked list. Insertion times become lengthy and the sort performance goes to heck. If you keep the tree balanced, then search and insertion times get better.

The AVL method (named for developers Adelson-Velskii and Landis) is one way of maintaining height balanced trees. Height balancing does cost. It requires additional processing time to keep the tree balanced. A balance factor, the difference between height of the right and left subtrees of a node, must be stored with each node.

A programmer has to weigh the advantage of faster access speed (i.e., data retrieval) against the computational and storage overhead of keeping the tree balanced. The height balanced tree is ideal when you'll build the initial tree and then make very few insertions and deletions, but make many queries.

Join us next time when Joe Reader says, "Is that really a full-blown, generic data, AVL, height-balanced tree toolbox?" And Michael (Midnight Programmer) Hunt replies, "Yes siree, it really is."

◆ ◆ ◆

Figure 1—Generic Sort Unit

```
unit GenSort;
(* Author: Michael S. Hunt          Date: June 1, 1989
   This source code is release into the public domain. *)
interface

const MAX_KEYS = 16;
      MAX_SRTS = 8;
      MAX_MSG = 10;
      MAX_DATA_LEN = 32768;
      MSG_SIZE = 40;
      srtMsg : array[1..MAX_MSG] of string[MSG_SIZE] =
                   ('successful operation',
                    'zero records left to retrieve ',
                    'routines called in incorrect order',
                    'maximum sorts exceeded',
                    'too many keys',
                    'invalid sort id',
                    'not valid sort',
                    'sort in release state',
                    'sort in retrieve state',
                    'sort done');
      GenSrtErr_NM   =  0;   (* successful operation *)
      GenSrtErr_ZR   =  1;   (* zero records left to retrieve *)
      GenSrtErr_ICO  =  2;   (* routines called in incorrect order *)
      GenSrtErr_MSE  =  3;   (* maximum sorts exceeded *)
      GenSrtErr_TMK  =  4;   (* too many keys *)
      GenSrtErr_ISI  =  5;   (* invalid sort id *)
      GenSrtSt_NV    =  6;   (* not valid sort *)
      GenSrtSt_REL   =  7;   (* sort in release state *)
      GenSrtSt_RET   =  8;   (* sort in retrieve state *)
      GenSrtSt_DONE  =  9;   (* sort done *)
      GenSrtDType_BL =  1;   (* type boolean *)
      GenSrtDType_B  =  2;   (* type byte *)
      GenSrtDType_W  =  3;   (* type word *)
      GenSrtDType_C  =  4;   (* type char 1 byte *)
      GenSrtDType_ST =  5;   (* type string 1..255 bytes *)
      GenSrtOrder_A  =  0;   (* ascending sort order *)
      GenSrtOrder_D  =  1;   (* descending sort order *)

type  KeyRec = record
                   dataType, order, offset, length : word
               end;
      Keyarr = array[1..MAX_KEYS*4+1] of word;
      Bytes  = array[1..MAX_DATA_LEN] of byte;
      Chars  = array[1..MAX_DATA_LEN] of char;
      PtrRec = record
                 ofs, seg : word
               end;
      SrtKeyRec = record
                       nbrKeys : word;
                       key : array[1..MAX_KEYS] of KeyRec
                  end;
      SrtStatRec = record
                        nbrKeys, dataLen, keyLen, srtState : word;
                        nbrRec  : longint
                   end;
      SrtStr = string[80];
var   srtKeyArr  : array[1..MAX_SRTS] of SrtKeyRec;
      srtStatArr : array[1..MAX_SRTS] of SrtStatRec;

procedure GenSrtBegin (var key; dataLen : word; var srtId : word;
                       var srtStatus : word);
function GenSrtBeginF (var key; dataLen: word; var srtId: word): word;
procedure GenSrtRelease (var rec; srtId : word; var srtStatus : word);
function GenSrtReleaseF (var rec; srtId : word) : word;
procedure GenSrtDoSrt (srtId : word; var srtStatus :word);
function GenSrtDoSrtF (srtId : word) : word;
procedure GenSrtRetrieve (var rec; srtId: word; var srtStatus: word);
function GenSrtRetrieveF (var rec; srtId : word) : word;
procedure GenSrtEnd (srtId : word; var srtStatus : word);
function GenSrtEndF (srtId : word) : word;
procedure GenSrtStat (var srtStatus : SrtStatRec; srtId :word);
procedure GenSrtMsg (srtStatus : word; var srtString : SrtStr);

implementation
uses GenBinTree;
```

Continued on next page

```
var srtRootArr : array[1..MAX_SRTS] of treePtr;
    j : word;

function NextSrtId : word;
var  j : word;
     done : boolean;
begin
  j := 1;
  NextSrtId := 0;
  done := false;
  repeat
    if srtStatArr[j].srtState = GenSrtSt_NV
    then begin
      NextSrtId := j;
      done := true;
      srtStatArr[j].srtState := GenSrtSt_REL
    end;
    j := j+1
  until (j > MAX_SRTS) OR (done)
end;

function ValidSrtId(srtId : word) : boolean;
begin
  if (srtId <= MAX_SRTS) AND (srtId > 0) then
    if srtStatArr[srtId].srtState <> GenSrtSt_NV
      then ValidSrtId := true
    else ValidSrtId := false
end;

procedure ClearSrtId(srtId : word);
begin
  if (srtId <= MAX_SRTS) AND (srtId > 0) then
    srtStatArr[srtId].srtState := GenSrtSt_NV
end;

procedure Descend(var rec; recLen : word);
var j : word;
begin
  for j := 1 to recLen do
  begin
    Bytes(rec)[j] := $FF xor Bytes(rec)[j]
  end
end;

procedure GenSrtBegin (var key; dataLen : word;
               var srtId : word; var srtStatus : word);
begin
  srtStatus := GenSrtBeginF(key, dataLen, srtID)
end;


function GenSrtBeginF (var key; dataLen: word;
                       var srtId: word): word;
var j, k : word;
begin
  srtId := NextSrtId;
  if srtId > 0
  then begin
    srtKeyArr[srtId].nbrkeys := KeyArr(key)[1];
    if srtKeyArr[srtId].nbrkeys <= MAX_KEYS
    then begin
      for j := 1 to srtKeyArr[srtId].nbrKeys do
      begin
        srtKeyArr[srtId].key[j].dataType :=
                                    KeyArr(key)[j*4-2];
        srtKeyArr[srtId].key[j].order :=
                                    KeyArr(key)[j*4-1];
        srtKeyArr[srtId].key[j].offset :=
                                    KeyArr(key)[j*4];
        srtKeyArr[srtId].key[j].length :=
                                    KeyArr(key)[j*4+1]
      end;
      srtStatArr[srtId].nbrKeys :=
                              srtKeyArr[srtId].nbrKeys;
      srtStatArr[srtId].dataLen := dataLen;
      srtStatArr[srtId].keyLen := 0;
```

```
      for j := 1 to srtKeyArr[srtId].nbrKeys do
        srtStatArr[srtId].keyLen :=
                        srtStatArr[srtId].keyLen
                      + srtKeyArr[srtId].key[j].length;
      srtStatArr[srtId].nbrRec := 0;
      srtStatArr[srtId].srtState := GenSrtSt_REL;
      GenSrtBeginF := GenSrtErr_NM
    end
    else begin
      ClearSrtId(srtId);
      GenSrtBeginF := GenSrtErr_TMK
    end
  end
  else GenSrtBeginF := GenSrtErr_MSE
end;

procedure GenSrtRelease (var rec; srtId : word;
                         var srtStatus : word);
begin
  srtStatus := GenSrtReleaseF(rec, srtId)
end;

function GenSrtReleaseF (var rec; srtId : word) : word;
var data, key, tkey : dataPtr;
    j, k : word;
begin
  if ValidSrtId(srtId)
  then begin
    k := 1;
    GetMem(key, srtStatArr[srtId].keyLen);
    GetMem(data, srtStatArr[srtId].dataLen);
    tkey := key;
    for j := 1 to srtKeyArr[srtId].nbrKeys do
    begin
      if (srtKeyArr[srtId].key[j].dataType =
          GenSrtDType_BL)
      then begin
        tkey^ := Chars(rec)[srtKeyArr[srtId].
                               key[j].offset];
        Inc(PtrRec(tkey).ofs,1)
      end
      else if (srtKeyArr[srtId].key[j].dataType =
               GenSrtDType_B)
      then begin
        tkey^ := Chars(rec)[srtKeyArr[srtId].
                               key[j].offset];
        Inc(PtrRec(tkey).ofs,1)
      end
      else if (srtKeyArr[srtId].key[j].dataType =
               GenSrtDType_W)
      then begin
        tkey^ := Chars(rec)[srtKeyArr[srtId].
                               key[j].offset+1];
        Inc(PtrRec(tkey).ofs,1);
        tkey^ := Chars(rec)[srtKeyArr[srtId].
                               key[j].offset];
        Inc(PtrRec(tkey).ofs,1)
      end
      else if (srtKeyArr[srtId].key[j].dataType =
               GenSrtDType_C)
      then begin
        tkey^ := Chars(rec)[srtKeyArr[srtId].
                               key[j].offset];
        Inc(PtrRec(tkey).ofs,1)
      end
      else if (srtKeyArr[srtId].key[j].dataType =
               GenSrtDType_ST)
      then begin
        for k := 1 to srtKeyArr[srtId].key[j].length do
        begin
          tkey^ := Chars(rec)[srtKeyArr[srtId].
                                 key[j].offset+k];
          Inc(PtrRec(tkey).ofs,1)
        end
      end;
      if (srtKeyArr[srtId].key[j].order<>GenSrtOrder_A)
      then begin
```

```
        Descend(key^, srtKeyArr[srtId].key[j].length)
      end
    end;
    Move(rec, data^, srtStatArr[srtId].dataLen);
    GenBinInsert (srtRootArr[srtId], key,
                  srtStatArr[srtId].keyLen,
                    data, srtStatArr[srtId].dataLen);
    srtStatArr[srtId].nbrRec :=
                  srtStatArr[srtId].nbrRec + 1;
  end
  else
    GenSrtReleaseF := GenSrtErr_ISI
end;

procedure GenSrtDoSrt (srtId:word; var srtStatus:word);
begin
  srtStatus := GenSrtDoSrtF(srtId)
end;

function GenSrtDoSrtF (srtId : word) : word;
begin
  if ValidSrtId(srtId)
  then begin
    srtStatArr[srtId].srtState := GenSrtSt_RET;
    GenSrtDoSrtF := GenSrtErr_NM;
  end
  else GenSrtDoSrtF := GenSrtErr_ISI
end;

procedure GenSrtRetrieve (var rec; srtId : word;
                          var srtStatus: word);
begin
  srtStatus := GenSrtRetrieveF(rec, srtId)
end;

function GenSrtRetrieveF (var rec; srtId : word): word;
var d, k : dataPtr;
    dlen, klen : word;
begin
  if ValidSrtId(srtId) then
    if srtStatArr[srtId].srtState = GenSrtSt_RET then
      if srtStatArr[srtId].nbrRec > 0
      then begin
        GenBinRetDelSmRec(srtRootArr[srtId], k, klen,
                         d, dlen);
        Move(d^, rec, dlen);
        srtStatArr[srtId].nbrRec :=
                          srtStatArr[srtId].nbrRec - 1;
        GenSrtRetrieveF := GenSrtErr_NM;
      end
      else GenSrtretrieveF := GenSrtErr_ZR
    else GenSrtRetrieveF := GenSrtErr_ICO
  else GenSrtRetrieveF := GenSrtErr_ISI
end;

procedure GenSrtEnd (srtId: word; var srtStatus: word);
begin
  srtStatus := GenSrtEndF(srtId)
end;

function GenSrtEndF (srtId : word) : word;
var d, k : dataPtr;
    j, dlen, klen : word;
begin
  if ValidSrtId(srtId)
  then begin
    for j := 1 to srtStatArr[srtId].nbrRec do
      GenBinRetDelSmRec(srtRootArr[srtId], d, dlen,
                        k, klen);
    srtStatArr[srtId].nbrRec := 0;
    srtStatArr[srtId].srtState := GenSrtSt_NV
  end
  else GenSrtEndF := GenSrtErr_ISI
end;

procedure GenSrtStat (var srtStatus : SrtStatRec;
                      srtId :word);
begin
  srtStatus := srtStatArr[srtId]
end;

procedure GenSrtMsg (srtStatus : word;
                     var srtString : SrtStr);
begin
  if srtStatus <= MAX_MSG then
    srtString := SrtMsg[srtStatus + 1]
end;

begin
  for j := 1 to MAX_SRTS do
    srtStatArr[j].srtState := GenSrtSt_NV;
end.
```

◆ ◆ ◆

Long Horn SOG

Stuart Yarus called to say the Texas SOG has been reduced to a regular meeting of the Computer Council of Dallas. However, we're all invited and it's free.

