No. 46 March-April 1989 $3.95

THE MICRO TECHNICAL JOURNAL

MICRO CORNUCOPIA

Software Tools

OK, you asked for it, an issue dedicated (mostly) to software.

The Art of Disassembly page 8

Using Sourcer to create commented, assembly source from object files.

Handling Interrupts With Any C page 16

Hacking Sprint: Creating Display Drivers page 36

Annual C Reviews

Comparing the latest, greatest C compilers. page 40

And More . . .

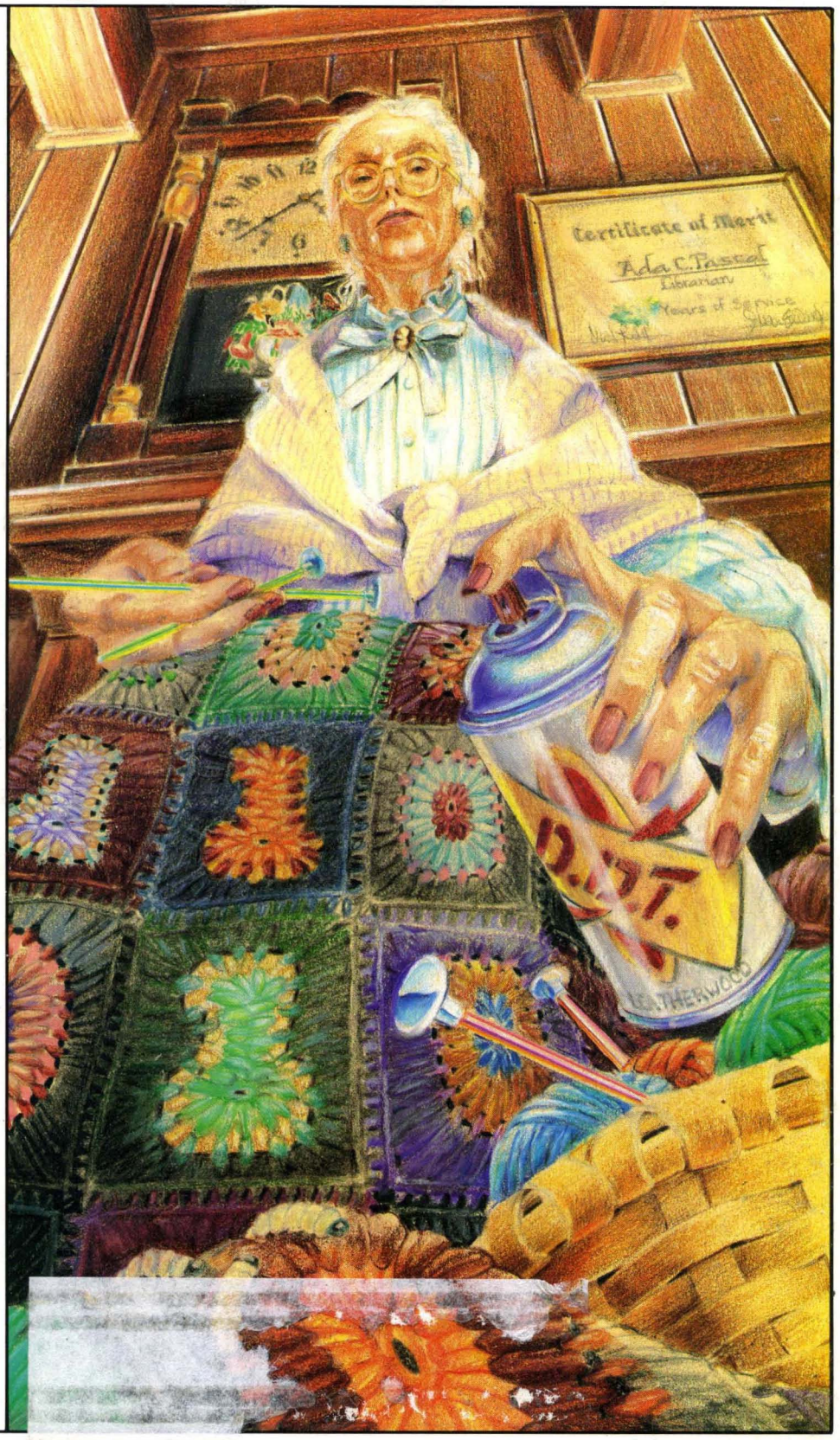
Turning A PC Into An Embedded Control System page 24

Bringing Up A Surplus 68000 Board page 28

Part 2 of Karl Lunt's cheap 68000 project.

Plus:

Practical Fractals 32
Shareware you must register 62
And Much, Much, More



04
0 74470 19388 3

Aztec C ROM Cross Development Systems Produce Fast, Tight C Code with Less Effort

Aztec C ROM Cross Development Systems give you the best results — clean, tight and fast running code. Aztec C systems are available for a variety of targets and for both MS-DOS or Apple Macintosh hosts! And, Aztec C systems come complete with all the tools to edit, compile, assemble, optimize and, now, *source debug* your C code in less time and with less effort.

Quality, tight code that's fast and efficient. An abundance of tools to produce better results in less time. That's why Aztec C systems are the choice of more professional ROM developers.

So when you're looking for the best results, insist on Aztec C ROM Cross Development Systems. Call today and find out more about our complete line of Cross Development Systems.

Supported targets include: the 68xxx family, the full 8086 family, the 8080/Z80 family and the 6502 family of microprocessors.

1-800-221-0440 (outside NJ)

(NJ and Outside U.S.) **1-201-542-2121** Telex: 4995812MANX Fax: 201/542-8386

MS-DOS is a registered trademark of Microsoft Corporation
Apple and Macintosh are registered trademarks of Apple Computer Corporation.

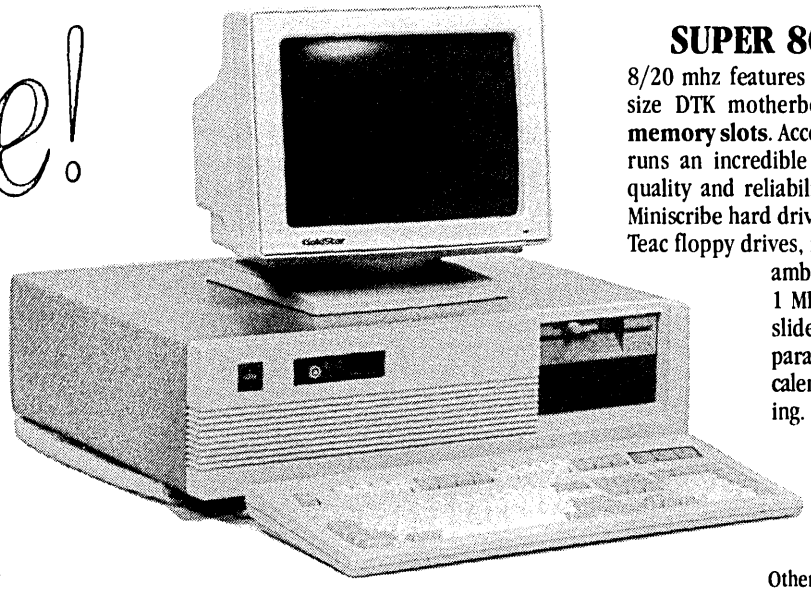
Aztec by MANX
SOFTWARE SYSTEMS

One Industrial Way Eatontown, New Jersey 07724

Reader Service Number 17

Quality & Price You Can't Pass Up!

Sale!



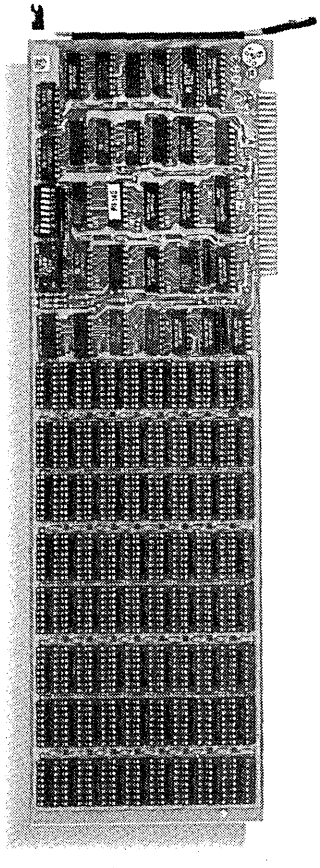
SUPER 80386 SYSTEM

8/20 mhz features an 80386-20 CPU on a full size DTK motherboard with **2 8MB 32 bit memory slots**. Accepts 80387-20. This machine runs an incredible Norton SI test of **24**! For quality and reliability we've included a 40 MB Miniscribe hard drive, 1.2MB & 360K Toshiba or Teac floppy drives, monographics with green or amber monitor, 101 keyboard, 1 MB of 80 ns DRAM, DTK Bios, slide case, 2 serial ports, 1 parallel port, power supply, clock, calendar. FREE assembly and testing. One year warranty.

Now!
$2395!

Other configurations:
tower case, color, etc. .....$Call
DTK 8 MB RAM Card 99
(32 BIT, 0K)

2 MB EMS MEMORY BOARDS



XT SYSTEMS

Include: 640K RAM, serial/parallel/ game ports, clock/calendar, 101 key keyboard, turbo switchable, slide cabinet, power supply, mono graphics with amber/green monitor. 1 year warranty. FREE assembly and testing.

4.77/10 with 2 Floppy 790
IFD and 1 Miniscribe HD:
4.77/10 with 20 MB HD 995
4.77/10 with 30 MB HD 1010

AT SYSTEMS

Includes: 640K RAM, 1.2 MBFD, 1 360K FD, 40 MB Miniscribe 3650HD serial/parallel/game ports, clock/calendar, 101 key turbo switchable keyboard, slide cabinet, power supply, monographics with amber or green monitor. Full one year warranty. **FREE** assembly and testing.

6/10 mhz 1450
12 mhz 1495
Color options for any kit (includes video card and monitor)
CGA Color 175
CGA/EGA Color 380
VGA (analog) 650
CGA/EGA/VGA 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 30
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

AT

200 Watt Power Supply 75
AT/386, Lock, LED 65
Tower AT/386, Lock,
LED & 200 Watt ps 239

MOTHERBOARDS

XT/Turbo 4.77/10 79
AT 6/10 Award/Phoenix/
DTK Bios 249
AT 6/12 Award/Phoenix/
DTK Bios 299
Baby AT 6/12 AMI/DTK 279
80386 8/20 DTK Bios 937
XT/AT Memory .....$CALL

SOFTWARE

MS DOS 3.21 w/GW Basic 49
DR DOS 3.3 w/GEM 49

DISK DRIVES

Teac/Toshiba 360K 80
Teac/Toshiba 1.2 MB 105
Teac/Toshiba 3½" 720K 99
Teac/Toshiba 3½" 1.44 MB kit ...139
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) 649
AT (MFM) HD & FD
Controller card DTK 110
WD 127
AT RLL HD & FD
Controller 189

MONITORS

EGA/CGA
(Autoswitch .31 dot) 385
CGA/EGA/VGA
MultiSync (.31 dot) 495
CGA Color 249
Amber 12" TTL 89
Green 12" TTL 89
VGA Analog
(Mitsubishi .28 dot) 549
Color/Graphics/Par 49
Mono/Graphics/Par 49
CGA/EGA/VGA (640x480) 169
VGA Analog, STB Extra 235

KEYBOARDS

Chicony Click 101 49
Keytronic KB101 67
Focus 101 Tactile,
Switchable, Control Caps Lock,
Dust Cover 89
(#1 find by MicroC Staff)
★ All keyboards, XT/AT switchable) ★

XT SYSTEM .. $49
AT SYSTEM ... $99

MicroSphere INC.
COMPUTERS "HARDWARE MANUFACTURER SINCE 1983"

Orders Only: **1-800-234-8086**
855 N.W. WALL • BEND, OREGON 97701

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

Reader Service Number 2

SERIOUS DEBUGGING *at a* REASONABLE PRICE

All the speed and power of a hardware-assisted debugger at a software price

Hardware-level break points

REAL-TIME break points on memory locations, memory ranges, execution, I/O ports, hardware and software interrupts. More powerful break points than ANY software-only debugger on the market. Soft-ICE gives you the power of an in-circuit emulator on your desk.