Leave a message on Stuart's machine if you want travel and lodging information (or leave a message to let him know you're coming). Otherwise just show up. It'll be Saturday, July 14th 9 a.m. to 4 p.m. at the Infomart. If you'd like to speak, definitely call him. If you'd like to see 90 SIGs meet at once, then plan to go.

Infomart 1950 Stemmons FWY. Dallas 75207
Stuart Yarus 214-867-8012 (Weird hours)

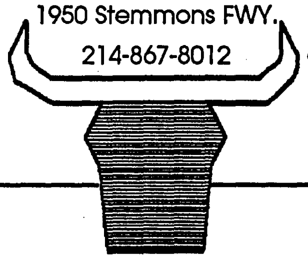
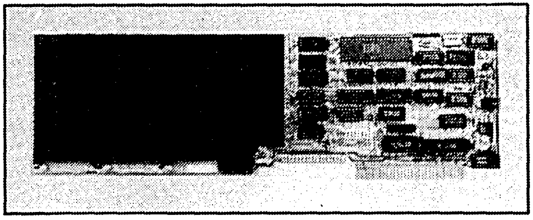
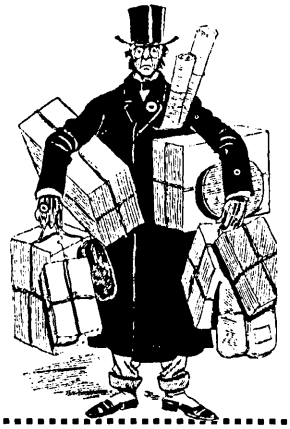


Figure 2—Binary Tree Unit

```
unit GenBinTree;
(* Author: Michael S. Hunt         Date: June 1, 1989
   This source code released into the public domain. *)
interface

type  dataPtr = ^data;
      data = char;
      treePtr = ^treeNode;
      treeNode = record
          llink, rlink : treePtr;
          data, key : dataPtr;
          datalen, keyLen : word
      end (* treeNode *);

procedure GenBinInsert (var node:treePtr; key:dataPtr;
        keyLen: word; data : dataPtr; dataLen : word);
procedure GenBinRetDelSmRec (var node : treePtr;
               var key : dataPtr; var keyLen : word;
               var data : dataPtr; var dataLen : word);
implementation

procedure GenBinInsert (var node : treePtr;
                        key : dataPtr; keyLen : word;
                        data : dataPtr; dataLen: word);
begin
  if node = NIL then
    begin
      new(node);
      node^.data := data;
      node^.dataLen := dataLen;
      node^.key := key;
      node^.keyLen := keyLen;
      node^.llink := NIL;
      node^.rlink := NIL
    end
  else if key^ < node^.key^ then
    GenBinInsert(node^.llink,key,keyLen,data,dataLen)
  else if key^ >= node^.key^ then
    GenBinInsert(node^.rlink,key,keyLen,data,dataLen)
end (* GenBinInsert *);

procedure GenBinRetDelSmRec (var node : treePtr;
               var key : dataPtr; var keyLen : word;
               var data : dataPtr; var dataLen : word);
var  delNode : treePtr;
begin
  if node^.llink = NIL then
    begin
      data := node^.data;
      dataLen := node^.dataLen;
      key := node^.key;
      keyLen := node^.keyLen;
      delNode := node;
      node := node^.rlink;
      Dispose(delNode);
    end
  else
    GenBinRetDelSmRec(node^.llink, key, keyLen,
                      data, dataLen)
end; (* GenBinRetDelSmRec *)
begin
end.
```

◆ ◆ ◆





Are You Moving?
Take Micro Cornucopia With You.

Please send us your new address at:
Micro Cornucopia
PO Box 223
Bend, OR 97709

Old Address
Name: ________________________
Street: ________________________
City: ________________________
State: ________________________
Zip Code: ________________________

New Address
Street: ________________________
City: ________________________
State: ________________________
Zip Code: ________________________

By William K. Rohwedder and Wayne L. Everhart

Northern Regional Research Laboratory
USDA-Agricultural Research Service
1815 University St.
Peoria, Illinois 61604

Interfacing Sixteen-Bit Devices

To The IBM AT Bus

Okay, so you got a real 286 machine. Now what're you going to do with it, huh? Hobble it with 8-bit I/O? Come on now, you can do better than that. A lot better.

With its fast processor and 16-bit data bus, the inexpensive AT-clone provides an excellent industrial and scientific tool for data measurement and control. The 16-bit interface is ideal for driving 16-bit digital-to-analog converters, reading 12-bit analog-to-digital converters, and communicating with other word-oriented devices.

Almost all personal computer interfacing is done in 8-bit bytes from the computer to printers, terminals, and modems. Publications like *Micro C* have thoroughly described these interfaces. But, little has been written about 16-bit interfaces.

Computer I/O

The machine language instructions IN AX,DX and OUT DX,AX control input and output of 16-bit data in AT computers. A MOV DX, (port address) instruction loads the port address into the computer's DX register.

During an OUT instruction, the computer loads the contents of AX (16 bits) onto the data bus, and the port address onto the address bus. The computer indicates to the interface that the data and port address are available by bringing the IOW bus line low.

Execution of an IN instruction is identical except that the data moves in the opposite direction, and the IOR line goes low. The interface must decode the port address, and the IOR or IOW lines; it must accept data from the data bus, or present data to the data bus during the active IOW or IOR.

Designing 16-bit interfaces is straightforward if you follow two rules: connect the I/O CS16 feedback (shown in Figure 1), and use only even-numbered ports.

Circuit Description

We used the inverters in the 74LS04 to generate logic one inputs to the 74LS30 NAND gate for the chosen port address. The correct I/O port address produces all ones at the 8 inputs to the NAND gate which in turn produces a logic zero at its output, pin 8. The decoding includes the AEN bus line to prevent DMA transfers from activating the interface.

The IBM Technical Reference for the AT (a lot of money for very little information) assigns port addresses hex 300 through hex 31F to their prototype card. This would appear to be a good place to put experimental interfaces. Hex 0 to hex 1F is 16 even-numbered input ports and 16 even-numbered output ports. The interface circuit in Figure 1 decodes only 4 input and 4 output ports.

Enabling the 74LS139 2-to-4 decoder chip requires a logic 0 at the 1G or 2G input. The 74LS32 OR gates supply this logic 0 only when the output of the 8-input NAND gate is low and either IOW or IOR is low, thus enabling the decoder.

We decode address lines A1 and A2 (along with IOR/IOW) to choose one of the eight devices. The eight outputs of the 74LS139 chip are normally high. Only the one that fits the address decoder goes low for the length of time the IOW or IOR goes low. The eight outputs connect to an 8-input NAND gate which enables the tri-state 74LS126 gate to pull the I/O CS16 line low, indicating to the computer that a 16-bit transfer is occurring.

This circuit uses only two of these outputs. The others are available for extra ports or as control pulses.

We use a pair of 74LS574 chips to latch the output from the computer, and a pair of 74LS244 tri-state bus drivers to control the input to the computer. In many designs, transceivers would be used here and the data lines would be multiplexed to handle both input and output on the same lines. (Of course, the data on these lines would probably have to be demultiplexed with a set of latches farther down the line.)

The 74LS574s latch data from the AT bus when they are clocked by the inverted IOW300 line, formed by decoding the IOW and the address lines. The OUTPUT CONTROL of these chips ties to ground so that the latched data is always available. The 74LS244 chips make the data on their input lines available to the computer bus when the IOR300 line pulls enable low.

Other octal tri-state bus drivers (74LS-240, 241, 373, 374, and 573) also could be used to latch the data.

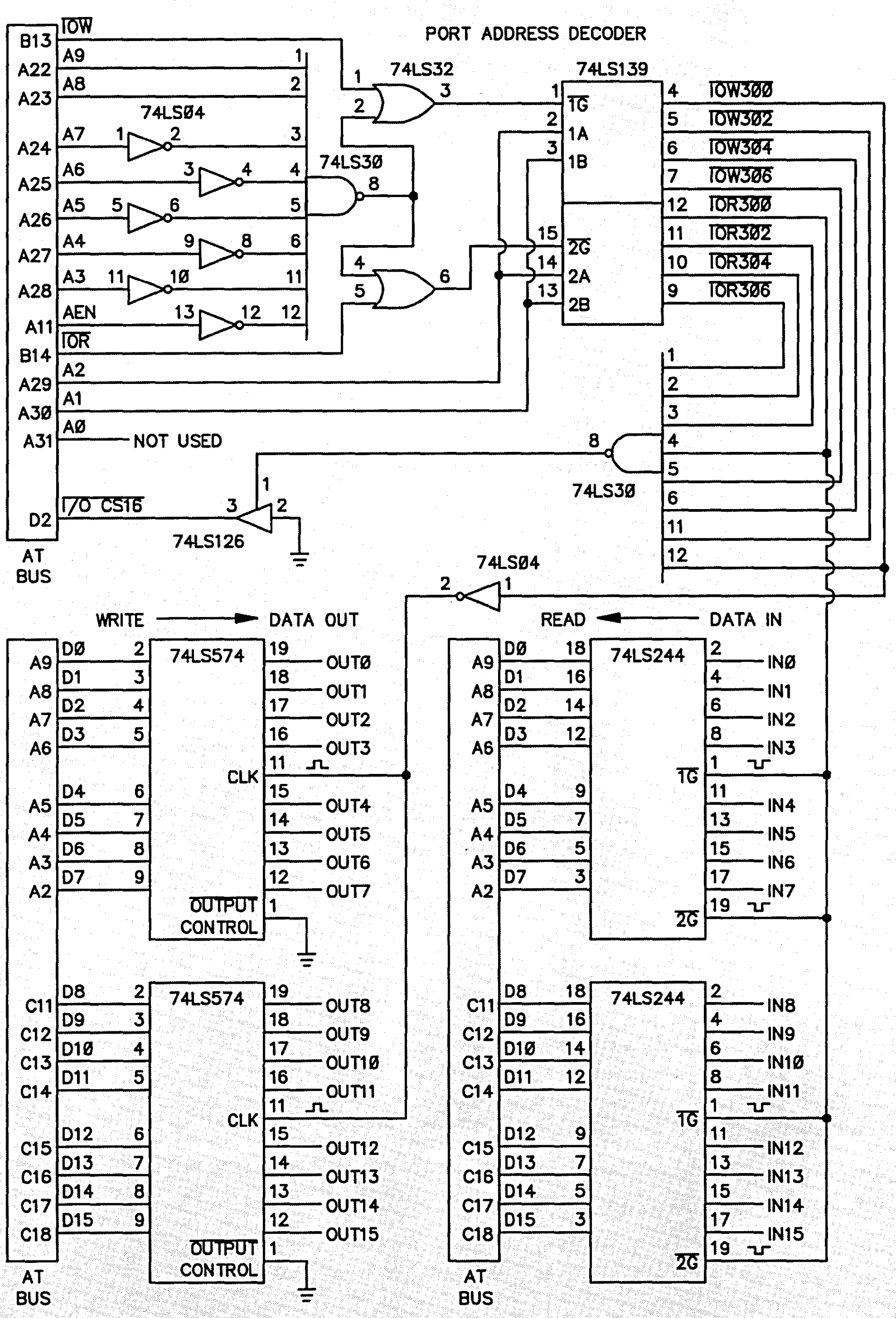
You can use the circuit in Figure 1 for 16-bit data transfers. Another set of input and output chips could be wired to IOW302 and IOR302 for control, to provide data ready, end of conversion, strobes, or handshaking capability. The IBM AT manual recommends "a maximum of two low-power Shotty (LS) loads per line." The outputs of tri-state chips in the high impedance state do not draw enough current to consider them as loads on the bus lines.