Break out of hung programs

With a keystroke - no external switch necessary. Even with interrupts disabled.

Breaks the 640K barrier

Soft-ICE uses ZERO bytes of memory in the first 1MB of address space. This is especially useful for those subtle bugs that change when the starting address of your code changes. With Soft-ICE your code executes at the same address whether the debugger is loaded or not.

Works with your favorite debugger

Soft-ICE can be used as a stand-alone debugger or it can add its powerful break points to the software debugger you already use. You can continue to use your favorite debugger until you require Soft-ICE. Simply pop up the Soft-ICE window to set powerful real-time break points. When a break point is reached, your debugger will be activated.

Solve tough systems problems too

Soft-ICE is ideal for debugging TSRs, interrupt handlers, self booting programs, DOS loadable device drivers, non-DOS operating systems, and debugging within DOS & BIOS. Soft-ICE is also great for firmware development because Soft-ICE's break points work in ROM.

How Soft-ICE Works

Soft-ICE uses the power of the 80386 to surround your program in a virtual machine. This gives you complete control of the DOS environment, while Soft-ICE runs safely in protected mode. Soft-ICE uses 80386 protected mode features, such as paging, I/O privilege level, and break point registers, to provide real-time hardware-level break points.

"Soft-ICE is a product any MS-DOS developer serious enough to own a 386 machine should have."
Dr. Dobb's Journal — May 1988

Both require 80386 AT compatible or IBM PS/2 Model 80. MagicCV requires at least 384K of extended memory. CodeView is a trademark of Microsoft Corporation.

RUN CODEVIEW IN ONLY 8K!

CodeView is a great integrated debugger, but it uses over 200K of conventional memory. MagicCV uses advanced features of the 80386 microprocessor to load CodeView and symbols in extended memory. This allows MagicCV to run CodeView using less than 8K of conventional memory on your 80386 PC.

Don't let 640K be your limit!

If you are closing in on the 640K limit and would like the power of CodeView, MagicCV is for you.

Don't let the debugger hide the bug!

Even if you're not closing in on the 640K limit, running CodeView with MagicCV makes your debugging environment much closer to the end user's program environment. You can use CodeView to locate subtle bugs that only occur when there is plenty of free memory, or those difficult bugs that only occur when your program is running with a couple of TSRs loaded.

How MagicCV works

MagicCV uses the 80386 to create a separate virtual machine for CodeView. MagicCV uses between 4K & 8K of conventional memory as a bridge between the DOS environment and CodeView.

MagicCV is easy to use

If you are a CodeView user, you already know how to use MagicCV too. Just type MCV instead of CV; everything else is automatic.

Save $86

MagicCV	$199
Soft-ICE	$386

Buy Both and Save $86!

CALL TODAY
(603) 888 - 2386
***or FAX* (603) 888 - 2465**

30 day money-back guarantee
Visa, Master Card and AmEx accepted

NU-MEGA TECHNOLOGIES

P.O. BOX 7607 • NASHUA, NH 03060-7607

Reader Service Number 110

MagicCV with Soft-ICE

Using Soft-ICE with CodeView gives you the features necessary for professional level systems debugging. MagicCV and Soft-ICE can work in concert with CodeView to provide the most powerful debugging platform you will find anywhere.

THE MICRO TECHNICAL JOURNAL

MICRO CORNUCOPIA

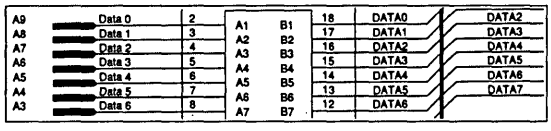
MARCH/APRIL 1989 - ISSUE NO. 46

FEATURES

8 B. H. Flusche, Jr.
The Art of Disassembly
Disassembly has long remained one of the black arts: black because few do it well, black because of its hacker reputation. However, it's a skill that will serve you well. Very, very well.

16 Sam Azer
Handling Interrupts With Any C
Some Cs support interrupts elegantly, some support them barely. Here's how to interrupt with any C in style.

24 Bruce Eckel
Turning A PC Into An Embedded Control System
Bruce goes back to hardware and produces an EEPROM board. This is the first of a series on embedded controllers.



28 Karl Lunt
Bringing Up A Surplus 68000 Board
What would you do with a fancy, but undocumented, 68000-based processor board? Figure out how to make it run, of course.

32 Larry Fogg
Practical Fractals
What does a fractal have that's practical? What does a fractal have in common with the Deschutes River? Larry brings us the answers to these burning questions.

36 Brett Glass
Hacking Sprint: Creating Display Drivers
Now that you've hacked everything else you might as well write a display driver for a text editor. (You'll, of course, use the editor's interface language. Right? You knew about that, I know you did.)

COLUMNS

40 **C'ing Clearly**
The great C comparison for 1989.

52 **86 World**
Laine creates packed and unpacked screen fonts.

60 **ShareWare**
Tony reviews PC-File:dB and PC-Write 3.0.

62 **The Culture Corner**
This column is for adults only.

64 **On Your Own**
Kent Peterson starts a computer newsletter.

68 **Units And Modules**
Absolute precision using rational numbers (via irrational means).

82 **CP/M Notes**
Sources of CP/M products.

90 **Tech Tips**

FUTURE TENSE

86 **Tidbits**
Graphing your data with SlideWrite.

96 **Last Page**
Go.

Cover illustration by Paul Leatherwood

THE MICRO TECHNICAL JOURNAL
MICRO CORNUCOPIA

Editor and Publisher
David J. Thompson

Associate Editors
Gary Entsminger
Cary Gatton

Technical Department
Larry Fogg

Director of Advertising & Distribution
Jackie Ringsage

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 (surfaca)	$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.

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.

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

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

ISSN 0747-587X

By David J. Thompson

Returning SOG To Its Roots

This has probably been the most difficult editorial I've had to write. It's been difficult because I've had to make some hard choices, and it's been difficult because I can't explain everything that went into those choices.

Over the past couple of SOGs (Semi-Official Get-togethers), a number of you have approached me about future SOGs. Would there be future SOGs? Was I as exhausted as I appeared?

"Of course," I said, "there will be another SOG."

"Yes, I'm exhausted," I added, "but this is a wonderful time, the highlight of my year. Sure it's fun to write for 30,000, but it's ten times more exciting to speak with 400."

However, I'm not sure I'd survive another SOG. The Monday after SOG VII, Laura Logan left Micro C. It wasn't a bad leaving, she'd always wanted to live on the Oregon Coast. However, when she left, we lost our SOG organizer. She planned to continue the project, remotely, so we announced SOG VIII. But finally, as things became busy for her there (making waves, I presume), she bowed out.

For SOGs I and II, loss of a key person wouldn't have been a problem. One hundred people showed up at our house for a day of potluck food and potluck discussions. Sure, it was a long day — two days, really, but it was wonderful.

However, SOG has grown to 400 people, too many for any facility short of the local college. Laura made sure there were rest rooms, dorm rooms, motel rooms, and classrooms. She also set up rafting, volleyball nets, food services, apple barrels, and that myriad of wonderful, soggy things that had to be done.

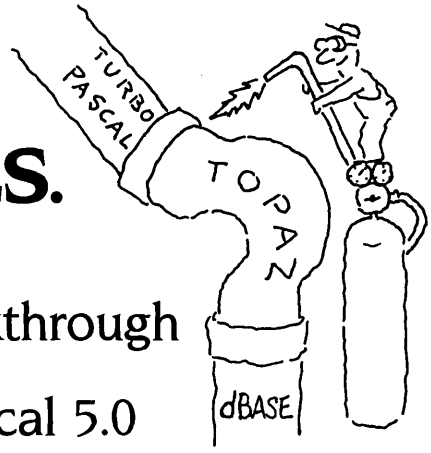
We no longer have someone who has either the time or the experience to effectively handle another SOG, so I've been mulling over some options.

Charge for SOG. That way we could afford to hire a convention manager or service. But who pays? It's a family event with activities for spouses and kids and babysitting and all that. So far we've not charged for admission. (If we did charge, who'd patrol the corridors to make sure everyone had paid?) Plus, everyone from staff to speakers donated their time. That really made SOG unique.

End SOG. That's it. Kaput. No more. When I mention this option to folks who call in, I always get a stunned silence. (Thanks.) But some have offered to

Continued on page 72

YOU'LL LOVE THESE UTILITIES.



SAYWHAT?! The lightning-fast screen generator

It doesn't matter which language you program in. With Saywhat, you can build beautiful, elaborate, colorful screens in minutes! That's right. Truly *fantastic* screens for menus, data entry, data display, and help-panels that can all be displayed with as little as one line of code in *any* language. Batch files, too.

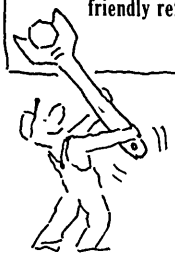
With Saywhat, what you see is *exactly* what you get. And response time is snappy and crisp, the way you like it. That means screens pop up instantly, whenever and wherever you want them.

Whether you're a novice programmer longing for simplicity, or a seasoned professional searching for higher productivity, you owe it to yourself to check out Saywhat. For starters, it will let you build your own elegant, moving-bar menus into any screen. (They work like magic in any application, with just one line of code!) You can also combine your screens into extremely powerful screen libraries. And Saywhat's remarkable VIDPOP utility gives all languages running under PC/MS-DOS, a whole new set of flexible screen handling commands. Languages like dBASE, Pascal, BASIC, C, Modula-2, FORTRAN, and COBOL. Saywhat works with all the dBASE compilers, too!

With Saywhat we also include a bunch of terrific utilities, sample screens, sample programs, and outstanding technical support, all at no extra cost. (Comprehensive manual included. Not copy protected. No licensing fee, fully guaranteed). **$49.95**

WE GUARANTEE IT!

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



TOPAZ. The breakthrough toolkit for Turbo Pascal 5.0

If you'd like to combine the raw power and speed of Turbo Pascal with the simplicity and elegance of dBASE, Topaz is just what you're looking for. You see, Topaz (our brand new collection of units for Turbo Pascal 5.0) was specially created to let you enjoy the best of *both* worlds. The result? You create truly dazzling applications fast. And no wonder. Topaz is a comprehensive toolkit of dBASE-like commands and functions, designed to help you easily write outstanding, polished programs. Now you can write in Pascal using **SAYs** and **GETs, PICTURE** and **RANGE** clauses, then **SELECT** and **USE** databases and **SKIP** through records, plus **BROWSE, INDEX ON, FIND, PACK, APPEND,** a complete set of time and date handling routines and lots more.

In fact, we've emulated nearly one hundred *actual* dBASE *commands and functions*, and even added *new* commands and functions! All *you* do is declare Topaz's units in your source code and you're up and running!

The bottom line? Topaz makes writing sophisticated Pascal applications a snap. Data entry and data base applications come together with a minimum of code and they'll always be easy to read and maintain.

Topaz comes with a free code generator that automatically writes all the Pascal code you need to maintain a dBASE file with full-screen editing. Plus truly outstanding technical support, at no extra cost. (Comprehensive manual included. Not copy protected. No licensing fee, fully guaranteed). **$49.95**

ORDER NOW. YOU RISK NOTHING. Thousands of satisfied users have already ordered from us. Why not 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! Buy Saywhat?! and Topaz together for just $85 (plus $5 shipping & handling). That's a savings of almost $15.

To order: Call toll-free

800-468-9273

In California: **800-231-7849**
International: 415-467-6840

The Research Group
100 Valley Drive
Brisbane, CA 94005

☑ **YES.** I want to try:

Saywhat?! your lightning-fast screen generator, so send _____ copies ($49.95 each, plus $5 shipping & handling) subject to your iron-clad money-back guarantee.

Topaz, your programmer's toolkit for Turbo Pascal 5.0, so send _____ copies ($49.95 each, plus $5 shipping & handling) subject to your iron-clad money-back guarantee.

☐ **YES.** I want to take advantage of your special offer! Send me _____ copies of both Saywhat?! and Topaz at $85 per pair (plus $5 shipping & handling). That's a savings of almost $15.

NAME _____

ADDRESS _____

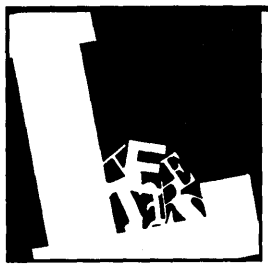
CITY _____ STATE _____ ZIP _____

☐ Check enclosed ☐ Ship C.O.D. ☐ Credit card

_____ Exp. date _____ Signature _____

THE RESEARCH GROUP

Reader Service Number 129



Letters

Oops Indeed

I thoroughly enjoyed Micro C's Object Oriented Programming issue (#44). Indeed, I am one of the religious fanatics of the WWLFSOG (We Who Live For SOG) sect that Micro C caters to so well. But I seem to be suffering from an acute case of Groping for Acronyms Syndrome (GAS).

In issue #44's articles and advertisements, the letters OO, OOP, or OOPS are used for the same phrase — Object Oriented Programming. Zak Urlocker calls it OOP on page 18, Bruce Eckel calls it OO on page 32, the Digitalk ad on page 12 uses OOPS (the picture on page 13 does include the word "system"), and the Zortech ad on the inside back cover uses both OOP and oops (oops?).

First, I think OO is incomplete (maybe Bruce saw fireworks as he bathed in the Pleasure of Object Oriented Programming (POOP)). Then there's OOPS. What's the "S" for? Software? System? Or is it just Silent? I suspect some advertisers think it's cute or has a nice ring. But, frankly, oops is something I say when I've made a misteak.

I can only hope these Fanatical Object Oriented Language Sellers (FOOLS) drop the "S" before we all Generate Adverse Gossip (GAG).

Larry Hollibaugh
2714 SE 19th Ave.
Portland, OR 97202

A Dose Of Reality

With all the Object hype being published, there is one article which I fear will not get the attention it deserves. In the May '88 issue of *IEEE Software*, the creator of C++ published an article titled, "What is Object Oriented Programming?"

Before some readers groan, "Oh God, not another one," I must say this article is *different*. I may do it injustice, but to brutally sum up the article:

In the beginning there was flat programming a lá BASIC, but it had a,b,c wrong with it. Trying to fix a,b,c gave us Pascal and procedures. But now we had d,e,f problems. Trying to fix d,e,f yielded modular programming, but with problems g,h,i. Fixing g,h,i yielded abstract data type programming, but with problems j,k,l. We fixed j,k,l with object oriented programming, but now we have problems m,n,o, and we have no idea how to fix these!

The article goes on to say much more, but the fact that it showed objects as part of a progression rather than being brought down off the mountain carved on stone tablets just

Continued on page 80

From C To Shining C

Greenleaf Has

The Best Reputation

Over the last five years the name Greenleaf has become synonymous with quality software.

As a leader in the professional C programming library market, we are recognized by the industry in general, and our customers in particular, as the premier supplier of quality products that consistantly lead the industry in performance.

Our customers rate our products, support and documentation EXCELLENT. It's a fact that almost 50% of our clients bought one or more of their Greenleaf products without ever considering the competition.

Greenleaf's dedication to the attainment of excellence is driven by our customers. They tell us what they need and want in order to perform their jobs as professionally as possible. It is this partnership in excellence that has made Greenleaf the most respected provider of C language libraries in the world.

If you don't own a Greenleaf product, talk to one of our users. We suspect that you will be a Greenleaf owner soon. If you already own Greenleaf products we thank you for your trust and support and we look forward to our continued partnership.

GREENLEAF Software©

Greenleaf Software Inc.
Bent Tree Tower Two - Suite 570
16479 Dallas Parkway
Dallas, TX 75248

1-800-523-9830

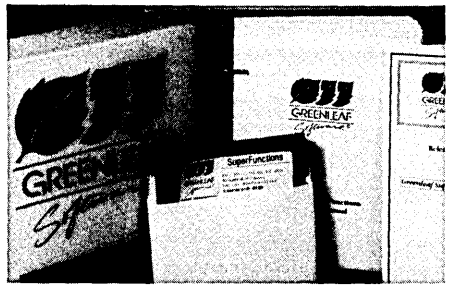
Texas and Alaska: (214)248-2561
Telex: 559-068

Reader Service Number 146

The Best Products

Greenleaf SuperFunctions

An advanced collection of functions for the experienced C programmer. Functions include expanded memory support, extended mouse functions, advanced time manipulations, unique date functions, DOS critical error handler along with windows, menus and keyboard functions.



Greenleaf Business MathLib

The only product available that gives the programmer a complete set of math functions while providing the accuracy of BCD math to the C language. Business functions such as compound interest, internal rate of return, cash flow analysis, bond calculations, amortization, array processing, statistics, trigonometry and print flexibilty make financial output a breeze.

Greenleaf DataWindows

This easy to use and powerful windowing C library features overlaid logical windows, transaction data entry and three styles of menus. Functionality such as context sensitive help, menus within data entry, keyboard idle, list boxes and ROM BIOS redirection. Pop-up, pull-down and Lotus style menus are also included.

Greenleaf MakeForm

Working in conjunction with DataWindows, MakeForm will reduce the time it takes to design and build a human interface. Forms stored in special files enable you to change them without recompiling your application. Cut and paste between multiple forms while using the fully supported editing features.

Greenleaf Comm Library

XMODEM, XMODEM CRC, RTS/CTS, and XON/XOFF protocols are all available on up to 17 ports with this interrupt driven, asynchronous communications library. Your programs can fly at up to 19,200 buad.

Greenleaf Functions

The original. Still as popular and powerful as it was 5 years ago, Over 300 functions to support your programming efforts. DOS and BIOS calls, string color text, time, date, and graphics functions are just the beginning.

The Art of Disassembly;

Getting To The Source Of The Problem When The Object's The Data

Disassembly is largely misunderstood. Many folks think it's some kind of reverse voodoo. Others think it's a nefarious sport better left to hackers. But the reasons to disassemble are many: including silencing an irritating bell, or recreating source after a hard disk crash.

I've divided this article into two parts. In the first I do a general disassembly of disassembly; in the second I use a disassembler called Sourcer.

What Is Disassembly?

Disassembly is the process of taking machine code and translating it back into assembly language. The program that does this is called a disassembler. There is no requirement that the original code be written in assembly, but no matter what language was originally used, disassembly will only produce the assembly translation of the machine code. An example of a disassembler is DEBUG's unassemble command.

The disassembler can translate numeric instructions into assembly mnemonics, but not every byte or word in an .EXE or .COM file is an instruction. There are tables for data, embedded messages, and so on, so it's often up to a human to make the disassembler's output useful.

A knowledgeable operator can greatly improve the quality of the final disassembled code by making multiple passes, each time gleaning new information. This new information can be fed back to the disassembler to improve the quality of the next run.

Why Disassemble?

Now the question, why on earth would anyone but a hacker want to disassemble code? There are many reasons.

First, to find out how the program works. To practice safe computing, you may want to check out any new pro-

By B. H. Flusche, Jr.
3892 E. Geddes Ave.
Littleton, CO 80122

grams to be sure they do not contain a virus.

Once you know how the program works, you can modify the program. I have a utility program called "browse" which allows me to view an ASCII file. It can move though the file one line at a time, or one page at a time.

However, to search for something, I must page a screen at a time until I find it. Tedious. So I disassembled the program and added a string search feature.

A third reason is to document a program. Sometimes programmers produce only object code as their finished product. With a disassembler, the company can at least regain the assembly code.

You can also reverse engineer a program. However, be very careful since distributing the results would probably be illegal.

How A Disassembler Works

There are two types of disassembler programs. The first assumes that every byte or word it reads is an instruction. This works until the disassembler reaches the first data area. The "unassemble" command in DEBUG is a good example of this kind of disassembler.

Figure 1 is a printout from DEBUG. Is this data, instructions, or both?

The main disadvantage to this method is that it requires a great deal of highly skilled operator intervention.

More sophisticated disassemblers also assume the binary code is all instructions, but they can later correct themselves and determine which addresses are data and which are instructions. But even these disassemblers will sometimes need help. Sourcer falls into this category and the intervention it requires is discussed in detail later.

A Different Type Of Disassembler

The second type of disassembler makes no initial assumptions about data or instructions. In this case the operator must tell the disassembler the entry point into the target program. The disassembler then starts translating instructions from that point until it encounters an unconditional jump or a subroutine return. It then stops translating and stores the beginning and ending addresses that define this portion of the program as instructions.

A knowledgeable operator can greatly improve the quality of the final disassembled code by making multiple passes, each time gleaning new information.

Figure 1 — DEBUG Unassembly of Simple.com

```
118B:0100 EB61      JMP   0163
118B:0102 90        NOP
118B:0103 0A0D      OR    CL,[DI]
118B:0105 48        DEC   AX
118B:0106 65        DB    65
118B:0107 6C        DB    6C
118B:0108 6C        DB    6C
118B:0109 6F        DB    6F
118B:010A 2C20      SUB   AL,20
118B:010C 706C      JO    017A
118B:010E 65        DB    65
118B:010F 61        DB    61
118B:0110 7365      JNB   0177
118B:0112 20656E    AND   [DI+6E],AH
118B:0115 7465      JZ    017C
118B:0117 7220      JB    0139
118B:0119 796F      JNS   018A
118B:011B 7572      JNZ   018F
118B:011D 206E61    AND   [BP+61],CH

/***/
```

As the disassembler translates, it notes the destination addresses of all conditional jumps and subroutine calls that it encounters. It stores these addresses in a table and uses them as new entry points into the program since it can be sure that these addresses are for instructions.

The disassembler goes back and forth through the code until it has used up all the destination addresses. Thus, it generates a map of the instructions. It assumes everything else is data.

What this disassembler calls instructions, are usually instructions. However, the rest isn't always data. Instructions will be missed if they aren't called by the rest of the program.

There are two primary causes for uncalled code:

1. Multiple external entry points to the program.
2. Non-accessible code produced by compilers.

The non-accessible code is not important since it is never executed. However, multiple entry points should be caught by the operator.

The listing produced by the disassembler should be in a standard assembler format. If not, you will have more difficulty reassembling the program after modifications. A good disassembler will even add the assembler directives.

The disassembler has no knowledge of what the program does or how it does it. The listing will be labeled for the explicitly called data and instruction locations, but they will have some sequential format such as "loc_12" or "data_24" instead of intelligible names like

"get_rs232_input" or "screen_offset."

One of your major tasks is to work through the code, figure out what's happening, and change the sequential labels to meaningful ones. This process is complicated if the disassembler mistakes data for instructions or vice versa.

Most of the commercial disassemblers advertise a feature called commented code where the disassembler automatically puts comments in the listing. This is not really as useful as it seems. Remember the disassembler does not understand the target program. So its comments can only tell you a bit about *what* is happening, not *why*.

These three goals, deciphering program logic, adding comments and labels, and ensuring accuracy are synergistic. As you improve one, it's easier to improve the others.

For example, Sourcer will comment a LODSB instruction with ";string [si] to al." This helps if you're an inexperienced 8086 programmer who would otherwise have to look it up, but what you really want to know is the purpose of the instruction. In this case, it might be "; get byte from filename for parsing."

How The Disassembly Process Works

Now that we have seen what a disassembler does, and doesn't do, let's look at the overall disassembly process. The basic steps are:

(1) Examine the machine code using DEBUG.

(2) Feed the machine code to the disassembler.

(3) Examine the disassembler's output.

(4) Determine new settings for the disassembler.

(5) GOTO step (2) until the listing is satisfactory.

Steps (2) and (5) are self-explanatory and step (4) is disassembler specific, so we will concentrate on steps (1) and (3).

In step (1) you use DEBUG to dump the target code. This is to give you a general idea which addresses contain data. Look for ASCII strings and initialized data areas.

ASCII strings are readily recognizable from the dump listing. Record their beginning and ending addresses so that you can tell the disassembler that this is a data area. (It will appreciate all the help you can give it.) By looking closely at the strings, you will be able to glean additional information.