Construction

We wirewrapped the interface on a Vector Electronic Co. Model 4617-1 plugboard. The board comes with a paper layout of each side, a universal bracket, and an instruction sheet containing the AT bus pin names and locations.

We soldered the power and ground wires (not shown in Figure 1) to the wirewrap pins using 22-gage wire. At each chip we also soldered in a 0.033 μF bypass. Then we wirewrapped the signal

Figure 1—Circuit Diagram Of Sixteen-bit AT Interface.



lines point to point to reduce crosstalk.

I/O CS16

The IBM AT manual defines the I/O CS16 line as an input to the AT bus which "indicates to the system that the data transfer is a 16-bit, 1 wait-state, I/O cycle." The manual does not say why it's important, nor does it say what will happen if it isn't used.

Figure 2 contains oscilloscope patterns. The "A" group is the desired condition with I/O CS16 pulled low when selecting the address port. The IOW300, consisting of the decoded address lines and the IOW pulse, exactly follows the shape of the IOW pulse. Since the I/O CS16 line connects to the IOW300 line, it also follows the IOW pulse.

If I/O CS16 isn't connected, Figure 2B shows the strange things that happen. A single OUT DX,AX instruction produces two IOW pulses! IOW300 has a single pulse aligned with the first IOW pulse. IOW301 (we wired it up specially) has a single pulse aligned with the second IOW pulse.

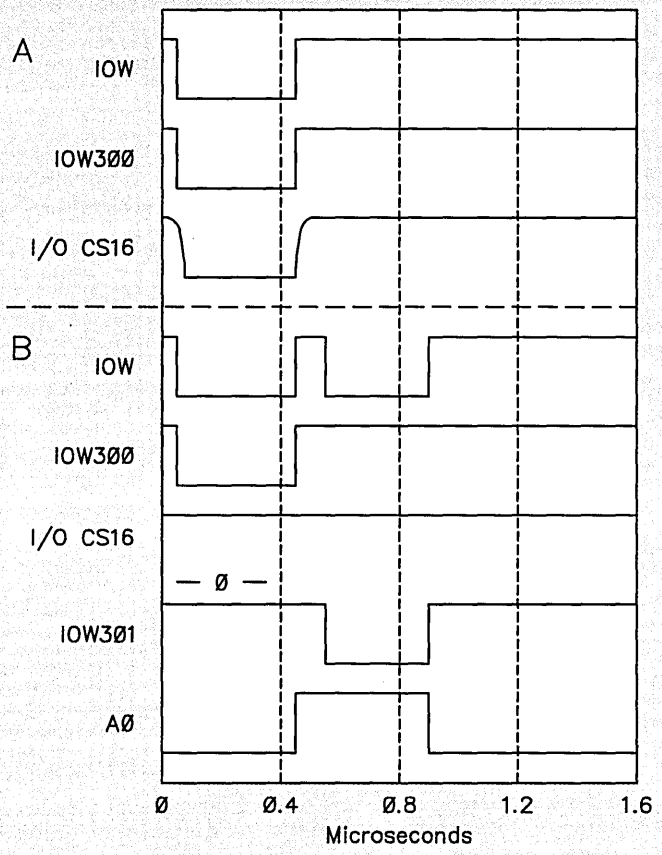
Address line A0 contains logic zero during the first IOW pulse and changes to logic one during the second. The computer is changing the address lines during the IOW cycle! The IOW301 pulse occurred with the second IOW pulse because the address lines had changed to address 301 and the IOW pulse was present. If one used these signals to control equipment, strange things would happen.

We connected fifteen of the data lines (OUT0 through OUT14), at the output of the 74LS574 latch, to a logic analyzer in order to follow the movement of data through the interface. We connected the IOW line to the 16th logic analyzer line and the logic analyzer clock to the 10 MHz AT clock CLK, bus pin B20.

The 10 MHz clock effectively makes the logic analyzer into a 16-channel storage oscilloscope with 2048 measured points, each 100 nanoseconds long. We set the output data word at hex CC55 so that the high-order byte could be recognized from the low-order byte. With the feedback connected to I/O CS16 and an instruction of OUT DX,AX, the hex CC55 transferred to the output of the chips with a single IOW pulse as expected.

With the I/O CS16 disconnected, Figure 2B shows the IOW line had two pulses. At the first IOW pulse, the 16-bit data transferred to the 16-bit output in proper order, both high and low bytes. At the second pulse, the high-order data transferred to the low-order byte of the output. The word went to the 8 low-order output lines in a byte serial procedure—55 followed by CC.

Figure 2—AT Bus Activity With and Without I/O CS16.



If the interface was wired to handle it, the byte serial transfer could be used to transfer a 16-bit word in two 8-bit bytes to the low-order output lines. The address line changes, as described earlier, could be used to clock two 8-bit latches. The logic analyzer showed CC in the high-order byte, during part of the second IOW pulse, but the data changed to FF part way through. Apparently some display terminal controllers used this 2-byte serial transfer to load 16-bit words into their 8-bit memories.

Port Addresses

Output to odd port addresses produces double-pulsed IOW lines, with or without feedback to the I/O CS16 line. These double pulses and the computer driven address changes complicate the operation of the decoder. It is simply best not to use odd port addresses.

Summary

At first glance, the double pulse and address line changes which occur on the AT bus during execution of the IN AX,DX or OUT DX,AX instructions appear strange. But these pulses could be used for data transfers of 16-bit words to 8-bit memories or data ports. You can easily achieve direct transfer of 16-bit words to 16-bit ports by pulling the I/O CS16 line low with a pulse generated by the port address decoder.

We have made five interface boards using circuits with both 74LS and 74ALS chips. We set up two of these boards with test programs. They ran through more than 5 billion transfers without an error.

◆ ◆ ◆

CP/M, NorthStar, Macintosh, Apple II, MS-DOS, and PS/2

Don't let incompatible diskette formats get you down – read them all with your PC!

Teach your PC to speak CP/M

UniDOS Z80 Coprocessor Board by MicroSolutions

Run your Z80 and 8080 code programs at LIGHTNING speed on your PC or AT with the UniDOS 8MHz. Z80 coprocessor board. UniDOS automatically switches from MS-DOS to CP/M mode when your CP/M program is executed. UniDOS emulates most common computers and terminals such as Kaypro, Xerox 820, Morrow, Osborne, and VT100. All standard CP/M system calls are supported. Includes UniDOS and UniForm-PC.

UniDOS Z80 Coprocessor Card	$ 169.95

UniDOS by Micro Solutions

Equip your PC/XT with an NEC V20 chip and run your favorite CP/M programs without taking up another card slot. Runs 8080 code directly on the V20, and uses emulation mode for Z80 code or systems without a V20.

UniDOS by MicroSolutions	$ 64.95
UniDOS w/UniForm & V20-8 chip	$ 135.00

UniForm-PC by MicroSolutions

How have you ever wished you could use your CP/M diskettes on your PC? Now you can access your CP/M files and programs on your MS-DOS computer just as you would a standard MS-DOS diskette. Install UniForm and use standard DOS commands and programs right on your original diskette without modifying or copying your files. UniForm-PC allows you to read, write, format, and copy diskettes from over 275 CP/M and MS-DOS computers on your PC, XT, or AT. With UniForm-PC and the Compaticard, you can use 5¼" high density, 96TPI, 3½" (720k/1.44M), and even 8" drives.

UniForm-PC by MicroSolutions	$ 64.95

Also available for Kaypro, & other CP/M computers

CompatiCard by MicroSolutions

THE universal four drive floppy controller board for the PC or AT. Run up to 16 disk drives (4 per CompatiCard), including standard 360K, 96 TPI, high density 1.2M, 8" (SSSD or DSDD), and 720k/1.44M 3½" drives. Comes with its own MS-DOS driver and format program. Use it with UniForm-PC for maximum versitility.

CompatiCard Board	$ 119.95
CompatiCard with UniForm-PC	$ 179.95
8" Drive adaptor	$ 15.00
External 5¼" drive cable	$ 15.00

Compaticard II by MicroSolutions

Two drive version of the CompatiCard, sorry no 8" or single density.

CompatiCard II	$ 89.95
CompatiCard II with 1.2M or 3½" internal drive	$ 199.95

Megamate by MicroSolutions

This is the 3½" drive package that you've been waiting for. Run 720k or 1.44M diskettes in this attractive external drive. Comes complete with its own controller card. Easy to install, just plug it into your PC or AT and go.

Megamate	$ 329.95

MatchPoint-PC by MicroSolutions

The MatchPoint-PC board for the PC/XT/AT works with your standard controller card to let you read and write to NorthStar hard sector and Apple II diskettes on your PC. INCLUDES a copy of the UniForm-PC program, as well as utilities to format disks, copy, delete, and view files on Apple DOS, PRODOS, and Apple CP/M diskettes.

MatchPoint-PC Board	$179.95

MatchMaker by MicroSolutions

Now you can copy your Macintosh diskettes right on your PC/XT/AT with the MatchMaker. Just plug your external 3½" Macintosh drive into the MatchMaker board and experience EASY access to your Mac diskettes. Includes programs to read, write, initialize, and delete files on your single or double sided Mac diskettes.

MatchMaker Board	$ 139.95
MatchMaker w/External Mac Drive	$ 325.00

Hard Disks for CP/M systems

Pep up your CP/M computer with hard disk performance. Our simple to install kits allow you to connect up to two 5¼" hard drives to your Z80 system. The Winchester Connection software customizes your system from an easy to use menu, with flexible drive parameters, partition and block size, and includes complete installation and diagnostic utilities. A complete system requires a HDS daughter board, WD1002-05 hard drive controller board, hard drive, software package and cables.

HDS Host Board with Software	$ 79.95
HDS Board, WD1002-05, and software	$ 245.00
WD1002-05 Controller Board only	$ 185.00
External drive cabinet with power supply	$ 139.95

Parts and accessories for the Kaypro and Xerox 820-1

Plus2 ROM Set for Xerox 820-1	$ 39.95
Plus2 ROM with X120 bare board	$ 49.95
KayPLUS ROM for Kaypro 2, 4, 10 - specify	$ 69.95
Kaypro 2X Real-time Clock parts kit	$ 29.00
Kaypro 2X Hard disk interface parts kit	$ 16.00
Kaypro 10 Hard Disk controller board	$ 185.00
Kaypro four drive floppy decoder board	$ 35.00
QP/M Operating System - bootable - specify system	$ 64.95
QP/M without CBIOS (installs on any Z80 system)	$ 49.95

Complete parts and repair services available

Special Purchases!!

PC-Mastercard by Magnum Computer

This is probably the BEST multi-function card on the market. Use mixed banks of 64k and 256k chips to install up to 1.5 Megabytes of RAMDISK, and PRINT SPOOLER (or fill your system up to 640k). Serial, parallel, game ports, and real time clock installed! Comes with complete software.

PC-MASTERCARD (0k installed)	$ 69.95

Turbo Editor Toolbox by Borland International ... $ 29.95

Ever wanted to add text editing to your Turbo Pascal application, or write a word processor that does things the way that YOU want? Comes with source for two sample editors, modules for windowing, multi-tasking, and many other options. Requires PC or compatible and Turbo Pascal 3.0.

COPY II PC by Central Point Software ..... $ 24.95

Stop worrying about your copy protected disks. COPY II PC lets you back them up, so you can keep going when your master disk can't.

Printer/Data Switches

Quality with economy. These boxes switch all 25 lines so they can be used with either RS232, or IBM parallel (DB25) printer cables.

Four port data switch	$ 39.95
Two port data switch	$ 34.95
IBM style Parallel Printer Cable	$ 12.00
Three cable set ★★ Special ★★	$ 30.00

MicroPro Manuals

WordStar V3.3 Manual	$ 12.00
InfoStar Set (DataStar & ReportStar)	$ 18.00

Call or write for our complete catalog of software, parts, accessories and complete repair services for the Kaypro, Xerox 820, and IBM PC/AT.

Prices subject to change without notice. VISA and Mastercard accepted. Include $6.00 shipping and handling, $8.50 for COD, UPS-Blue or RED Label additional. Please include your phone number with all correspondence.

EMERALD MICROWARE

(503) 641-8088

P.O. Box 1726 • Beaverton, OR 97075

Reader Service Number 10

TIDBITS

Pascal + Objects—

A Good Lookin' Turbo Pascal 5.5 & An I/O Error Handlin' Object

By Gary Entsminger
P.O. Box 2091
Davis, CA 95617

First it was structures, now it's objects. What are objects? The latest buzz word? The ultimate way to write Pascal? The best and worst of self-modifying code? The reason you have to buy a whole new set of libraries?