If the strings end with a zero, they are in the ASCIIZ form. This means the program was probably compiled with a high level language (for instance, C uses ASCIIZ strings). If the strings end with a $, then the source was probably an assembly program.

You will be able to recognize the initialized areas if you get large areas of the same byte. In particular an area initialized to zero is easy to notice.

If you are not sure, use DEBUG to unassemble the code. If you get the same instruction 100 times, then it's probably data. Two in a row, well, they could be data or instructions. When in doubt, assume instructions.

Step (3), examining the disassembler's output, is the hardest part of disassembly. This is a job for an experienced assembly programmer. The key to successfully disassembling the target program is to examine the listing, instruction by instruction, to decipher the original program logic.

Deciphering The Listing

You must understand the target program so that you can add labels and comments, and ensure the disassembled version is an accurate recreation of the original. The mechanics of deciphering the listing vary, but I have found the following effective:

1. Start at the beginning and follow the program flow. When you come to a conditional jump, try to figure out why the jump exists. Record the conditions and take both branches. You may not initially know why the jump is there, but by comparing the instructions following both branches you can figure it out.

2. When you get to a subroutine call, note the contents of the registers, then go to the subroutine code. Try to determine the purpose of the subroutine. Record which registers it uses for input and which it uses to return values.

3. Don't try to get everything straight

Figure 2 — Simple.com Object Code

```
2EED:0100  EB 61 90 0A 0D 48 65 6C-6C 6F 2C 20 70 6C 65 61   .a...Hello, plea
2EED:0110  73 65 20 65 6E 74 65 72-20 79 6F 75 72 20 6E 61   se enter your na
2EED:0120  6D 65 3A 20 24 0A 0D 57-65 6C 6C 2C 20 24 2C 20   me: $..Well, $,
2EED:0130  68 6F 77 20 64 6F 20 79-6F 75 20 6C 69 6B 65 20   how do you like
2EED:0140  63 6F 6D 70 75 74 65 72-73 3F 0A 0D 24 15 00 20   computers?..$..
2EED:0150  20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
2EED:0160  20 20 20 8D 16 03 01 B4-09 CD 21 BA 4D 01 B4 0A      .......!.M...
2EED:0170  CD 21 BA 25 01 B4 09 CD-21 8A 0E 4E 01 32 ED BE   .!.%....!..N.2..
2EED:0180  4F 01 B4 02 E3 07 AC 8A-D0 CD 21 E2 F9 B4 09 BA   O.........!.....
2EED:0190  2E 01 CD 21 B8 08 4C CD-21                        ...!..L.!

/***/
```

Figure 3 — Initial Definition File, and Listing

```
--------------- Section 1  Control Information -----------------
keep segments = def
Input filename = simple.com

--------------- Section 2  Range Definitions -------------------
  begin     end   default   seg
 seg:off    off   ds   es   type
 -------   ----  ---- ----  ----
sega:0100  0102  sega sega  code
sega:0103  0162  0000 0000  data
sega:0163  0198  sega sega  auto

--------------- Section 3  Reference Definitions ---------------
 seg:off  type     options and label
 -------   --  --------------------------
sega:0103  da        msg1
sega:0125  da        msg2
sega:012e  da        msg3


simple.lst   Example disassembly   Sourcer Listing v1.68   Page 1

Editor's note: We've taken some small liberties with this listing
to make it fit on the page. Please excuse the line wraps.

= 014E                          data_1e equ     14Eh
; (0000:014E=0)

                                codeseg         segment
                                assume  cs:codeseg, ds:codeseg


                                org     100h

                                simple  proc    far

81BF:0100                       start:
81BF:0100  EB 61                jmp     short loc_1
81BF:0102  90                   db      90h
81BF:0103  0A                   msg1    db      0Ah
81BF:0104  0D 48 65 6C 6C 6F            db      0Dh, 'Hello, please
                                        enter your na'
81BF:010A  2C 20 70 6C 65 61
81BF:0110  73 65 20 65 6E 74
81BF:0116  65 72 20 79 6F 75
81BF:011C  72 20 6E 61
81BF:0120  6D 65 3A 20 24               db      'me: $'
81BF:0125  0A                   msg2    db      0Ah
81BF:0126  0D 57 65 6C 6C 2C            db      0Dh, 'Well, $'
81BF:012C  20 24
81BF:012E  2C                   msg3    db      ','
81BF:012F  20 68 6F 77 20 64            db      ' how do you like
                                        computers?', 0Ah
81BF:0135  6F 20 79 6F 75 20
81BF:013B  6C 69 6B 65 20 63
81BF:0141  6F 6D 70 75 74 65
81BF:0147  72 73 3F 0A
81BF:014B  0D 24                        db      0Dh, '$'
81BF:014D  15 00                        db      15h, 0
81BF:014F  0014[20]                     db      20 dup (20h)
81BF:0163                       loc_1:
81BF:0163  8D 16 0103                   lea     dx,msg1
```

continued on page 12

the first time. Get as much as you feel you can, then rerun the disassembler with the new information. You will be surprised at how much easier the second examination will be since the new listing will have some descriptive labels.

Instead of trying to remember that "sub_23" reads a sector and "sub_15" writes a sector, you will have changed the labels to "read_sector" and "write_sector." If you can make the disassembler put comments in its listing, do so. You want as much information as possible on the listing.

4. As you examine the listing, be sure that the data is really data, and that the instructions are really instructions. Ideally you could check this before the detail exam, but unfortunately the exam is the only real way to know.

These three goals, deciphering program logic, adding comments and mnemonic labels, and ensuring accurate translation have a synergistic effect. As you improve one, it's easier to improve the others. Conversely you cannot complete one until the others are nearly complete. Working on these three simultaneously is the secret to disassembly.

Sourcer

Now that we have thoroughly examined general disassembly, let's look at Sourcer. In the rest of this article, I'll explain how Sourcer works, how to use it, and some things about it not mentioned in the operators manual. If you haven't read Sourcer's user manual, the last section may be hard to follow.

How Sourcer Works

Sourcer is a multipass disassembler where you set the number of passes from 2 to 5. On the first pass Sourcer assumes all the code in the target program is made up of instructions. As it goes through the code, it stores the symbol addresses for use in the next pass. It maintains three symbol tables; one each for data, locations, and subroutines. On the next pass it looks at the symbol tables and makes sure the item referenced is valid.

With each additional pass, Sourcer refines the symbol tables and its ability to distinguish instructions from data. The number of entries in each table may decrease as invalid entries are recognized and removed. On the last pass it writes the listing to disk.

Sourcer works through the program sequentially. It does not take any jumps or subroutine calls, and as it works through the code it maintains a set of simulation registers which it tries to keep

Continued from page 11

```
                                ; (81BF:0103=0Ah) Load effective addr
81BF:0167  B4 09                        mov     ah,9
81BF:0169  CD 21                        int     21h
                                ; DOS Services ah=function 09h
                                ;  display char string at ds:dx
81BF:016B  BA 014D                      mov     dx,14Dh
81BF:016E  B4 0A                        mov     ah,0Ah
81BF:0170  CD 21                        int     21h
                                ; DOS Services  ah=function 0Ah
                                ;  get keybd line, put at ds:dx
81BF:0172  BA 0125                      mov     dx,125h
81BF:0175  B4 09                        mov     ah,9
81BF:0177  CD 21                        int       21h
                                ; DOS Services  ah=function 09h
                                ;  display char string at ds:dx
81BF:0179  8A 0E 014E                   mov     cl,ds:data_1e
                                ; (0000:014E=0)
81BF:017D  32 ED                        xor     ch,ch
                                ; Zero register
81BF:017F  BE 014F                      mov     si,14Fh
81BF:0182  B4 02                        mov     ah,2
81BF:0184  E3 07                        jcxz    loc_3
                                ; Jump if cx=0
81BF:0186                       locloop_2:
81BF:0186  AC                           lodsb
                                ; String [si] to al
81BF:0187  8A D0                        mov     dl,al
81BF:0189  CD 21                        int     21h
                                ; DOS Services  ah=function 02h
                                ;  display char dl
81BF:018B  E2 F9                        loop    locloop_2
                                ; Loop if cx > 0
81BF:018D                       loc_3:
81BF:018D  B4 09                        mov     ah,9
81BF:018F  BA 012E                      mov     dx,12Eh
81BF:0192  CD 21                        int     21h
                                ; DOS Services  ah=function 09h
                                ;  display char string at ds:dx
81BF:0194  B8 4C08                      mov     ax,4C08h
81BF:0197  CD 21                        int     21h
                                ; DOS Services  ah=function 4Ch
                                ;  terminate with al=return code
                                simple  endp

                                codeseg ends

                                end         start

/***/
```

Figure 4 — Original Program Simple.asm

```
codeseg segment
        assume  cs:codeseg, ds:codeseg
        org     100h
start:  jmp     begin

msg1:   db      0Ah, 0Dh, 'Hello, please enter your name: $'
msg2a:  db      0Ah, 0Dh, 'Well, $'
msg2b:  db      ', how do you like computers?', 0Ah, 0Dh, '$'
name_buf:       db      21
        db      0                       ;buffer for name input
        db      20 dup (' ')

begin:  lea     dx,msg1                 ;ask for users name
        mov     ah,9
        int     21h
        mov     dx, offset name_buf     ;get name via string inp
        mov     ah,0AH
        int     21h
        mov     dx, offset msg2a        ;write 1st part of response
        mov     ah,9
        int     21h
        mov     cl, byte ptr name_buf + 1       ;get char count to cl
        xor     ch,ch                   ;count in cx
        mov     si, offset name_buf + 2 ;get buffer addr
        mov     ah,2                    ;set up for char disp
        jcxz    done                    ;check for null string
disp_char:      lodsb                   ;get char from buffer
```

```
        mov     dl,al                   ;mov to dl for service 2
        int     21h
        loop    disp_char               ;loop til all char displayed
done:   mov     ah,9                    ;write rest of msg
        mov     dx, offset msg2b
        int     21h
        mov     ax,4C08h                ;terminate process
        int     21h
codeseg ends
        end     start

/***/
```

Figure 5 — Example of Range Overlap

```
Range Section fragment

sega:0100 0102 sega sega code
sega:0100 0100 0000 0000 data           Note overlap


Listing fragment

81BF:0100               start:
81BF:0100  EB 61                jmp     short loc_1
81BF:0102  90                   db      90h
81BF:0100  EB 61 90             db      0EBh, 61h, 90h
81BF:0103  0A           data_1  db      0Ah

/***/
```

updated. These registers are used in producing comments or finding the address of jump tables.

Since Sourcer does not jump or loop, the simulation register will be incorrect if the program changes a register and then jumps to a previous location.

You control Sourcer via a command menu or a definition file.

Definition File

The definition file has three sections.

The first is for control information and duplicates all of the commands available from the menu.

Section two is for range definition and it specifies how Sourcer processes a range of bytes.

The third section is for reference definition. It specifies how a particular address is to be treated. Figure 3 has sample definition files.

The range definition tells Sourcer which portions of the target program are data, which are instructions, which are unknown, and which are ROM. Sourcer will read these directions, and it will skip the data areas during the first pass. This prevents bogus labels from being entered into the symbol tables.

Reference definition allows finer control. You specify individual addresses and direct how Sourcer is to treat those addresses. For data you can specify a byte, word, dword, ASCII, or even a structure. If you tell Sourcer that the data item is an offset, it will add the offset directive and target label.

Sometimes Sourcer will override your range definitions. This can happen if you miss a data area, and Sourcer erroneously translates the data as a jump to the area you defined as data. Sourcer will then believe the bogus jump and label the area (you previously defined as data) as instructions.

Example Disassembly Using Sourcer

Let's look at an example disassembly process using Sourcer. Figure 2 is a DEBUG dump of the program "simple.com." This is the same code as in Figure 1. Note the possible data areas. Figure 3 is the define file derived from looking at the DEBUG dump and Sourcer's output based on that definition file. Compare the disassembled listing to the original program in Figure 4.

Sourcer Secrets And Tips

(1) Some of the definition file control commands are sequence sensitive. For instance, the "Header" command must be after the "Input filename" command or you'll get the default header. The "keep segments" command must be before the "Input filename" command.

(2) Sourcer will accept almost any

string as a label, but it's a good bet your assembler won't. If you plan on reassembling the code, save yourself some pain later and use only labels acceptable to your assembler.

(3) If the target program contains an INT 20H, Sourcer does not recognize this as the end of the procedure and does not put the required "endp" directive into the listing.

(4) Sourcer has a number of built-in labels which it uses when it feels the need. They include: "copywrt" for a copyright statement, "start" for the program's entry point, "int_XXh_ext" for external interrupt entries, and "strategy" and "interrupt" for device drivers. You have no control over these labels.

(5) If you use the "repeat" option to define a data area and you miscount the number to be repeated, Sourcer will add bogus bytes as needed. This can rear its ugly head if you have defined ASCII text and it contains a copyright. The built-in copywrt label will be applied, and your repeat count will be wrong.

(6) When you define segments via the range definition section, you must explicitly specify the last byte's address. There is no end of code operator. The best way to find the address of the last byte is to load the program with Sourcer then note the last address as shown on the main menu. Since Sourcer will process all the addresses you specify, it will stop where you told it regardless of the length of the program.

(7) The addresses in the range definition section should be contiguous. If there is a gap, Sourcer will skip over the gapped code and generate an org statement. This ensures the subsequent addresses are in their proper place, but the gapped code is removed.

(8) If two range definitions overlap, Sourcer will process the overlapped code twice. If the first range is a data definition, and the second is code definition, Sourcer will output both the data and code. Figure 5 illustrates this.

(9) The "keep segments" command is not explained well. It tells Sourcer which of two sets of range definition to use. The first is internal to Sourcer and is created as Sourcer processes the binary code. The second is the one you write in the definition file and is optional. The "keep segments" command tells Sourcer to use either: its internal definition (FILE), the one from the definition file (DEF), or to use both (BOTH).

If you haven't written a range definition, use the default FILE option. If you have written a range definition that includes the target program's address, use the DEF option. If your range definition contains only addresses outside the program's areas (such as the BIOS data area), use the BOTH option. Sourcer will process the code twice if your range definition covers the program's address space and you specify BOTH.

(10) Labels can be added to the disassembly listing at will. However you should restrict their use so that they are present only when they are the target of an instruction. For instance, if you add extraneous labels to instructions, it will be unclear which labels are decorations and which are the destinations of jumps and calls. Unfortunately Sourcer requires you to specify a label in order to add a comment. I recommend you use a sequential label notation so that each label is unique and is distinguished from functional labels.

(11) If Sourcer's analyzer makes a mistake and translates data as instructions, these bogus instructions can cause trouble farther down the line. Any addresses the bogus instructions reference as data will be marked as data, even if they are really instructions. Bogus jumps or calls will mark the destinations as instructions and these second generation bogus instructions can cause additional trouble. If you find that Sourcer made a mistake, don't just fix the part you found, backtrack until you find the root cause. The cause will always be bogus instructions.

(12) The more passes you specify, the better the analyzer can separate data from instructions. Five passes will produce the best resolution, and you can use the cross reference option. By using cross references, you can backtrack to the source error.

Sometimes a label will not have a cross reference. This indicates the label was created by a bogus instruction that the analyzer has converted back to data. You can find the cause by reducing the number of passes and then using the range definition to fix the problem.

Sourcer
V Communications
3031 Tisch Way Suite 905
San Jose, CA 95128
(408) 296-4224

♦ ♦ ♦

ERAC CO.

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

AT

Motherboard 6 & 10 Meg
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

$1581

EGA ADD $449
40M HD ADD $150
6 & 12 MHz ADD $73

BABY AT

Motherboard 6 & 10 Meg
Zero Wait State
8 Expansion Slots
80286 Processor
Math Co-Processor Option
640K RAM On-Board
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
Mini AT Case with Turbo &
Reset, Hard Drive Light and
Keyboard Disable Switch
Amber Graphics Monitor

$1531

EGA ADD $449
40M HD ADD $150

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

★ ★

$999

EGA ADD $429
40M HD ADD $150
5 & 10 MHz ADD $21

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
F Cells	2.50

GEL CELLS

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

ROBOTICS

5V DC Gear Motor with 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
Capacitor, .47farad 10V 1"x1¾"	4.00
Solar Cells .5V .5A, .8"x1.6"	2.50
High Voltage Power Supply	
Input: 15-30V DC	
Output: 100V 400V 16KV	6.50

KAYPRO EQUIPMENT BARGAINS

9" Green Monitor	$50
9" Amber CRT	$45
Keyboard	75
PRO-8 Mod. to your board	149
Host Interface Board	15
Replacement Power Supply	50

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

★ NEW ★ CPM COMPUTERS

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

ELGAR UNINTERRUPTIBLE POWER SUPPLIES

560 Watt MODEL IPS560 $650
Sinewave, 560W complete w/batteries.

400 Watt MODEL SPR401 $149
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, not guaranteed.

NEW 24V INTERNAL BATTERY $75

POWER SUPPLIES

0-8VDC 100A Metered	$249.00
Volt & Current Regulated	
5V/1A. -5V/.2A, 12V/1A, -12V/.2A, -24V/.05A	9.90

HOURS: Mon. - Fri. 9 - 6 — Sat. 10 - 4
MINIMUM ORDER — $15.00
TERMS: VISA, MasterCard, Certified Checks, Money Order, NO COD. Visa and MasterCard add3%. *Personal checks must clear BEFORE we ship.* Include shipping charges. California residents add 6½% Sales Tax. For more information please call.

CPU & SUPPORT CHIPS

MC68000-8 CPU	$8.00
Z80 CPU	.75
Z80A CPU	1.50
Z80 CTC	1.50
Z80A PIO	2.00
Z80A SIO	5.00
8088	6.50
8089-3	6.50
D8284A	2.50
SIP DRAM 256-12	7.00
4164-10	2.50
4164-12	2.10
1793	6.00
1797	7.00
6845	5.00
VC3524 Switching Regulators	5.00
1458 Dual Op-AMP	.70
LM2877P 4W Stereo Amp Dual	2.50
MB81464-15	2.75
2716	3.00
2732	3.25
2764	3.50
27C128-1	9.00
74HC00	.38
74LS125	.30
74LS373	.50
74LS174	.30

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/1¼A, 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

TEST EQUIPMENT

OSCILLOSCOPES

TEK 7403N/7A18N/7B50A 60 MHz	$650
TEK 465 Dual Trace 100 MHz	1000

ANALYZERS

TEK 491 10MHz - 40 GHz	$4500
Nicolet 500A 1 Hz 0-100 KHz	1800
Biomation 805 Waveform Rcrdr	259
Biomation 8100 2-Channel Waveform Recorder	795
HP1600A Logic Analyzer	600
HP1600A/1607A Logic Anlyzr	1000

FREQUENCY COUNTER

Optronics 550 MHz	$100

DBASE BOOK OF BUSINESS APPLICATIONS *by Michael J. Clifford*
Reg. $19.95 ***NOW ONLY*** **$3.00**

Reader Service Number 93

Handling Interrupts With Any C

A Detailed Look At Supporting Interrupts

Micro C is famous for cramming lots of useful information into very little space. This article may be too long to qualify as "little space" but it's so crammed with information on implementing interrupt service routines (ISRs) that you've got to read on. (Find a spot where you won't be interrupted.) This is a great one, folks.

One idea which Borland added to Turbo C really bowled me over when I first saw it. It's their "interrupt" type modifier! Hooray for Borland, but not everyone uses Turbo when working in C.

I found another simple but impressive idea hiding in a listing for a low level driver. (I don't remember where I saw the listing.) The listing contained assignments to CPU registers! What incredibly nice code to read — no structure references full of obscure object names. Very neat!

I decided to combine these two ideas into support headers and library functions. The CPU registers were easy. A long list of #defines did most of the job. Adding support for high level interrupt handlers wasn't too tricky either, and it's turning out to be more useful than I expected.

In this article, I'll describe two main modules, each module a group of functions. The first group supports existing system interrupt and far call functions. The second group contains support routines to allow standard C functions to service interrupt requests or be called by external programs.

Everything was written to be compiled with Manx Aztec C 86, version 4.10a. Other compilers might run into minor problems handling the assembly code, but it shouldn't be anything serious.

CALLs & INTs

The functions in this group are super simple. They do software interrupts and far calls. The work is done by two functions in the CALL.C module, SEG86 and CALL_8086(). (See Figure 1.) They're written in a mix of assembly level spaghetti and C. The third function, INTN_8086(), and fourth, regs_dump, I'll describe shortly.

```
#include <call.h>

unsigned int far call_8086( ri, ro )
_ _regs far *ri, *ro;

unsigned int far intn_8086( ri, ro )
_ _regs far *ri, *ro;

unsigned int far seg86( r )
_ _regs far *r;

void _ _regs_dump( r )
_ _regs far *r;
```

ri and ro are pointers to _ _regs structures; *ri contains the register values passed and *ro contains the values returned.

call_8086() does a far call to the far function at location ri->CS:IP. intn_8086() assumes that the desired interrupt number is in ri->IP, and does a software interrupt to that handler. Both functions return the result in the PSW register.

seg86() copies the DS, ES, CS, SS and PSW register values into the corresponding objects in the _ _regs struct at *r. _ _regs_dump() displays the contents of the _ _regs structure *r on STDOUT.

The _ _regs structure is typedefed in the CALL.H header (see Figure 2). It's a complicated structure full of unions of structures of pointers and unsigned chars and ints, representing the CPU registers. The main difference between this structure and the ones included with most 8086 compilers is the organization of the registers. You can directly access some of the more commonly used register pairs: ES:BX, DS:DX and DX:AX. I also used bit fields to fully define the PSW (Processor Status Word).

You'll see the list of #defines following the structure definition in CALL.H. The list names each CPU register, its lower and upper halves, plus some register pairs and the PSW flags. Each register/flag name expands into the notation required to get at it in a _ _regs structure.

An almost identical list follows, only this time the names expand into a reference to an object within the _ _regs struct _REGS. Nothing called _REGS is declared in the header, so you'll have to declare a structure or macro by that name before using any of the macros.

The point of the second list is to eliminate completely all structure/union notations so you can work directly with the desired registers, register pairs and CPU status flags.

The last group of #defines expands into function calls. call86(pc) does a far call to the function at void (far *pc)(), int86(i) does a software interrupt to interrupt i, and pcdos(f) loads the DOS function number f into AH, then calls DOS. Each macro expects to use the _REGS structure for both the "input" and "output" registers.

ISRHs

When an ordinary C function gets called, it expects the proper values in the segment registers, expects parameters on the stack, and expects the stack to be big enough for automatic storage.

An ISR Handler can't expect anything. To get a C function to handle an interrupt, you'll need to add some support code between the interrupt and the function call (and before the return-from-interrupt).

I figured the easiest solution would be to write a "straight" version of the support code, then have another routine copy it into a data, structure for each ISR handler. The file ISRHT.C contains this "isr handler template." (See Figure 3.) The ISRH data structure contains space for the code, and variables are positioned so you can modify the code.

This way, the function isr_install() can install a C function as an interrupt handler, then isr_restore() can put the previous vector back into the interrupt table when the program's done. It's that simple!