Yep. That about covers it.

If you've been trying to wade through the object-oriented muck, and know how to program in Turbo Pascal, you can now take a leisurely step (up, I believe) into the powerful world of object-oriented programming.

OOP, object-oriented programming's acronym, has suddenly become a buzz term, engulfed in hype. But try not to let that bother you. Objects are a very powerful way of conceptualizing programs. Turbo Pascal 5.5 is a great way for Pascal programmers to bypass the buzz and get down to thinkin' out objects.

In short (Pascal terms)—objects are records that hold not only data fields, but the procedures and functions to manipulate the data fields as well. In OOP terms, we call these procedures and functions "methods."

In addition, objects can inherit data fields and methods from each other. So you can build complex objects (easily and quickly) from simpler ones.

Third, objects can be polymorphic—just like their ancestors in some ways and different in others. A new object created from an existing object can use some of the methods it inherits while reimplementing others.

In this installment of Tidbits, I'll describe the key features of object-oriented programming (from a Turbo Pascal perspective). Then, I'll build a simple object, an I/O error-handler, to illustrate these features.

If you're already familiar with object-oriented programming languages like C++ and Smalltalk, it might help to know that Turbo Pascal objects are very much like those of C++. A little less macho, perhaps, but powerfully featured.

Turbo Pascal 5.5

TP 5.5 is TP 5.0 plus objects. (Actually, 5.5 has several other important improvements, but I'll leave their details to the Borland advertisements and my favorite *Dr. Dobbs* columnist.) The big reason for getting your hands on 5.5 is the new type—object.

Objects have lives and a language of their own, and it's worth spending a few paragraphs getting to know the language.

Three main properties characterize an object-oriented programming language—

- encapsulation
- inheritance
- polymorphism

Encapsulation

I repeat ('tho slightly differently)—From a Turbo Pascal perspective, encapsulation means you can combine a record with the procedures and functions which will manipulate the record. The new data type formed by combining data (a record) with code (procedures and functions) is an object.

Declare a record like this—

```
type

record1 = aRecord;

aRecord = record
  Name : string;
  X,Y : integer;
end;
```

where X, Y, and Name are data fields.

Declare an object like this—

```
type

object1 = anObject;

anObject = object
  X,Y : integer;
  procedure doSomethingWithX&Y;
end;
```

X and Y are still data fields and the procedure doSomethingWithX&Y is a method for manipulating anObject.

An object consists of everything we know about it. Data (which define the variables of the object) and code (which manipulates the object) are together in one box: a box that can be both black and white at once (i.e., polymorphic, but more on that later.)

To use a record, we access its data via "dotting" or via a With statement—

```
record1.X := 2;
record1.Y := 3;
```

or—

```
With record1 do
  begin
    X:= 2;
    Y:= 4;
  end;
```

We use the same methods to manipulate objects (dotting or a With statement)—

```
object1.X := 2;

With object1 do
  begin
    X:= 2;
    Y:= 3;
    doSomethingWithX&Y;
  end;
```

Think of an object as a descendent type (object) that first inherits all the power of an ancestor type (record) and then extends itself with methods—

```
anyObject = object(record)
  methods;
end;
```

If a record is a container for data fields, then an object is a container for data fields and methods.

Notice that we don't have to pass X and Y to doSomethingWithX&Y. doSomethingWithX&Y already knows about X and Y because they're both contained within the same object (i.e., they share a scope). Everything in an object (methods and data) is shared. It's a small town with a gossip—everything knows about everything else in the object.

This combining of data and code in one container is a neat extension of units. Programs consist of units. Units consist of objects. So at one level, an object is a new way to organize tools. But that's just the beginning. More important, it's a new way of thinking about tools. Let's take another step.

Inheritance

Once you've defined an object, you can use it to build a hierarchy of objects. Each time you build a new object (called a descendent) from an existing object (called an ancestor), the new object inherits all its ancestor's code and data.

An object can have more than one ancestor, but only one immediate ancestor. So an object hierarchy might look like this—

```
object1  — object2
         |
         — object3    — object4
                      |
                      — object5
```

In this little sketch, object1 has two descendents, object2 and object3. object3 has two descendents, object4 and object5. object4, for example, inherits everything from object3 (its immediate ancestor) and everything from object1 (an earlier ancestor). This can go on ad infinitum (or until you run out of memory, disk space, or ideas).

In an important sense, object-oriented programming is a method for building family trees for data structures.

Let's say we've built an object—

```
type

anObject1 = object
  X,Y: integer;
  procedure doSomethingWithX&Y;
end;
```

and now we want to build anObject2 out of anObject1.

We want anObject2 to be identical to anObject1, except we want to extend it by adding a procedure called doSomethingElse.

We build anObject2 simply and quickly by inheriting (from anObject1)—

```
anObject2 = object (anObject1)
  procedure doSomethingElse;
end;
```

Since anObject2 inherits access to all the data and methods of anObject1, we don't need to redeclare anObject1's data fields and methods. We simply add the new method.

Now we can doSomethingWithX&Y and doSomethingElse to anObject2—

```
type

object2 = anObject2;

With object2 do
  begin
    X:= 5;
    Y:= 8;
    doSomethingWithX&Y;
    doSomethingElse;
  end;
```

Notice (again) that we don't have to pass the variables X and Y to doSomethingElse; it inherited knowledge of X and Y along with everything else (from object1).

Inheritance lets us build very complex objects without repeating any code. The new object simply inherits whatever it needs from an ancestor. But that's only the beginning (haven't we heard that before?)!

More important, the new object can use as much or as little of its ancestors' code as it needs. In fact, it can reimplement any method it chooses. We call this reimplementation of code polymorphism, and in it lies incredibly beautiful (and subtle) power.

Polymorphism

Polymorphism means that a method can have one name that's shared up and down an object hierarchy. Each object can (if it chooses) reimplement the method.

For example, suppose we want to define anObject3 which inherits the data and methods from anObject2 and in addition reimplements the procedure doSomethingElse. We define it like so—

```
anObject3 = object (anObject2)
  procedure doSomethingElse;
end;
```

Now when we use a variable of type anObject3, it will inherit the data fields from anObject2 (which anObject2 inherited from anObject1), but will use its own doSomethingElse procedure.

In one sense, a descendent (object) is both itself and all its ancestors simultaneously. *And simultaneously*, it's different from all of them by 0, 1, or more methods.

The trick for the compiler is to determine when to call one descendent's method and not another's.

When we call (or use) a method from a descendent object, the compiler looks first to see if the current object has the method we're calling. If it finds the method in the current object, it uses it and then moves on to the next instruction.

If the current object doesn't contain the method, the compiler backtracks up the hierarchy through ancestor objects until it finds a method with the correct name. When it finds it, it executes it, stops looking, and moves on to the next instruction. If it doesn't find the method in any ancestor, it reports an error.

If you're a Prolog programmer, this backtracking scenario should sound fa-

Figure 1—Objects in Turbo Pascal

```
unit myIOhandler;

interface

type

Prompt = string[80];

myIOhandler = object
  IOErr : boolean;
  IOCode: integer;
  constructor init;
  procedure report(Msg : Prompt); virtual;
  procedure IOCheck; virtual;
end;

implementation

uses
  DOS,
  Unit_win,
  Crt;

constructor myIOhandler.init;
begin {this is all we have to do: TP handles the work.}
end;

procedure myIOhandler.IOCheck;
begin
  IOCode := IOResult;
  IOErr := (IOCode <> 0);
  if IOErr then
    begin
     case IOCode of
       $02 : report('File not found');
       $03 : report('Path not found');
       $04 : report('File not open');
       $10 : report('Error in numeric format');
       $20 : report('Operation not allowed on logical device');
       $21 : report('Not allowed in direct mode');
       $22 : report('Assign to standard files not allowed');
       $90 : report('Record length mismatch');
       $91 : report('Seek beyond EOF');
       $99 : report('Unexpected EOF');
       $F0 : report('Disk write error');
       $F1 : report('Directory is full');
       $F2 : report('File size overflow');
       $F3 : report('Too many open files');
       $FF : report('File disappeared')
       else
         report('Unknown I/O error: ');
         write(IOCode:3);
    end {case}
  end {if}
end; {IOCheck}

procedure myIOhandler.report(Msg : Prompt);

var
   F : text;
   FileName, P : string;
   W1 : _WindowPtr;
   Window_status : boolean;
   C : char;

begin
  W1 := __MakeWin(3,5,74,18,22,20,Black,LightGray,
       _DLBORD_WIN,LightRed,Black);
  Window_status := __DispWin(W1);
  writeln(Msg);
  C := readkey;
  Window_status := __RemWin;
end;
```

◆ ◆ ◆

miliar, and at this juncture (polymorphism) object-oriented programming *really* (I mean it) *gets interesting.*

Static & Virtual

Each object I've created so far uses "static" methods. When we call a static method, the compiler searches for methods one way—by backtracking up the hierarchy. A fine and dandy way to search for methods unless we create methods that call other methods, and we decide to reimplement some of these other methods.

Once the compiler has backtracked to an ancestor object to find a method, it only uses methods from that ancestor (or its ancestors), *if the method is static.*

We need a way to tell the compiler to remember which object *was current,* before it started backtracking. Then, finding one method in an ancestor, the compiler could look for the next method beginning with the calling object or descendent instead of its ancestor.

Static methods (like static variables) are static because the compiler allocates and resolves all references to them at compile time. What we're asking the compiler to do is to resolve some references at runtime.

To resolve references to methods at runtime, we create "virtual" methods by adding the key word virtual to the method and by including a special procedure called a constructor within the object. But there's a catch. A method can't be both static (in one object) and virtual in another. You must anticipate, by declaring these methods virtual from the beginning.

For example, doSomethingElse must be declared virtual at its earliest and all subsequent declarations—

```
anObject2 = object(anObject1)
  constructor init;
  procedure doSomethingElse;virtual;
end;

anObject4 = object(anObject3)
  constructor init;
  procedure doSomethingElse;virtual;
end;
```

The virtual method effectively changes the order in which the compiler determines which method to use when methods share names. This is a thorny (but quite powerful) aspect of object-oriented programming. I'll spend the rest of Tidbits implementing a simple but useful object—an I/O error handler, which combines all three principal properties of objects.

Object: myIOhandler

Let's define an object—

```
myIOhandler = object
  IOErr : boolean;
  IOCode: integer;
  Message: string;
  constructor init;
  procedure report; virtual;
  procedure IOCheck; virtual;
end;
```

This object will check for I/O errors and report the type of errors (if any) it discovers. It consists of three methods (init, report, and IOCheck) and three data fields. Notice that report and IOCheck are virtual methods. This means I'm implementing them my way now (so you can use them as a black box); and simultaneously, I'm giving you the chance to change them later (if you want or need to).

IOCheck calls IOResult (a standard Turbo Pascal function) to determine if there's been an error. If so (if IOResult <> 0), IOCheck calls report to report the specific error.

myIOhandler's method, report, (see Figure 1) uses a Blaise Toolbox procedure to write a message in a window. IOCheck calls report if it finds an error.

Now, suppose you're satisfied with IOCheck, but don't want to incur the overhead of creating and destroying a window. You can create your own object (yourIOhandler) which inherits IOCheck but not report. Because report is virtual, the compiler knows to keep track of the *current* object (the one calling IOCheck).

Then when IOCheck calls report, the compiler will begin searching for report by backtracking from yourIOhandler (the descendent), not from the ancestor (myIOhandler) where it found IOCheck.

First, let's look at your object (yourIOhandler) which consists of everything from myIOhandler except your new report—

```
yourIOhandler = object(myIOhandler)
  constructor init;
  procedure report; virtual;
end;
```

Now recall the order in which the compiler looks for methods when methods are static. Let's say you call IOCheck from yourIOhandler. Since IOCheck is an inherited method, the compiler finds it in an ancestor, myIOhandler, and (effectively) executes IOCheck there.

Now IOCheck (see Figure 1) calls report. If report is static, the compiler uses the report in myIOhandler.

If report is virtual, the compiler remembers that yourIOhandler called IOCheck and begins backtracking from yourIOhandler. So it will execute report in yourIOhandler, not myIOhandler.