By Sam Azer
1405 Bishop St Suite 100
Montreal, PQ Canada H3G 2E5

Figure 1 — CALL.C far calls and software interrupts from C

```
#include <call.h>

/*      SEG86

        This function copies the 8086 segment register contents
        into the _ _regs register structure pointed at by ri.
*/

#pragma vpindex seg86
unsigned int far seg86( ri )
_ _regs far *ri;
{
#asm
        push  es
        push  es
        les   bx, _RI
        pop   ax
        mov   _ES, ax
        mov   _DS, ds
        mov   _SS, ss
        mov   _CS, cs
        pushf
        pop   ax
        mov   _PSW, ax
        pop   es
#endasm
}

/*      CALL_8086()

        This function calls the code at _PC in *ri. It doesn't
        check for messed up stacks, but is reentrant.
*/

#pragma vpindex call_8086
unsigned int far call_8086( ri, ro )
_ _regs far *ri, *ro;
{
#asm
        push  si          ;save C regs
        push  di
        push  ds
        push  es

        les   bx, _RI     ;get pointer to input regs

        push  bp          ;we're going to need this later...

        push  cs          ;here's where the routine that we'll call
        mov   ax, offset retadr
        push  ax          ;will return

        push  _CS         ;put address to call on the stack
        push  _IP

do_it:  push  _ES         ;can't touch these now...
        push  _BX         ;  so put them on the stack

        mov   ax, _AX     ;load up the cpu regs
        mov   cx, _CX
        mov   dx, _DX
        mov   si, _SI
        mov   di, _DI

        mov   bp, _BP
        mov   ds, _DS
        pop   bx
```

Continued on page 19

The file ISR.H contains the typedef for the ISRH data structure, the prototypes for the functions in ISR.C, and some support macros (see Figures 4 and 5). ISR.C contains all the support routines. The listings are reasonably straightforward, so I'll try to be brief here.

The ISR functions are:

```
#include <isr.h>

void far *isr_get_vector( inum )
int inum

void isr_set_vector( inum, farfuncp )
void far (*farfuncp)();
int inum;

void isr_install( isr, inum, handler )
isrh *isr;
int inum;
void far (*handler)();

void isr_restore( isr )
isrh *isr;
```

isr_get_vector() returns the vector for interrupt inum from the 8086 interrupt vector table. isr_set_vector() puts the vector farfuncp into the table.

isr_install() copies the "handler template" code into the isrh structure *isr. It then modifies the code in that structure. The far function *handler is installed as the function that will handle the interrupt. The vector for interrupt inum is saved in isr->prev_hndlr.

If that vector is NULL, *isr will do an IRET when *handler returns; otherwise, *isr jumps to *isr->prev_hndlr on return from *handler. The last thing isr_install() does is point the CPU's interrupt vector to interrupt inum at *isr; at which point *handler will be called if an interrupt occurs.

isr_restore() restores isr->prev_hndlr to the interrupt vector table.

The template code does the following:

- Stacks the CPU registers.
- Loads the correct data segment value into the DS and ES registers. (You'll have to check your start-up routine to find out what

values you need in DS and ES.)
- Stacks a far pointer to the ISRH data structure (i.e., this routine.)
- Does a CALL FAR to your C handler function.
- On return, pops the far pointer to the ISRH structure off the stack, discarding it.
- Restores the CPU registers.
- If your C handler used the isr_iret() macro before returning, then an IRET instruction is executed here. If it used the isr_pass() macro, or neither of the two macros, the default procedure at this point is to jump to the previously installed handler; or in the absence of one, to execute an IRET.

The isr_retf() macro can also be used by functions that are called by external programs; it causes the handler to execute a far return. This can be handy, for example, when writing handlers for mouse driver interrupts. The mouse driver will intercept the interrupt, then do a far call to the user's handler.

Using The ISR Functions

Since the CPU registers are pushed on the stack used by the C handler function, you can access anything that's on that stack as a parameter. A handler would therefore be declared like this:

```
void far handler( *isr, bx, es, cx,
    ax, dx, ds, di, si, bp, ip, cs,
    psw )
isrh far *isr;
unsigned int bx, es, cx, ax, dx,
    ds, di, si, bp, ip, cs, psw;
{}
```

Or, if your compiler supports passing structures by value (most do)

```
void far handler( *isr, _REGS )
isrh far *isr;
isr_regs _REGS;
{}
```

Again, CS:IP and PSW were pushed by the CPU prior to calling *isrh, which stacks isr and BX through BP, then calls handler(). On return from handler, *isr pops BP through BX and isr. CS:IP and PSW get popped off when an IRET is executed.

If, for example, you install an INT 0x23 Ctrl-Break exit handler, it can either clear the caller's Carry flag to indicate that you want your program to continue executing, or set the carry flag to indicate that you'd like to crash. An INT 0x23 handler to ignore Ctrl-Break interrupts would be set up like the following:

```
main()
{
 void far int_23();
 isrh isr_23;

 isr_install(&isr_23,0x23,int_23);
 /* your program */
 isr_restore( &isr_23 );
}

void far int_23( isr, _REGS )
isrh far *isr;
isr_regs _REGS;
{
 _CARRY = 0;
 isr_iret(isr);
}
```

As you can imagine, if a program puts other stuff onto the stack before generating a software interrupt, your handler would have access to that, too. An INT 0x24 Critical-Error handler

would have a parameter list like this: (isr, r1, r2). r1 was pushed by DOS before it issued the INT 0x24. r2 was pushed by DOS when it received control from your program. To force DOS to retry the operation, you'd have to set r1.AX to 1, then do an IRET (return after an isr_iret(isr)).

A couple of side notes: See the IBM Disk Operating System Technical Reference, chapter 5, if you need more info on the Critical Error handler (and others).

The isr pointer is passed by the template code to allow the interrupt handler to identify the source of the interrupt. I did that to allow a general handler to intercept and identify illegal interrupts. If you find you need the stack space, or aren't suffering from spurious interrupts, you can get rid of it by deleting the associated code in the template, in isr_install() and in the ISRH structure definition in the ISR.H header. (Spurious interrupts aren't a problem most of the time.)

The Listings

CPU.H contains CPU related constants and the definition of the _ _flags structure. REGSDUMP.C contains the entire _ _regs_dump() routine, which is kept separate to ensure that it's only linked in if it's used (see Figures 6 and 7).

At the end of ISR.C is a test routine that gets compiled if MAIN is #defined. As you can see, it installs handlers for the keyboard, timer and CTRL-BREAK interrupts. Essentially, it displays the last scan code received from the keyboard, and the elapsed time since the program started. It's not very easy to break out of since it traps all attempts from the keyboard, and tells DOS to ignore anything that might come down a COM line if the console is redirected.

Two other files, CALLER.C and CALLER.H, contain improved versions of code originally written as part of a big project two years ago (see Figures 8 and 9). They served as the prototypes for CALL.C and CALL.H. They're based on a routine, caller(), that's faster, smaller and uses less stack space than the new functions. It also saves and restores SS:SP to recover from routines that don't behave properly.

I didn't bother to discuss it here because it's not ROMable or reentrant. For some applications, though, caller() would do just fine, so those two files are included with the others in the file ISR.ARC. Look for it on the Micro C BBS or the Issue #46 disk.

ISRH Notes

The main thing to bear in mind when

Continued from page 17

```
        pop   es

        ret                 ;do the call

retadr:                     ;when the routine does a retf, we get
here...
        push  bp            ;save the current bp
        mov   bp, sp        ;make it possible to get values off the
stack
        mov   bp, 2[bp]     ;get value of BP that we had before the
call

        push  es            ;save the current value of ES
        push  bx            ;   and BX

        les   bx, _RO       ;get pointer to output register structure

        mov   _AX, ax       ;save the current value of AX
        pop   ax            ;get the returned value of BX
        mov   _BX, ax       ;  and save it too
        pop   ax            ;get ES
        mov   _ES, ax       ;save that...
        mov   _CX, cx       ;and this...
        mov   _DX, dx
        mov   _DS, ds
        mov   _SI, si
        mov   _DI, di
        pop   ax            ;then the BP that we last pushed
        mov   _BP, ax
        pop   ax            ;get rid of BP that we pushed before that
        pushf               ;and last but not least, save the flags...
        pop   ax
        mov   _PSW, ax

        pop   es            ;restore C regs
        pop   ds
        pop   di
        pop   si
#endasm
}

/*        INTN_8086()

        This function calls the interrupt handler pointed at by
        element (IP) in the 8086 interrupt vector table.
*/

#pragma vpindex intn_8086
unsigned int far intn_8086( ri, ro )
_ _regs far *ri, *ro;
{
#asm
        push  si            ;save C regs
        push  di
        push  ds
        push  es

        push  bp            ;we're going to need this later...

        les   bx, _RI       ;get pointer to input regs

        pushf               ;simulate an INT nn
        push  cs            ;here's where the routine that we'll call
        mov   ax, offset retadr
        push  ax            ;will return

        mov   ax, _IP       ;interrupt number is hidden in _IP...
        shl   ax, 1
        shl   ax, 1
        cwd
        mov   es, dx
        mov   bx, ax
        push  es:word ptr 2[bx] ;put address to call on the stack
        push  es:word ptr 0[bx]

        les   bx, _RI       ;restore the es:bx pointer to _RI
        jmp   do_it         ;use the code in call_8086...
#endasm
}

/***/
```

Figure 2 — CALL.H

```
/* CALL.h   -  header to allow low level routines
to look like they work directly with CPU registers
and do callf/ints "naturally!" Works with the
functions in CALL.C/ISR.*
*/

#ifndef OP_RETF
#include <cpu.h>
#endif

/* These typedefs describe the types of structures
that are found in the final register structure at
the end.
These aren't intended to be used by humans...
*/

typedef union _ _REG {
   struct {
      unsigned char l, h;
   } b;
   unsigned int  x;
} _ _reg;

typedef union _ _PCREG {
   void far (*x)();
   struct {
      unsigned int l, h;
   } w;
} _ _pcreg;

typedef union _ _REGDP {
   struct {
      _ _reg r;
      unsigned int s;
   } i;
   unsigned char far *p;
   unsigned long l;
} _ _regdp;

typedef union _ _REGPSW {
   _ _flags bit;
   unsigned int w;
} _ _regpsw;

typedef union _ _REG3R {
   struct {
      _ _reg ax, dx;
      unsigned int ds;
   } r1;
   struct {
      _ _reg ax;
      _ _regdp dsdx;
   } r2;
   struct {
      _ _regdp dxax;
      unsigned int ds;
   } r3;
} _ _reg3r;


/* This is the main register file structure. The
interrupt handlers all push and pop registers in this
order, and the functions in CALL.C work with this
"register file" organization. Note that the stack
pointer isn't actually used. That object was included
in the structure for "completeness" and/or "future
expansion," etc.
*/

typedef struct _ _REGS {
  _ _regdp esbx;
  _ _reg c;
  _ _reg3r dsdxax;
```

writing an ISRH is that your function will be using whatever's left of the stack frame that the interrupted code was using (i.e., don't expect to have any stack space available). Better still, don't use automatic storage at all — if possible, declare your local variables as static.

Make sure that your handler isn't being interrupted, then called reentrantly. If that's going to happen, be careful about how you use static locals; they might get wiped out by the reentrant call. You might also have to implement some form of mutual exclusion lock, or modify the template code to use an array of stack frames.

Often, you can avoid these hassles by keeping a handler short and sweet. Let it collect whatever information the main program might need, execute a procedure, or whatever; but keep it simple, and get out *fast*!

Remember that the compiler makes assumptions about the structure of the memory spaces that it works with. In particular, it may assume that your stack frame is in its data segment (especially in the small data models), so it's wise to be careful about pointers while servicing an interrupt. If you've got a bug somewhere and don't see any problem with your source, check the assembly level output from the compiler. Make sure that the proper segment values are used in pointer operations.

Not all ISRHs are going to be tricky, but many will. In most cases, it won't be possible to run them through a debugger, and library functions like printf() won't work from within them. On the other hand, they're useful enough to be worth the extra effort!

♦ ♦ ♦

Continued from page 20

```
  unsigned int di;
  _ _regdp bpsi;
  _ _pcreg pc;
  _ _regpsw psw;
  _ _regdp stk;
} _ _regs;

/* These macros are provided to
make it easier to access
objects in the above structure.
Use either p->AX or s.AX as
appropriate to keep the code
from looking too silly...
*/

#define AX   dsdxax.r1.ax.x
#define AH   dsdxax.r1.ax.b.h
#define AL   dsdxax.r1.ax.b.l

#define BX   esbx.i.r.x
#define BH   esbx.i.r.b.h
#define BL   esbx.i.r.b.l

#define CX   c.x
#define CH   c.b.h
#define CL   c.b.l

#define DX   dsdxax.r1.dx.x
#define DH   dsdxax.r1.dx.b.h
#define DL   dsdxax.r1.dx.b.l

#define DI   di

#define SI   bpsi.i.r.x
#define BP   bpsi.i.s

#define DS   dsdxax.r1.ds
#define ES   esbx.i.s

#define BPSI bpsi.p
#define DXAX dsdxax.r3.dxax.l
#define DSDX dsdxax.r2.dsdx.p
#define ESBX esbx.p

#define PSW  psw.w

#define PC   pc.x
#define CS   pc.w.h
#define IP   pc.w.l

#define STKPTR   stk.p
#define SS   stk.i.s
#define SP   stk.i.r.x

#define OF   psw.bit.of
#define DIR  psw.bit.dir
#define INTE  psw.bit.inte
#define TRAP  psw.bit.trap
#define SIGN  psw.bit.sign
#define ZERO  psw.bit.zero
#define AUXC  psw.bit.auxc
#define PE    psw.bit.pe
#define CARRY  psw.bit.carry


/* The macros here all refer to
something called _REGS, which
must be defined in your module.
It can either be a structure,
or another #defined macro which
refers to a structure, etc.

The idea here is that modules
that contain many DOS or BIOS
calls can put a _ _regs _REGS;
declaration at the start of
every function. As such, _AX
allows access to the AX object
in the structure, etc.

This makes the source code look
MUCH better!
*/
```

```
#define _REGSP ((_ _regs far
                 *)&(_REGS))

#define _AX      (_REGS.AX)
#define _AH      (_REGS.AH)
#define _AL      (_REGS.AL)

#define _BX      (_REGS.BX)
#define _BH      (_REGS.BH)
#define _BL      (_REGS.BL)

#define _CX      (_REGS.CX)
#define _CH      (_REGS.CH)
#define _CL      (_REGS.CL)

#define _DX      (_REGS.DX)
#define _DH      (_REGS.DH)
#define _DL      (_REGS.DL)

#define _SI      (_REGS.SI)
#define _DI      (_REGS.DI)
#define _BP      (_REGS.BP)

#define _DS      (_REGS.DS)
#define _ES      (_REGS.ES)

#define _BPSI    (_REGS.BPSI)
#define _DXAX    (_REGS.DXAX)
#define _DSDX    (_REGS.DSDX)
#define _ESBX    (_REGS.ESBX)

#define _PSW     (_REGS.PSW)

#define _PC      (_REGS.PC)
#define _IP      (_REGS.IP)
#define _CS      (_REGS.CS)

#define _STKPTR  (_REGS.STKPTR)
#define _SS      (_REGS.SS)
#define _SP      (_REGS.SP)

#define _OF      (_REGS.OF)
#define _DIR     (_REGS.DIR)
#define _INTE    (_REGS.INTE)
#define _TRAP    (_REGS.TRAP)
#define _SIGN    (_REGS.SIGN)
#define _ZERO    (_REGS.ZERO)
#define _AUXC    (_REGS.AUXC)
#define _PE      (_REGS.PE)
#define _CARRY   (_REGS.CARRY)



/* The following functions are
in the module CALL.C.
Call_8086() calls the far
function at _PC, while
intn_8086() expects an
interrupt number to be in _IP.
Seg86 simply fills the _ _r
structures' segment registers
with the values found in the
CPU registers at the time of
the call.
*/
```

Listing continued on next page

```
unsigned int far seg86( _ _regs far *ri );
unsigned int far call_8086( _ _regs far *ri, _ _regs far *ro );
unsigned int far intn_8086( _ _regs far *ri, _ _regs far *ro );
void _ _regs_dump(_ _regs far *rp);

/*        These macros do what they seem to do...

NOTE: the cast to (void far (*)()) for _PC in the call86() macro
doesn't work unless x appears before l,h in the typedef for PCREG
(Manx C 86, V. 4.10a.) */

#define call86(pc)   ( _PC = (pc), call_8086( (_REGSP), (_REGSP) ))
#define int86(i)     ( _IP = ( i), intn_8086( (_REGSP), (_REGSP) ))
#define pcdos(f)     ( _AH = ( f), int86( 0x21 ))


#asm
        ;If a C function is declared as being far, and having two
        ;far pointers (_RI and _RO) to _ _regs structures as its
        ;first two parameters, the instruction "les bx, _RI" would
        ;set es:bx to point to the _RI _ _regs structure, and
        ;"les bx, _RO" would set es:bx to point to the _RO _ _regs
        ;structure.
        ;
_RI         equ   dword ptr  6[bp]
_RO         equ   dword ptr 10[bp]

        ;The following are offsets from ES:BX to the contents
        ;of the _ _regs structure.
        ;
_BX         equ   es:word ptr 00[bx]
_BL         equ   es:byte ptr 00[bx]
_BH         equ   es:byte ptr 01[bx]

_ES         equ   es:word ptr 02[bx]

_CX         equ   es:word ptr 04[bx]
_CL         equ   es:byte ptr 04[bx]

_CH         equ   es:byte ptr 05[bx]
_AX         equ   es:word ptr 06[bx]
_AL         equ   es:byte ptr 06[bx]
_AH         equ   es:byte ptr 07[bx]

_DX         equ   es:word ptr 08[bx]
_DL         equ   es:byte ptr 08[bx]
_DH         equ   es:byte ptr 09[bx]

_DS         equ   es:word ptr 10[bx]

_DI         equ   es:word ptr 12[bx]
_SI         equ   es:word ptr 14[bx]
_BP         equ   es:word ptr 16[bx]

_IP         equ   es:word ptr 18[bx]
_CS         equ   es:word ptr 20[bx]

_PSW        equ   es:word ptr 22[bx]

_SP         equ   es:word ptr 24[bx]
_SS         equ   es:word ptr 26[bx]
#endasm

/***/
```

Note: You'll find figures 3 - 9 on pages 74-79.

Turning A PC Into An Embedded Control System:

Phase 1 — Interfacing EEPROMs To The PC Bus

Okay, guys, Bruce is hacking hardware again. He's taken on a project that'll keep him embedded in our hearts and minds for some time. It's an unforgettable design project using EEPROM.

In the coming months, I'll be working (piece by piece) on an ambitious project — turning a PC motherboard into an embedded control system.

In this issue, I'll start by showing you how to use nonvolatile memory (or EEPROM: Electrically-Erasable Programmable Read Only Memory) in your PC.

Embedded Controllers

Embedded controllers are standalone systems generally dedicated to a single task. They usually don't include monitors or disk drives, but sometimes do have a very simple keyboard or display.

You hear a lot about the microcomputers in telephones, toasters, toilets, and televisions. These embedded controllers on a single chip have been optimized for cost. Single-chip microcontrollers are ideal for situations where you make thousands of products, because the design costs can be spread out. These costs are (of course) significant.

While I was at Fluke (in Seattle), I programmed a 4-bit microcontroller designed by NEC to drive a vacuum-fluorescent display directly (like the one on your VCR). This involved learning the hardware characteristics of the chip and a *very* arcane assembly language. We had to develop and debug the assembler while writing the code (with little in the way of examples or English documentation).

All this work was expensive. However, since it was going into a meter with projected sales of 15,000 to 25,000 per year, the costs were insignificant compared to the price of using, for instance, a Z80 with the necessary peripheral support chips.

Many companies can't justify the cost of building everything from scratch, and there's a rapidly growing industry which supplies various types of PC-environment embedded systems. These include normal PC hardware with a special BIOS (which doesn't look for disk drives, etc.) and (often) a ROMed version of DOS.

Systems like this allow you to develop your embedded systems using the powerful compilers and tools available on the PC (you can, for instance, use C or C++).

Commercial PC-environment embedded systems tend to be expensive; not as expensive as most other alternatives, but often as expensive as an XT with a hard disk. The prices drop dramatically if you get into volume.

My favorite example is Intel's "Wildcard 88," which has the core of an XT motherboard on a card 2" x 4". Unfortunately, there isn't any RAM (an Intel spokesman said, "we don't make RAM, so why should we put any on the board?"), but these boards are only $50 apiece, in lots of 1000!

Since XT motherboards are under $100 (and you don't need the usual hefty power supply since there won't be disk drives), why can't we build our own embedded controller (in quantities of *one*) with the power and development environment of a PC for under $200?

We can, if we can negotiate the tricky parts —

(1) Figuring out how to modify (or write from scratch) the ROM BIOS.

(2) Using RAM — probably static RAM, since it uses less power and doesn't need refreshing.

(3) Establishing nonvolatile memory, since we don't have disk drives (the subject of this article).

(4) Writing C or C++ embedded applications and burning them into EPROM.

This is a big tricky ticket, and it's going to take us several issues before we have all the nooks and crannies explored and worked out. But that's where we're headed.

EEPROM

EEPROM allows you to store information not lost when you power down. It's like an EPROM which doesn't require an EPROM burner — it gets "burned" right in the circuit. EEPROMs take much longer to write than they do to read. The read cycle is on the same order as normal memory.

You can only write to a memory location of an EEPROM a limited number of times (for the chip I'll talk about here, it's 10,000). However, there's no restriction on the number of times you can read it. Normally you erase an EEPROM by writing over old data, but new crops of "flash erasable" chips are showing up.

Embedded systems often use EEPROMs as long-term storage for state information (since there are no disks) or for storage of information which can't rely on the presence of a disk.

As EEPROMs get cheaper, it becomes feasible to begin using them to store large amounts of data. Several companies make cards containing EPROM (with ROMed DOS) and large amounts of EEPROM.

These cards look like bootable DOS disks. You simply load your program onto the EEPROM and install it in your target system as drive A, so the computer boots from the card. This looks like it might be an alternative way to create an embedded system from a PC. However, the price of these cards has always been too high for me.

The Chip

I selected the chip for this project by going to the back pages of *BYTE* and looking for EEPROMs. I always try to use easy to find parts, but sometimes people still have trouble, so I felt this was the safest approach.

By Bruce Eckel
Eisys Consulting
501 N. 36th Street, Suite 163
Seattle, WA 98103

This part is a Samsung KM2816A-25 (compatible with Xicor X2816A) 2K by 8 bit (i.e., 2K bytes) TTL-compatible 5-volt-only EEPROM. The "25" in the part designation refers to the 250 nanosecond read time. It's available through Jameco for $6.25, but they do have a minimum order of $20. I suggest you get the data sheet ($0.50 extra).

Jameco Electronics
1355 Shoreway Rd.
Belmont, CA 94002
(415) 592-8121

If you need a denser part, the 2865A-30 is 8K bytes for only $9.95; it should be easy to modify my design for that part.

Many EEPROMs require special hardware support for the extra-long write cycle. Or they may have other interfacing requirements (higher write voltages, for example).

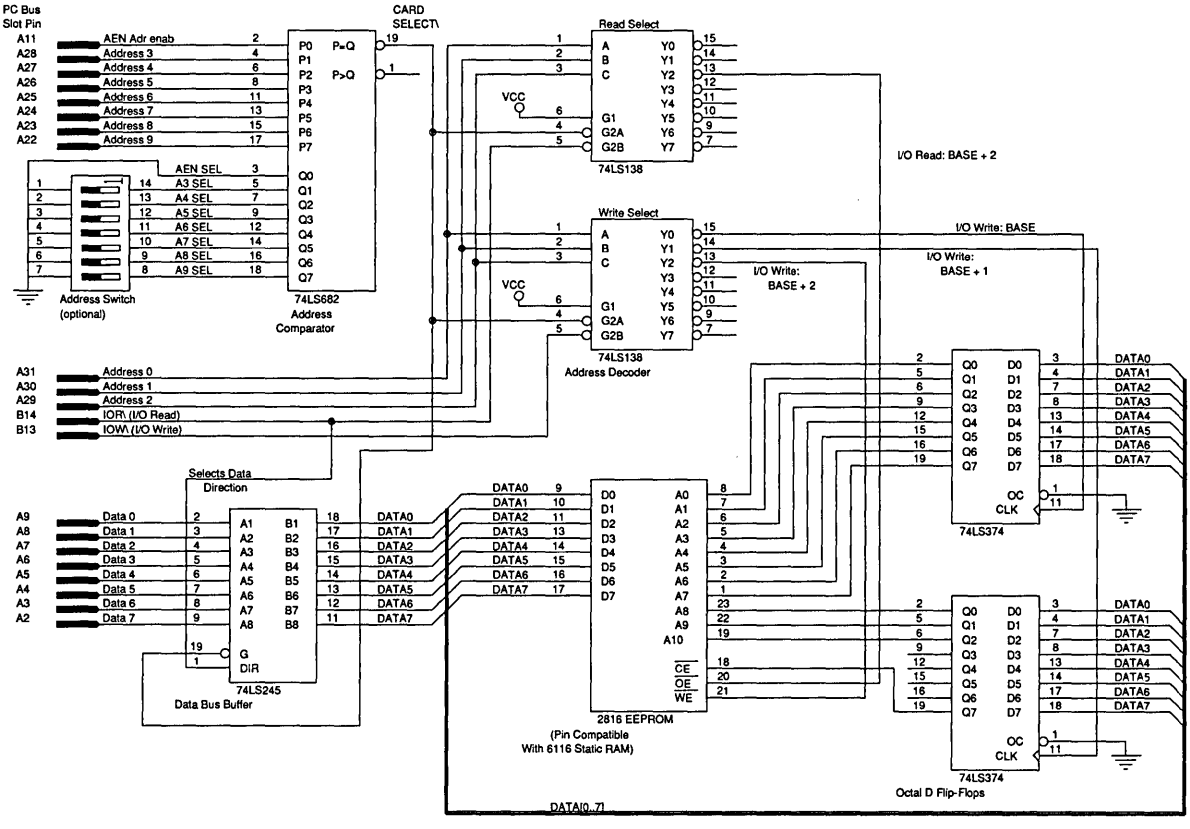
The 2816A chip was designed to interface directly to a microprocessor bus. It only requires 5V for reading and writing, and has internal hardware to latch the write value.