Methods by default are static, so we effectively override this default by using the keyword, virtual.

Whew!

Virtual methods, as you should begin to expect, *positively get interesting* about here. That's where I'll bail out for this issue.

Thanks to Borland International for letting me have an early look at Turbo Pascal 5.5 and Turbo Debugger 1.5, which includes debugger support for objects.

Thanks to Blaise Computing for letting me use their terrific Turbo Pascal Toolbox, now recompiled for version 5.5. Power Tools Plus includes several tools for generating windows, menus, and TSRs. It's a timesaver deluxe for only $99.

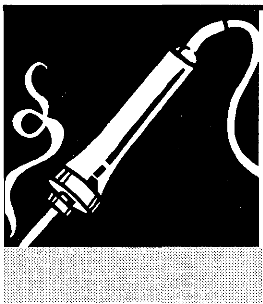
Blaise Computing Inc.
2560 Ninth St., Ste. 316
Berkeley, CA 94710
(415) 540-5441

Also, thanks to Osborne/McGraw-Hill who furnished me a galley of the object-oriented chapter from Stephen O'Brien's new version of the *Complete Turbo Pascal Reference.* O'Brien develops a neat little airplane object which illustrates the basics of extending Turbo Pascal with objects. Recommended for getting your feet wet.

Finally, thanks to Bruce Eckel for stimulating conversations in the Elk mountains. If you want to leap into the depths of object-oriented programming C-style, check out his book, *Using C++* (also from Osborne/McGraw-Hill). Bruce really knows objects.

And that, friends, is Tidbits.

◆ ◆ ◆



TECHTIPS

Batteries, Disks, And Drives

T1000 Battery Upgrade

The Toshiba T1000 contains 4 "sub-C" batteries. On a good day they yield 5 hours running time (assuming no disk accesses). Periphex sells a set of 4 high capacity C cells that will power the T1000 for up to 10 hours, after a 16-18 hour recharge. You don't need to be a brain surgeon to mount these batteries, but you should be careful. You'll need:

1. The four Periphex C cells. The C cells, shrink-wrapped into a block with two leads welded in place, cost about $35. If you ask Periphex, they'll ship the battery pack with wires already on the tabs. If you call and ask for the battery pack for the T1000 modification, they'll know just what you want. They even send instructions and a proper piece of cushioning foam right along with the nicad pack!
2. A soldering iron or gun (about 45 watts).
3. Two female crimp-on connectors, commonly used to connect to blade-type auto electrical systems. When you look at the end of these connectors, you see a shape like this:

```
(___)
```

Radio Shack sells a package of eight female quick-disconnects (64-3039) for just 99¢.

The Toshiba batteries mount in a plastic tray that takes up a lot of room in the battery compartment of the machine. C cells are slightly bigger than sub-Cs, so you can't use the plastic tray. Keep the old batteries and tray in case of emergencies—for example, if you must return your T1000 to Toshiba for service.

Toshiba batteries come with two red wires on one corner and two black wires on the other end. Match this arrangement if you're soldering the wires to the battery pack yourself. Use stranded wire, about eight inches long.

Why two leads from each pole? Because the design allows for separate circuits between the battery and the computer, and the battery and the charger. However, as currently configured, the computer doesn't take advantage of that separation. The connector that attaches the batteries to the motherboard is hard to come by, so you'll have to improvise. Fortunately, the male side of the connector is on the PC board.

Before you remove the female connector:

1. Back up your D: drive (if you have a RAM drive card) to a floppy, because as soon as you disconnect the battery, you'll lose all files and D: will have to be reformatted from the C: drive (the ROM chip) before it can be used again.
2. Note carefully which two pins have the red (positive) wires attached, and which have the black (negative) wires.

The configuration of the pins will look like this:

```
[ *  *  *  * ]
  B  B  R  R


(B=Black)      (R=Red)
```

After you solder (or crimp) a quick-disconnect to the black and red wires, you'll slip each over the appropriate pair of male pins. You should pick the female disconnects that most closely match the spacing of the pins, then adjust them with a small screwdriver blade until they fit smoothly over a pair of pins.

Solder (or crimp) those connectors on the wires. Then, wrap each with electrical tape to keep them from touching as you push them onto the pins.

The physical mounting of the pack is up to you. Some people have used peel-and-stick velcro (also available at Radio Shack) to stick the battery pack to the "ceiling" of the case, and a thin (¼") piece of plastic-tape-covered rubber foam under the pack to protect the motherboard. I used a thin 1⁄16" aluminum strip about 1" wide and 5" long and placed one end under the modem support and the other under the old battery pack holder. Thin foam under the nicad pack protects the motherboard.

That's it. Carefully reassemble the two halves of the case (don't pinch the flat cable that connects the LCD screen to the motherboard). Reformat the D drive and run the batteries down to shutdown, then charge them for at least 24 hours with the original charger. After that, you should get 8-10 hours of total life between charges. Ordinarily, you should double the recharge time to about 16 hours.

Periphex Inc.
149 Palmer Rd.
Southbury, CT 06488
(203) 264-3985

Guruka Singh Khalsa
Sierra On-Line
P.O. Box 485
Coarsegold, CA 93644
voice: (209) 683-4468
data: (209) 683-2084
CIS ID 71500,34

Traveling With Computers And Disks

What's the effect of airport security on disks? Will the system erase or damage data? I've heard people answer both yes and no, but no one could say why. So I decided to do a little research.

I started this quest by contacting the FAA (Federal Aviation Administration). There I spoke to Dr. Lyle Malotky, who explained that airport metal detectors use less than 1 gauss of electromagnetic field strength. The earth has a natural

Micro Ads

A Micro Ad is the inexpensive way to reach over 22,000 technical folks like yourself. To place a Micro Ad, just print out your message (make it short and sweet) and mail it to Micro C. We'll typeset your ad (no charge) and run it in the next available issue. You can also send camera ready copy. Rates: $99 for 1 time, $267 for three times, $474 for 6 times (a best buy at only $79 per insertion). Full payment must accompany ad. Each ad space is 2 1/4 inches by 1 3/4 inches.

DUPLICATOR TOOLKIT PRO

THE DUPLICATOR TOOLKIT PROfessional is a superfast diskette duplication utility for your PC. It does what diskcopy can't do! Copies a 360K diskette in 20 seconds, formats and verifies "on the fly", and even produces diskette labels at the same time! Copies 5 1/4 360K DDDS, 1.2M high-density, 5 1/4 360K using a 1.2M high density drive, 3 1/2 720k, 3 1/2 1.44M and 3 1/2 720K using a 1.44M drive. Stores master in RAM or on the hard drive. Format-only option allows for high-speed formatting. Eliminates the "floppy shuffle"!

$129.00 + 5.00 s/h Visa/MC/Cod
COPY TECHNOLOGIES
14252 Culver Dr. Ste. 323
Irvine, CA 92714
(714) 975-1477

Reader Service Number 163

F L A S H

THE DISK ACCELERATOR

Add logo **

- EASY to Install
- Cache up to 32 MEGS of EXTENDED and or EXPANDED
- Buffers up to 26 DEVICE driven drives
- Comes with 2 FREE utilities!!!!!

ORDER NOW $69.95
(800) 25-FLASH

SOFTWARE MASTERS 6352 North Guilford Ave.
Indianapolis, In 46220 / (317) 253-8088
$5.00 Shp/hnd in USA & CANADA, $15.00 overseas.

Reader Service Number 106

LATEST AWARD BIOS
PC/XT ☆ 286 ☆ 386

Support for:
- ✧ Enhanced Keyboards
- ✧ EGA & VGA Graphics
- ✧ 3.5 inch Floppies
- ✧ More...

Authorized AWARD Distributor
(800) 423-3400

KOMPUTERWERK, INC
851 Parkview Blvd
Pittsburgh, PA 15215

Reader Service Number 126

STOCKS OPTIONS FUTURES

Turn Your PC Into A
MARKET QUOTATION MONITOR

100 page book covers satellite and radio data reception of financial news and quotes for your PC. $19 (includes demo diskette). Free informative catalog of:

- * Data receivers and kits
- * Quote processing and display software
- * Descrambling software utilities

303-223-2120 **$5 Demo Diskette**

DATArx
111 E. Drake Rd. Suite 7041
Fort Collins, CO 80525

Reader Service Number 133

Mr. MOX™ $99⁹⁵

by Epoch Data

Modem operated power controller for your PC.

Makes any PC with external modem remote-accessable!

Software included.

Order direct from:

Epoch Data
P.O. Box 1093
Cardiff, CA 92007
(619) 543-9423

Reader Service Number 135

Why you want BATCOM!

BATCOM is a batch file compiler that compiles your ".bat" files to ".exe" files to make them faster, more professional, and more capable. **BATCOM** extends DOS with new commands so you can read keyboard input, perform arithmetic, use subroutines, and much more. In addition, **BATCOM** protects your source code, and you can distribute your compiled programs without royalties. For IBM PC. **$49.95.**

Wenham Software Company
5 Burley St.
Wenham, Ma. 01984
(508)-774-7036 **FREE catalog.**

Reader Service Number 124

ScreenLib™

Complete C Source Code—No Royalties!

Simple Screen Definition ★ Windows
Pop-up Menus ★ Context-Sensitive Help
Lots of low-level functions

Introductory Price: $69.95
Plus $5.00 shipping ($10.00 outside US)
CA residents please add sales tax

Business Computer Services
1800 S. Robertson Blvd., Suite 206
Los Angeles, CA 90035
VISA/MC orders: (213) 836–5026
Demo disk available–$10.00

Reader Service Number 155

8031 µController Module $39.95
Shipping: $3.00 US/$5.00 Canada

Ideal for prototypes, one of a kind devices or short production runs. Perfect building block for PC or Macintosh data aquisition interface or stand alone control projects. Assembled and tested board includes 8031 microprocessor, crystal, 8K of EPROM, 128 bytes of RAM, 2 byte-wide I/O ports and provisions for a MAX232 to provide industry compatible serial I/O. All ICs are socketed and I/O is via a 2X17 header. Size: 2.75" by 4.0". OEM discounts.

Cottage Resources
Suite 3-672C, 1405 Stevenson Drive
Springfield, Illinois 62703
(217) 529-7679

Reader Service Number 158

The $25 Network

Try the 1st truly low cost LAN

- Connect 2 or 3 PCs, XTs, ATs
- Uses serial ports and 5 wire cable
- Runs at 115 K baud
- Runs in background, totally transparent
- share any device, any file
- Needs only 14K of ram

Skeptical? We make believers!

Information Modes
P.O. Drawer F
Denton, TX 76202
817-387-3339

Reader Service Number 149

WHITNEY EDITOR $39

Small and fast
Uses all available memory
Split-screen editing
Configurable keyboard
Regular expression search
One key compile
Features for writing documentation
Condensed/Outline display
Runs on IBM PC's, AT's, and PS/2's

USA shipping & handling $3; Outside USA $15
CA residents add sales tax

Whitney Software, Inc.
P.O.Box 4999, Walnut Creek, CA 94596 (415) 933-9019

Reader Service Number 164

MINI 386-20/25 MOTHERBOARD

- 20/25 MHz 80386 CPU • On Board Memory Expandable 1MB/2MB/4MB/8MB • Fully PC/AT Compatible
- Phoenix ROM BIOS PLUS with ROM-Based SET-UP
- RESET, TURBO LED Jumpers
- Battery Back-Up System Clock/Calendar and CMOS Ram
- Supports 80387 Math-Coprocessor Asynchronously
- Interleaved Memory Access for Zero Wait-State On-Board Operation • Exact-XT Dimension and AT Mounting Holes
- 3 8-bit, 4 16-bit and 1-32-bit expansion slots
- Surface Mounting Technology Improves Reliability
- Support Diskless Network Application
- NOVELL, OS/2 Compatible

SPECIAL $649/$949

McTEK SYSTEMS, INC.
1521 SAN PABLO AVENUE
BERKELEY, CALIFORNIA 94702
(415) 525-5129

Reader Service Number 42

CROSS ASSEMBLERS

PseudoCode releases the PseudoSam professional series of Cross assemblers. All popular processors. Macros, Conditional Assembly, and Include Files. Virtually unlimited size. For IBM-PC's MS-DOS 2.0 or greater with manual $50.00. Simulators and disassemblers also available. (MI res. 4% tax). S&H USA $5, Canada $10, Foreign $15. Visa/MC.

KORE Inc.
6910 Patterson S.E.
Caledonia, MI 49316 616-887-1444.

30 Day satisfaction guaranteed
or purchase price refunded.

Reader Service Number 136

field of 0.7 gauss and normal magnetic media (mag strips, tapes, and disks) require 200 to 300 gauss to write data.

Airport X-ray units use less than 1 mr (millirad). If the X-ray unit was built in the last five years, it will use 0.3 mr per scan. In every day living you're exposed to 0.3 mr per day, and when flying at normal altitudes (about 35,000 feet for commercial airlines) you get 0.5 to 1.0 mr per hour.

The only X-ray unit that uses 1.0 mr or more is the fluoroscope, which has to have a stationary object to produce an image on the screen. The longer it's viewed, the more rads the object absorbs. However, at present fluoroscopes are not in use at any major U.S. airport.

Now what does all this mean? There's no need to worry about the security equipment at the airport erasing your disks. Just to satisfy my own curiosity, I carried some disks through the metal detector and sent others through the X-ray machine at Dulles airport. I then tested the disks and guess what: the data was fine.

Now for the bad news. If you travel overseas, there's a possibility that they'll use different equipment with higher dosages. That means there's a possibility your disks could be erased. But don't worry. All airports will allow you to have items inspected visually instead of by X-ray. The only item that should never be X-rayed is photo film with an ASA of 1000 or higher.

Here comes the worst news of all. The biggest potential for data loss during air travel comes from vibrations and static electricity. Under most circumstances your disks will not suffer damage if they're packed tightly and are well padded. But static electricity can zap your disks in a heartbeat.

At 35,000 feet, air temperature is approximately -40°F with only 3% humidity. That air is taken in, compressed, and warmed up which brings the humidity down to 1%. With the low humidity, there's a lot of static electricity.

Simply moving around the plane will generate more. So, keep your hands off your disks during flight.

The bottom line is that U.S. airport security systems will not damage your disks. Using a laptop computer or touching a disk while flying can be dangerous. If you're traveling overseas with disks or a laptop, have them visually inspected.

Figure 1—Find Number of Drives in System

```
(* PROGRAM DRV_INFO.PAS    AUTHOR - JEFFREY SCOTT DONOVAN           *)

program drv_info;
uses crt,dos;                            (* TURBO CRT AND DOS UNITS *)

type
  drive_ids=record                    (* DISK-SUBSYSTEM INFORMATION *)
      no_drive:integer;                   (* NUMBER OF TOTAL DRIVES *)
      no_logical:integer;               (* NUMBER OF LOGICAL DRIVES *)
      no_floppy:integer;                 (* NUMBER OF FLOPPY DRIVES *)
      no_hard:integer;                     (* NUMBER OF HARD DRIVES *)
      drive_chars:array[1..10] of char;   (* DRIVE CHARACTERS ARRAY *)
      drive_type:array[1..10] of integer;   (* ARRAY OF DRIVE TYPES *)
      drive_numb:array[1..10] of integer; (* DOS DRIVE NUMBER ARRAY *)
      c_drive:integer;                             (* CURRENT DRIVE *)
      c_drive_ch:char; ,                 (* CURRENT DRIVE CHARACTER *)
      last_drive:char;                      (* LAST DRIVE CHARACTER *)
      end;

var
  drives:drive_ids;                     (* GLOBAL INFO ABOUT DRIVES *)
  regs:registers;                                  (* DOS REGISTERS *)

(* MODULE : GET CURRENT DRIVE                                       *)
(* Returns current drive number.   A=1, B=2, C=3, ETC.              *)
(*                                                                  *)
function get_current_drive:integer;
   begin
   regs.ah:=$19;                   (* FUNCTION 19H, GET CURRENT DRV *)
   intr($21,regs);                                       (* INT 21H *)
   get_current_drive:=regs.al;             (* RETURN DRIVE NUMBER *)
   end;

(* MODULE : GET NUMBER LOGICAL DRIVES                               *)
(* Return number of logical drives in system.                       *)
(* NOTE : DOS 3.0 + returns either 5 or the last drive in system.   *)
(*                                                                  *)
function get_number_logical_drives:integer;
   var
      current:integer;
   begin
   current:=get_current_drive;       (* GET CURRENT DRIVE TO SET TO *)
   regs.ah:=$0E;                         (* FUNCTION 0EH, SET DRIVE *)
   regs.dl:=current;                        (* SET TO CURRENT DRIVE *)
   intr($21,regs);                                       (* INT 21H *)
   get_number_logical_drives:=regs.al;   (* RETURN # LOGICAL DRIVES *)
   end;

(* MODULE : GET NUMBER FLOPPY DRIVES                                *)
(* Return the number of floppy drives.                              *)
(*                                                                  *)
function get_number_floppy_drives:integer;
   var
     drives_there,number:integer;
   begin
   number:=-1;                                  (* SET NUMBER TO -1 *)
   intr($11,regs);                        (* INT 11, EQUIPMENT INFO *)
   drives_there:=regs.ax and 1;       (* FLOPPY INSTALLED, BIT0 = 1 *)
   if drives_there=1 then          (* IF FLOPPYS INSTLLD, HOW MANY? *)
      number:=(regs.ax shr 6) and 4;    (* BIT 6-7 HOLD APPROP INFO *)
   get_number_floppy_drives:=number+1;      (* RETURN # OF FLOPPIES *)
   end;

(* MODULE : GET DRIVE DATA                                          *)
(* Get all the drive data ness. Store in global var drives.         *)
(*                                                                  *)
procedure get_drive_data;
   var
      loop,count,h_count,start:integer;
   begin
   count:=0;                            (* SET TOTAL COUNTS TO ZERO *)
   h_count:=0;

   for loop:=1 to drives.no_floppy do  (* ASSIGN FLOPPY INFO IF ANY *)
      begin                          (* SET CORRECT DRIVE CHARACTER *)
      drives.drive_chars[loop]:=chr(loop+64);
```

```
      drives.drive_type[loop]:=1;   (* SET DRIVE TYPE TO FLOPPY *)
      drives.drive_numb[loop]:=loop-1;    (* LOGICAL DRIVE NUMBER *)
      inc(count);                          (* INC TOTAL DRIVE COUNT *)
      end;
                                          (* NO MORE THAN 10 DRIVES *)
   if (drives.no_logical>10) then drives.no_logical:=10;

   start:=drives.no_floppy+1;        (* FIGURE START NUMBER H-DRIVE *)
   if (drives.no_floppy<=1) then inc(start);
   if (drives.no_floppy=0) then inc(start);
   for loop:=start to drives.no_logical do     (* CHECK EACH HDRIVE *)
      begin
      regs.ah:=$1C;                                 (* FUNCTION 1CH *)
      regs.dl:=loop;                    (* SET DRIVE NUMBER TO LOOP *)
      intr($21,regs);                                   (* INT 21H *)
      if (regs.al<>$FF) then        (* IF DRIVE IS VALID, SET APPR. *)
         begin                       (* SET CORRECT DRIVE CHARACTER *)
                                       (* SET DRIVE TYPE TO H-DRIVE *)
                                   (* FIGURE ACTUAL LOGICAL DRIVE # *)
         drives.drive_chars[loop-start+drives.no_floppy+1]:=
                 chr(loop+64);
         drives.drive_type[loop-start+drives.no_floppy+1]:=2;
         drives.drive_numb[loop-start+drives.no_floppy+1]:=loop-1;
         inc(count);                       (* INC TOTAL DRIVE COUNT *)
         inc(h_count);                         (* INC H-DRIVE COUNT *)
         end;
      end;
   drives.no_hard:=h_count;            (* ASSIGN NUMBER HARD DRIVES *)
                                   (* ASSIGN NUMBER OF TOTAL DRIVES *)
   drives.no_drive:=h_count+drives.no_floppy;
   end;

(* MODULE : WRITE DISK CHOICES                                      *)
(* Write all available drives (up to 10), and write the drive type. *)
(*                                                                  *)
procedure write_disk_choices;
   var loop:integer;
   begin
   for loop:=1 to drives.no_drive do     (* LOOP FROM FIRST TO LAST *)
      begin
      write(drives.drive_chars[loop],': ');(* WRITE DRIVE CHARACTER *)
                                                (* WRITE DRIVE TYPE *)
      if (drives.drive_type[loop]=1) then write('[ Floppy  ]')
      else write('[ H-Drive ]');
                                      (* WRITE LOGICAL DRIVE NUMBER *)
      writeln('  ,DOS Logical Disk # ',drives.drive_numb[loop]:2);
      end;
   end;

begin
clrscr;
                                          (* GET ALL THE DRIVE INFO *)

drives.no_floppy:=get_number_floppy_drives;
drives.no_logical:=get_number_logical_drives;
drives.c_drive:=get_current_drive;
get_drive_data;
drives.c_drive_ch:=chr(drives.c_drive+65);
drives.last_drive:=chr(drives.no_logical-drives.no_floppy+65);

                                         (* WRITE ALL THE DISK INFO *)
writeln('SYSTEM DISK INFORMATION');
writeln('————————————');
writeln('Number   Logical Drives = ',drives.no_logical:1);
writeln('Number   Floppy  Drives = ',drives.no_floppy:1);
writeln('Number   Hard    Drives = ',drives.no_hard:1);
writeln('Current Drive           = ',drives.c_drive_ch);
writeln('Last Drive              = ',drives.last_drive);
writeln;
writeln('ACTUAL DRIVES IN SYSTEM');
writeln('————————————');

write_disk_choices;
end.
```

◆ ◆ ◆

R. S. Cunningham III
512-D Georgetown Rd.
Charlottesville, VA 22901

Disk Drive Sensing

Recently I had to write an install program for a new application. My first problem was to find out how many floppy drives, how many hard drives (if any), and their letter designations. (I was working in Turbo Pascal, but any language that supports DOS interrupts should do.)

The solution seemed simple at first: find the number of logical drives in the system, then find the number of floppy drives. The difference will be the number of hard drives.

However, DOS 3.0 (and higher) does not accurately report the number of logical drives. It reports either the number of drives, or the number in config.sys's LASTDRIVE= statement, or it returns a 5 if there are no drives.

The routines listed in Figure 1 start out by getting the number of floppy drives. I use DOS interrupt 11H, checking the value of bits 6-7.

Then I find the number of logical drives. I do this by calling DOS interrupt 21H, function 19H to get the current drive number. Then I call DOS interrupt 21H, function 0EH which selects a drive and returns the number of logical drives. Finally I figure out the number of the first hard drive.

Once I know the designation(s) of the hard drive(s), I check its status. I do this by calling DOS interrupt 21H, function 1CH—get drive data. If the function returns FFH, the drive is not valid.

If there's no FFH, I add the number to the valid drive list. Since I know the drives I'm checking are hard drives, I don't have to bother about checking the drive type.

The routines store all the information in the global variable "drives." I've included lots of remarks so you shouldn't have much trouble reading the code.

The routines could be improved by thoroughly examining each drive (e.g., for color, noise level...). I'll let you figure this out. As the code stands, only RAM disks aren't correctly identified.

Jeffrey Donovan
1515 Santa Barbara St. #A
Santa Barbara, CA 93101

◆ ◆ ◆

Is There A Gap In Your Info?

Fill in your Back Issues of Micro C today!

ISSUE #1 (8/81)
Power Supply
1/2 PFM.PRN
16 pages

ISSUE #2 (10/81)
Parallel Print Driver
Drive Motor Control
16 pages

ISSUE #3 (12/81)
4 MHz Mods
Configuring Modem 7
Reverse Video Cursor
FORTHwords Begins
16 pages

ISSUE #4 (2/82)
Keyboard Translation
More 4 MHz Mods
Modems, Lync, and S10s
Undoing CP/M ERASE
20 pages

ISSUE #5 (4/82)
Two Text Editors
Double Density Review
20 pages

ISSUE #6 (6/82)
BBI EPROM Programmer
Customize Your Chars
Double Density Update
24 pages

ISSUE #7 (8/82)
6 Reviews Of C
Adding 6K Of RAM
On Your Own Begins
24 pages

ISSUE #8 (10/82)
SOLD OUT

ISSUE #9 (12/82)
BBII EPROM Program
Relocating Your CP/M
Serial Print Driver
Big Board I Fixes
32 pages

ISSUE #10 (2/83)
ISSUE #11 (4/83)
SOLD OUT

ISSUE #12 (6/83)
Bringing Up BBII
Double Sided Drives for BBI
Packet Radio
5 MHz for Kaypro
40 pages

ISSUE #13 (8/83)
CP/M Disk Directory
More 256K for BBI
Mini Front Panel
Cheap Fast Modem
BBI Printer Interface
Kaypro Reverse Video Mod
44 pages

ISSUE #14 (10/83)
BBII Installation
The Perfect Terminal
BBI Video Size
Video Jitter Fix
Kaypro Color Graphics Review
48 pages

ISSUE #15 (12/83)
Screen Dump Listing
Fixing Serial Ports
Playing Adventure
Upgrading Kaypro II To 4
Upgrading Kaypro 4 To 8
48 pages

ISSUE #16 (2/84)
Xerox 820 Column Restarts
BBI Double Density
BBII 5"/8" Interface Fix
Kaypro ZCPR Patch
Adding Joystick To Color Graphics
Recovering Text From Memory
52 pages

ISSUE #17 (4/84)
Voice Synthesizer
Kaypro Morse Code Interface
68000-Based System Review
Inside CP/M 86
56 pages

ISSUE #18 (6/84)
Kaypro EPROM Programmer
I/O Byte: A Primer
Kaypro Joystick
Serial To Parallel Interface
Business COBOL
60 pages

ISSUE #19 (8/84)
Adding Winchester To BBII
6 MHz On The BBI
Bulletin Boards
Track Buffering On Slicer
4 MHz For The 820-I
64 pages

ISSUE #20 (10/84)
HSC 68000 Co-Processor
DynaDisk For The BBII
Serial Printer On BBI Sans S10
Cheap & Dirty Talker For Kaypro
Extended 8" Single Density
72 pages

ISSUE #21 (12/84)
Analog To Digital Interface
Installing Turbo Pascal
Low Intensity BBI Video
Turbo Pascal, The Early Days
80 pages

ISSUE #22 (2/85)
Xerox 820-II To A Kaypro-8
Sound Generator For the STD Bus
Reviews Of 256K RAM Expansion
88 pages

ISSUE #23 (4/85)
Automatic Disk Relogging
Interrupt Drive Serial Printer
Low Cost EPROM Eraser
Smart Video Controller
Review: MicroSphere RAM Disk
86 pages

ISSUE #24 (6/85)
C'ing Into Turbo Pascal
8" Drives On The Kaypro
68000 Versus 80x86
Soldering: The First Steps
88 pages

ISSUE #25 (8/85)
Why I Wrote A Debugger
The 32-Bit Super Chips
Programming The 32032
Modula II
RS-232C: The Interface
104 pages

ISSUE #26 (10/85)
Inside ZCPR3
Two Megabytes On DSI-32
SOG IV
The Future Of Computing
Graphics In Turbo Pascal
104 pages

ISSUE #27 (12/85)
SOLD OUT

ISSUE #28 (2/86)
Rescuing Lost Text From Memory
Introduction To Modula-2
Inside The PC
104 pages

ISSUE #29 (4/86)
Speeding Up Your XT
Prototyping In C
C Interpreters Reviewed
Benchmarking The PCs
104 pages

ISSUE #30 (6/86)
PROLOG On The PC
Expert Systems
Logic Programming
Building Your Own Logic Analyzer
256K RAM For Your 83 Kaypro
PC-DOS For Non-Clones
104 pages

ISSUE #31 (8/86)
RAM Resident PC Speedup
Practical Programming In Modula-2
Unblinking The PC's Blinkin' Cursor
Game Theory In PROLOG and C
104 pages

ISSUE #32 (10/86)
Public Domain 32000: Hardware And Software
Writing A Printer Driver for MS-DOS
Recover A Directory By Reading & Writing Disk Sectors
96 pages

ISSUE #33 (12/86)
ISSUE #34 (2/87)
ISSUE #35 (4/87)
ISSUE #36 (6/87)
ISSUE #37 (9/87)
SOLD OUT

ISSUE #38 (11/87)
Parallel Processing
Laser Printers, Typesetters And Page Definition Languages
Build A Graphics Scanner For $6, Part 2
Writing A Resident Program Extractor In C
96 pages

ISSUE #39 (1/88)
PC Graphics
Drawing The Mandelbrot And Julia Sets
Desktop Graphics
Designing A PC Workstation Board
Around the TMS-34010
96 pages

ISSUE #40 (3/88)
The Great C Issue
11 C Compilers
Writing A Simple Parser In C
C++, An Object Oriented C
Source Level Debugger For Turbo C
96 pages

ISSUE #41 (5/88)
Artificial Intelligence
3-D Graphics
Neural Networks
Logic Of Programming Languages
Applying Information Theory
96 pages

ISSUE # 42 (7/88)
Maintaining PCs
Keeping Your Hard Drives Running
Troubleshooting PCs
XT Theory of Operation
Simulating A Bus
Ray Tracing
96 pages

ISSUE #43 (9/88)
Building Databases
Build a C Database
Selecting a dBase III Compatible Compiler
Working with Paradox
Designing Custom PC Cards
Accessing dBase III Plus Records from Turbo Pascal
96 pages

ISSUE#44 (11/88)
Object-Oriented Programming
A Taste of Smalltalk
Actor
Thinking Objectively
Building MicroCad
Peripheral Technology-PT68K-2
Hercules Graphics Printer Dump
96 pages

ISSUE #45 (1/89)
Computer Aided Design
CAD In A Consulting Business
Choosing PCB Layout Systems
Building Circuits With Your Computer
Secrets of Optimization
Finding Bargains in the Surplus Market
MASM 5.1
96 pages

ISSUE #46 (3/89)
Software Tools
The Art of Disassembly
Handling Interrupts With Any C
Hacking Sprint: Creating Display Drivers
Greatest C Compilers
Turning A PC into An Embedded Control System
Practical Fractals
96 pages

ISSUE #47(5/89)
Robotics
The LIMBO Project
Starting A Robotics Company
How To Write and Use A SystemProfiler
Problem Solving and Creativity
Turn Your XT Into A Controller
Writing Code For Two Operating Systems
96 pages

ISSUE #48(7/89)
Tools For The Physically Impaired
The Adventure Begins
Selecting A Talking Computer For A Blind Friend
Writing Software For The Blind
File Transfer Via The Parallel Port
The LIMBO Project—Part Two
PCX Compatibility
A 68000-Based Multitasking Kernel
The Very Early Days of Computing

◆◆◆

To Order:

Phone: 1-800-888-8087
Mail: PO Box 223
Bend, Oregon 97709

United States,
Issues #1-34 $3.00 each ppd.
Issues #35-current $3.95 each ppd.

Canada & Mexico
All issues $5.00 each ppd.

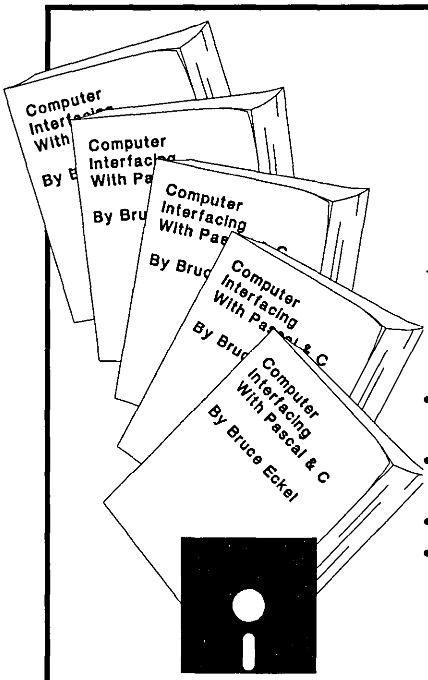
Foreign (air mail)
All Issues $7.00 each ppd.

ADVERTISERS INDEX

Issue 49

** Contact Advertiser Directly.

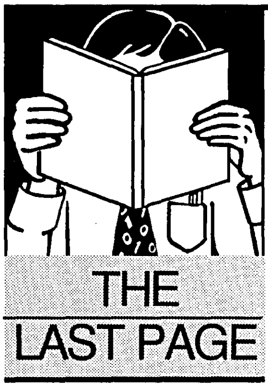
When you write for information, please tell these folks you read about their products in **Micro Cornucopia.**



Issue #50

I/O, Yet Again

- **Capturing Video Images**
- **Voice Capture and Analysis**
- **The Further Exploits of LIMBO**
- **Bits From Our Past**
- **3-D Graphics Programming**
- **PostScript Programming, Part 2**



By Gary Entsminger
1912 Haussler Dr.
Davis, CA 95616

Stochastic Fiction—Fiction From Fractals

This introduction should not be words, it should be a fractal something. I think.

I type this at a desk far from the Davis heat; I'm sitting in the Elk mountains of Colorado and the face of this mountainside varies daily. (In fact, it varies much more frequently, but my perception of the variation is limited by a matrix of factors.)

This morning, the ghostly purple larkspur (*Delphinium nelsonii*, named both for its dolphin-like shape and a Wyoming botanist) has suddenly bloomed. It's surrounded by sky blue lupines and various green bushes yet to bloom. And it seems they're taller by several inches since yesterday.

As you read this, the mountainside has varied many times, until the only mountainside that exists is the one you imagine. Perhaps now (in my future) it's in late summer seed.

Discussions of time (like this one?) are strange and always eventually inconclusive. Yet time continues to fascinate poor and great imaginations alike. Thinking about time is fun; and reading someone's well-constructed thoughts is better yet.

Great Work Of Time

Recently, I read a skillfully-constructed story about time by John Crowley, a World-fantasy Award-winning author. The story, "Great Work Of Time," extends that well-worn subclass of science fiction, "time travel," by incorporating chaos theory into the argument.

In "GWOT," a strange genius, named Caspar Last, discovers (or creates) a process which can be used for time travel. The key tool in leading him to discover the process is a computer, which he uses to plot the shadows of imaginary numbers.

Last showed me [the narrator of the story] after our bargain was struck and he was turning over his data and plans to me. I told him I would not probably grasp the theoretical basis of the process, however well I had or would come to manage the practical processes of it, but he liked to show me.

He first summoned up x-y coordinates, quite ordinary, and began by showing me how some surprising results were obtained by plotting on such coordinates an imaginary number, specifically the square root of minus one. The only way to describe what happens, he said, is that the plotted figure, one unit high, one unit wide, generates a shadow square of the same measurements "behind" itself, in space undefined by the coordinates.

It was with such tricks that he had begun; the orthogons he obtained had first started him thinking about the generation of inhabitable—if also somehow imaginary—pasts.

Fogg fans will no doubt recognize these tricks. They're fractals, which Last uses to travel into the past and then back to the future.

Sometime later (back in the future), Last discovers an inescapable paradox—that the future he's returned to isn't the future he left. Not the future of his former past, but the future of the new past he created by time-traveling.

Although Last had calculated to the penny the necessity of disturbing nothing of importance during his journey, he hadn't (and couldn't) succeed. All futures (in "GWOT") are determined by minute variations and choices. Paths diverting, particles close together, *pasts altered slightly* vary little by little but wind up very far apart.

Each choice, each detail that made up Last's past is a bifurcation (or branching) from what might have been, and leads to a *future* slightly removed (at least) from that of any other *past*.

So Last's one and only journey into the past opens up a kettle of futures and pasts and futures ad infinitum. The stories within the story, "Great Work Of Time" are intriguing and subtle. Not the least is a moralist's viewpoint that our choices lie at the crux of all that's important.

The difference you make, makes all the difference.

Last must have known the instant he arrived on the morning of 1856, 127 years earlier, that something was up—

Out of the unimaginable chaos of its interminable stochastic fiction, Time thrust only one unforeseen oddity on Caspar Last as he, or something like him, appeared beneath a plaintain tree in 1856: he had grown a beard almost down to his waist. It was abominably hot.

Funny Lane

I expect, as literary folk learn more and more about the implications of scientific theories, we'll see more and more thoughtful excursions like this one into *stochastic fiction*.

For now, if you want to enjoy a good thought, check out, *Novelty: Four Stories* by John Crowley, in paperback ($6.95) from Doubleday.

◆ ◆ ◆

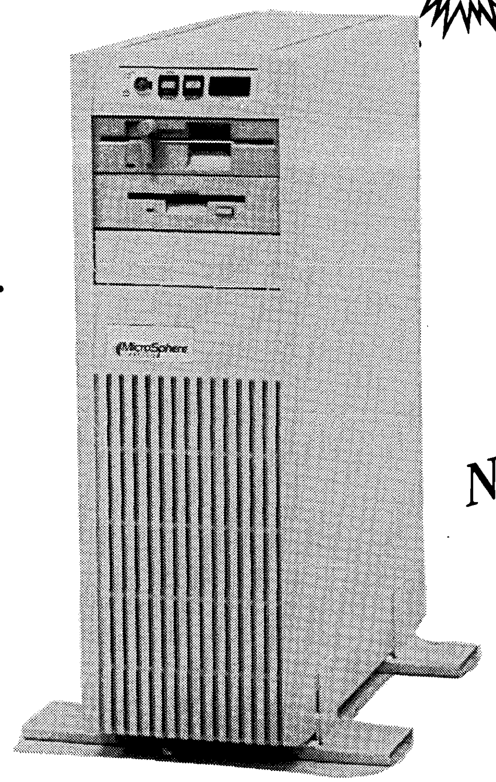
Quality & Price You Can't Pass Up!

New Features

Display Color Graphics on a monochrome monitor!

All Microsphere XT, AT & 386 systems NOW include the GRAPHICS COMBO multimode video card. NEW!

The Graphics Combo combines the video functions and software compatibility of the IBM Monochrome Display Adapter (MGA), the Hercules Graphics Card, and the IBM Color/Graphics Adapter (CGA) all on a single card. In addition the CGA graphics are automatically converted to display on a standard TTL monochrome monitor in 16 shades of gray.



New

COMPLETE 80386-25SDX SYSTEM

with Rotary Voice Coil Hard Drive

This machine features an 80386 CPU running at 25 Mhz on a full size motherboard. For extra quality and reliability we've included a 45 Mb Miniscribe 3053 Hard Drive with a 25 ms access time, 1.2Mb & 1.44Mb Toshiba or TEAC floppy drives. 2 8Mb 32 bit memory slots, 1Mb of fully optimized 80ns RAM (Runs an incredible Norton SI test of 24!), 2 serial ports, 1 parallel port, 101 key keyboard, **Graphics Combo** video card with amber or green monochrome monitor, Bios, socket for an 80387-20 math coprocessor, 200 watt power supply, clock/calendar, and housed in a sophisticated tower case. Full 1 year warranty. Ask for FREE assembly and testing..

Now! $2495! *(Tower Case)*

$2395 *(Std. Case)*

20MHz	$200 Less
8 MB RAM Crd (32 BIT, 0K)	99

SPECIAL OFFER!
Order a Complete MicroSphere Computer System and receive 7 FREE disks of our best public domain games.

XT SYSTEM

Includes: 640K RAM, serial/parallel/game ports, clock/calendar, 101 key keyboard, turbo switchable, slide cabinet, power supply, **Graphics Combo** video card with amber or green monitor. Full 1 year warranty. Ask for FREE assembly and testing.

4.77/10 Mhz with 2 360K floppies	**725**
1 FD and 1 Miniscribe HD:	
4.77/10 Mhz with 20 Mb HD	**929**
4.77/10 Mhz with 30 Mb HD	**955**

AT SYSTEM

Includes: 640K RAM, 1.2 Mb FD, 1.44Mb FD, 40 Mb Miniscribe 3650 HD, serial/parallel/game ports, clock/calendar, 101 key keyboard, turbo switchable, slide cabinet, power supply, **Graphics Combo** video card with amber or green monitor. Full 1 year warranty. Ask for FREE assembly and testing.

6/10 Mhz	**1249**
6/12 Mhz	**1295**

Color options for any kit (includes video card and monitor)

CGA Color	175
CGA/EGA Color	380
VGA (analog) with Mitsubishi monitor	650
CGA/EGA/EGA 480 (Multisync)	450

PC XT & AT

Clock	19
Game	14
Parallel (LPT 1, 2 or 3)	18
Serial Port Card - 1 installed Switchable Com 1, 2, 3 or 4	18
Kit for 2nd SerialPort	18
Multi I/O	
Serial/Par/Game	32
2nd Serial Kit	20
Multi Drive Controller	39
Supports 1.44, 720K, 1.2, 360K drives	

PC/XT

Floppy Controller	19
Multi-function-1 ser/par/clk/game/2 floppy	47
640K RAM (0K)	25
150 Watt Power Supply	50
Slide case lock, LED	38
2 MB EMS Memory Board 0K Installed	49

AT

200 Watt Power Supply	75
AT/386 Case, Lock, LED	72
Tower AT/386 Case, Lock, LED & 200 Watt ps	239
2 MB EMS Memory Board 0K Installed	99

MOTHERBOARDS

XT/Turbo 4.77/10	75
AT 6/10 Award/Phoenix/DTK Bios	229
AT 6/12 Award/Phoenix/DTK Bios	279
Baby AT 6/12 AMI/DTK	249
AT 8/16 DTK Bios	395
80386 8/20 DTK Bios	799
XT/AT Memory	$CALL

SOFTWARE

MS DOS 3.21 w/GW Basic	49
DR DOS 3.3 w/GEM	49
MS DOS 3.3 w/GW Basic	95

DISK DRIVES

Teac/Toshiba 360K	69
Teac/Toshiba 1.2 MB	85
Teac/Toshiba 3½" 720K	79
Teac/Toshiba 3½" 1.44 MB kit	90
XT 20 MB Miniscribe	
8425 (65ms)	279
8425 w/controller	319
XT 30 MB Miniscribe	
8438 (65ms)	299
8438 w/controller	349

DISK DRIVES (Continued)

AT 40 MB MiniScribe 3650 (61ms)	339
AT 40 MB MiniScribe 3053 (25ms)	489
AT 71MB MiniScribe 6085 (28ms)	631
AT (MFM) HD & FD Controller card DTK	110
WD	127
AT RLL HD & FD Controller	189

MONITORS/CARDS

EGA/CGA (Autoswitch .31 dot)	385
CGA/EGA/VGA MultiSync (.31 dot)	495
CGA Color	249
Amber/Green 12" TTL	89
Graphics Combo	79
VGA Analog (Mitsubishi .28 dot)	549
Color/Graphics/Par Card	49
Mono/Graphics/Par Card	49
CGA/EGA Card	175
VGA Analog/Digital Card	249

KEYBOARDS

Casper Enhanced 101	54
Keytronic KB101	67
Focus 101 Tactile, Switchable, Control Caps Lock, Dust Cover (#1 find by MicroC Staff)	89

★ All keyboards, XT/AT switchable ★

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

BUILDING YOUR OWN CLONE V2.1
******FREE BOOKLET******

*90-day warranty/30-day money back (subject to restrictions)

Tech Calls: (503) 388-1194

Hours: Monday-Friday 9:00-5:30

MicroSphere INC.
COMPUTERS
"HARDWARE MANUFACTURER SINCE 1983"

Orders Only: 1-800-234-8086
855 N.W. WALL • BEND, OREGON 97701

Reader Service Number 2

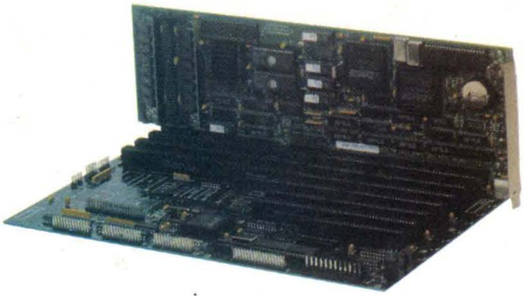
VERY HIGH PERFORMANCE

Processors, Memory, and Display Adapters

The X24 High performance processor

- 12 or 16 MHz 80286 with NO WAIT STATES!
- Small size ("XT" height and length) passive bus design
- 1 to 4 Mbyte 0 wait state dynamic memory
- Fully "AT" compatible Award BIOS
- Runs DOS versions 2.2 and later, Xenix and OS/2

The X24 combines the best of motherboard and backplane designs in a 100% AT compatible system. Incorporating a 16 MHz 80286, the X24 processor is designed to operate with the PC Tech Advanced System Motherboard, which contains the peripheral interfaces (hard disk, floppy disk, two serial ports and a parallel port). The X24 processor can also be used with other totally passive bus backplanes. Most critical components including the microprocessor and up to 4 megabytes of fast memory are contained on a single PC size plug-in card. This allows the processor and main system memory to be serviced or upgraded without disturbing other peripherals such as serial ports and disk drives.



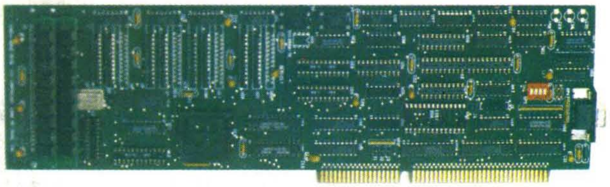
PC Tech X24 and ASMB

The PC Tech Advanced System Motherboard

- Built in "IDE" interface for AT interface type hard drives
- Fully AT compatible floppy disk support for 3.5", 5.25" drives, capacities of 360k, 1.2m and 1.44m
- Two serial ports and one parallel port
- 8 total expansion slots PC/XT/AT compatible (4 slots have 32 bit bus)

The PC Tech Advanced System Motherboard is designed to complement PC Tech's X24 and X32 high performance processor cards. It contains the mass storage interfaces necessary for a complete system, plus the basic I/O required in most systems. Extra care has been given to FCC compliance by design.

34010 Monochrome Graphics Adapter II



PC Tech Mono-II

- Up to 384k bytes display memory
- Up to 2 Megabytes program memory
- Software is RAM based, allowing complete operating software replacement and timing re-programming from the host bus
- 34010 program loader included. Assembler, debugger, and C compiler available.
- Full hardware and software CGA, MDA and Hercules emulation
- Single bit shared memory bit-map with optional resolution up to 2048 x 1536 (736 x 1008 standard)
- Very high resolution COLOR version available
- Custom 34010 software development available

The TMS34010 is a true general purpose graphics processor. PC Tech makes the total processing power of the 34010 available to both programmers and end users. Our 34010 Monochrome Graphics Adapter is designed to allow programming from the PC/XT/AT host bus. You can completely replace our 34010 software with yours to directly harness the incredible image processing power of the TMS 34010 for your application. We make a complete set of development tools available, including an assembler, C compiler, program loader, 34010 debugger, and PC interface tracer/debugger. Our standard product includes support for extended CGA, MDA and Hercules emulation as well as a host addressable graphics bit- map. We also support and recommend the DGIS graphics interface standard (from Graphic Software Systems) for applications development as an alternative to native 34010 software development. Ready to run drivers are available for most major applications software packages as well.

Custom Designs Available

PC Tech will license most products for non-exclusive manufacture. We will also customize any of our designs to better meet your needs on our in-house CAD systems. All of our standard products are available in private label versions.

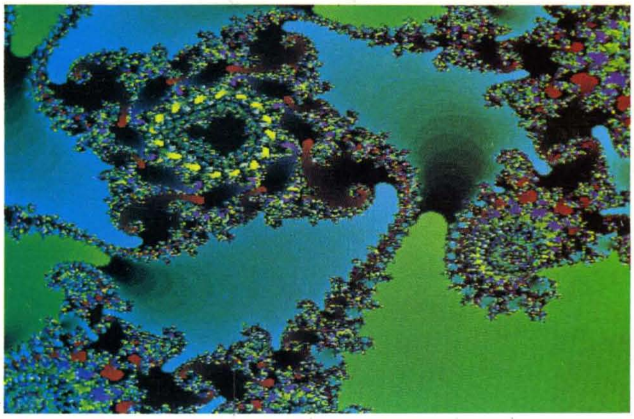
About PC Tech

PC Tech has been designing, manufacturing and marketing high performance PC related products for over three years. Our standard product line includes processor, memory, and video products. All products are designed, manufactured and supported in our Lake City, Minnesota facilities.

Designed, Sold and Serviced By:

PC Tech

907 N. 6th St., Lake City, MN 55041
(612) 345-4555 • (612) 345-5514 (FAX)



High resolution fractal produced on the PC Tech COLOR 34010

Reader Service Number 3

PC, XT, AT, DGIS, Hercules, and Award BIOS are trademarks or registered trademarks of their respective companies.