You use the same kind of bus cycle to write to the EEPROM as you do to a RAM. (The pinout is the same as a common 6116 static RAM; you can just drop it into the same socket if the bus timing is within specs.) The only caveat is you must wait 10 milliseconds between writes.

In an embedded application with hardware dedicated to a specific task, it would be appropriate to take over some of the PC's memory space for EEPROM, some for static RAM (lower power, fast — no wait states, and no refreshing necessary), and some for the program EPROM.

On a general-purpose PC, however,

Figure 1 — PC Bus To EEPROM Interface



we can't dedicate any of the memory space to EEPROM so I've designed it to use the I/O space. Even then, there isn't enough free I/O space to support 2K bytes (only 10 lines, or 1 K, of I/O is brought out to the PC bus, with much of that dedicated to other purposes).

To test the EEPROM, I've designed a card which occupies 8 I/O addresses. (For an introduction to designing PC adapter cards, see my article in *Micro C* #43, Sept./Oct. 1988, or chapter 11 of my book, *Computer Interfacing with Pascal & C.*)

The card only uses three of the addresses. You write address and control line information to BASE and BASE+1. (These are write-only.) You write data to and read data from BASE+2.

Once again, I've taken a chip designed to interface directly to a bus and made it independent of the bus through a combination of hardware and software (see *Micro C* #42, July/Aug. 1988, or chapter 10 of my book). Maybe it's a step backward, but I find it a very useful technique for testing hardware.

Figure 1 shows the complete circuit for the chip. Most of the circuit is simply the PC adapter card interface described in issue #43. Two latches were added to hold the address and control lines to the EEPROM; the rest is done in software.

Figure 2 is the assignment of the bits in BASE & BASE+1 (BASE+2 is simply the data bus, so I didn't show it here). Notice I've combined address and control functions in the two bytes. I could put the control lines in a separate byte, but that would take more hardware. If you modify this design to add more EEPROMs, however, you might want to separate the control lines.

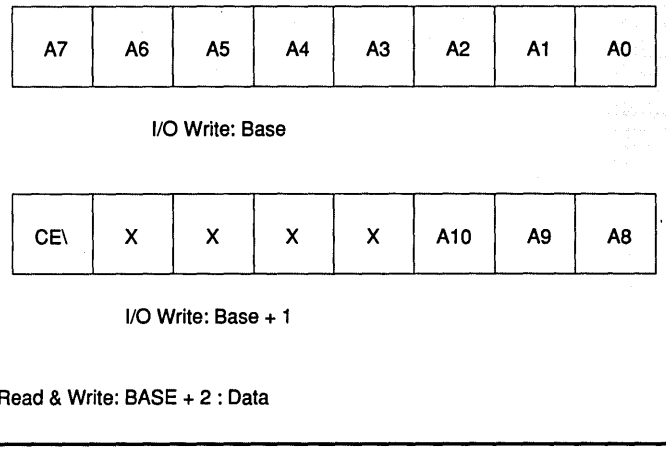
Figure 3 shows the two functions —

```
EEPROM_write()
EEPROM_read()
```

and a very simple program which uses these functions.

If you program in another language, you should be able to find similar functions to substitute for inport() and outport().

Figure 2 — 2816 EEPROM Control Bits



Cycling The Lines

Reading the chip is a two-step process. First, you write the address to BASE, latching it onto the 2816's pins, and force CE (chip enable) and OE (output enable) low by writing to BASE+1. (These lines are active low.)

Then you can read data from BASE+2.

You can save power by raising the CE line (again a write to BASE+1) following the read. While unnecessary in our system, this happens automatically if the chip is interfaced to a bus.

For a write cycle, you send the address (BASE) and data (BASE+2) to the chip. Then you enable the chip by dropping CE. Here, OE must be high (since we're WRITING to the chip, outputs are not enabled). Finally, you lower and raise the WE line to generate a write pulse.

After the write, you raise the CE line to disable the chip and then twiddle your thumbs for 10 msec to make sure another write doesn't happen too quickly. (This isn't necessary if you have some other insurance that your program won't write a byte more often than once every 10 msec.) To write large quantities of data while performing other tasks, you might want to use an interrupt routine tied to a clock tick.

In a system where the EEPROM ties directly to the microprocessor bus, the microprocessor and the decoding logic generate signals to the chip, so they look other than they do here. In that situation, you must examine the timing on your bus and compare it to the spec sheets for the 2816A.

Circuit Notes

You'll notice I left off the power and ground pins on the chips. I drew the diagram with OrCAD (what a *wonderful* program!) and the parts came from the library this way, which is very common on circuit diagrams. Fortunately, there's a pattern. If you look down at the top of the chip with pin one in the upper left corner, you'll see the lower left pin as ground and the upper right pin as power.

Another "hidden feature": every two or three TTL chips, you should put a capacitor with a value from 0.01 to 0.1 uF between power and ground to keep TTL noise from propagating through the supply lines (called "bypass capacitors").

Editor's note: Bypasses should be low-impedance ceramic or plastic capacitors and you should keep their leads as short as possible. In fact, it doesn't hurt to have one tied between +5V and ground at every IC. Also add three or four 10 to 50 uF tantalum capacitors between the 5V line (and any other supply lines) and ground.

Tantalums are polarized (like common electrolytics) so be sure someone shows you which lead is plus. Also, tantalums have a voltage rating. Those rated at 20V or greater will work fine for 5V and 12V supplies.

The DIP switch controls address selection. I've shown it as "optional" since you can simply wire the lines to ground or leave them open. To select an address for BASE of 0x220 (generally unused), ground all lines except A5 SEL and A9 SEL. The 74LS682 has internal pullup resistors so any lines left open will be forced to "1."

The backslashes ("\") after signal names denote "not" or "active low." OrCAD won't let me draw bars above signal labels (which would have been prettier). At least, I haven't figured out how to do it.

For quick bread-boarding, you can get a superstrip card that plugs into the PC bus from Global Specialties. The part number is PB88 or PB88-2. They also have a new proto-development card with all the decoding and buffering done for you (PCL-750). Call them for a catalog.

Global Specialties
P.O. Box 1405
Newhaven, CT 06505
(800) 345-6251

Figure 3 — 2816 EEPROM Reads And Writes

```
/* Turbo C program to read and write data on the 2816 EEPROM
by Bruce Eckel, EISYS Consulting */

#define BASE 0x220  /* address set on DIP switch */

void EEPROM_write(int address, unsigned char value) {
  /* See figures 1 and 2 to understand this function */
  outportb(BASE, address & 0xff);  /* low 8 bits */
  /* High 3 bits and CE forced low */
  outportb(BASE+1, (address >> 8) & 0x07);
  /* Write the data: */
  outportb(BASE + 2, value & 0xff);
  /* 'delay()' is a function from Turbo C 1.5
  to wait for a number of milliseconds.  If you
  have version 1.0, write a simple "for" loop. */
  delay(10); /* Wait 10 mS for EEPROM to finish writing */
}

int EEPROM_read(int address) {
  /* Place the address at the 2816 pins and force CE low: */
  outportb(BASE, address & 0xff);
  outportb(BASE+1, (address >> 8) & 0x07);
  /* Read the data: */
  return inportb(BASE + 2);
}

main() {
  int i, address, value;
  /* First, dump the contents of the EEPROM: */
  for (i = 0; i < 2048; i++) {
    /* Generate breaks & line numbers: */
    if ( i % 20 == 0) printf("\n%d: ",i);
    printf("%d ",EEPROM_read(i-1));
  }
  while(1) {
    printf("\nRead, Write, or Quit?\n");
    switch(getch()) {
      case 'R' : case 'r' :
        printf("address to read? ");
        scanf("%d",&address);
        printf("%d\n", EEPROM_read(address));
        break;
      case 'W' : case 'w' :
        printf("address to write? ");
        scanf("%d",&address);
        printf("\n");
        printf("value to write? ");
        scanf("%d", &value);
        printf("\n");
        EEPROM_write(address,value);
        break;
      case 'Q' : case 'q' :
        exit(0);
      default:
    }
  }
}
***
```

Next Time

I may get distracted from embedded systems sooner than I expected. I was cruising Radio Shack the other day and saw a pair of nifty LSI chips which, combined with some hardware, create speech.

I know speech projects are popular, and it seems like a PC adapter card to generate speech could be a lot of fun. Embedded systems, C++, speech synthesis, PCB cards ... there are just *too many* fascinating projects waiting to be created!

Editor's note: Bruce often refers to articles in previous issues. You can get a book of his hardware articles, Computer Interfacing with Pascal & C, *and a disk of source code. Send a check for $30 to Micro Cornucopia, P.O. Box 223, Bend, OR 97709.*

If you want to learn more about C++ programming, you can get disk #1 of the C++ Source Code Library, "Windows, Matrices & MicroCAD," by sending a check for $15 to the Bruce Eckel, Eisys Consulting, 501 N. 36th St., Ste. 163, Seattle, WA 98103.

◆ ◆ ◆

Bringing Up A Surplus 68000 Board

A New Father Describes The Experience

Last issue Karl talked about purchasing boards on the surplus market. This issue he describes the process of bringing up a surplus system. It's a fascinating process, one you'll miss if you limit your building to plugging together clone boards.

I wanted a fast 68000 system with lots of RAM, capable of running the SK*DOS operating system, and I wanted it for less than $200. This article tells how I did it.

I built my system around a surplus Convergent Technologies Mini-Frame 68010 board running at 10 MHz. The board boasts 512K of RAM, a floppy disk controller, a hard disk controller, two serial ports, a parallel printer port, and a 16-channel HDLC cluster controller. I added a Panasonic JU-455 floppy drive, high-current power supply and a Tele-Video 910 terminal. Total cost of the components (all surplus) — about $180.

Getting a surplus floppy drive to work seldom presents a problem; bringing up a full mini-computer without user's manuals, technical docs, or even a schematic is another story. When I first brought my $40 board home from the surplus shop, all I had for information was what was silk-screened on the board.

There Are Easier Systems

If you want to try your hand at bringing up a surplus 68000 system, you can make your job easier by starting with a better documented board. Schematics and documentation for something like the Motorola M68KVM02 VERSAmodule family are easy to get. The boards turn up often in the surplus markets.

Or, you might stumble onto a used Amiga or Atari 68000 system, in which case there should be no problem with documentation. Of course, you can also buy these systems new, or go with the Peripheral Technology PT-68K. The latter is a good system that can be purchased for as little as $200 in kit form; drop PT a line for more info.

Checking Out The Hardware ...

A quick look at the Mini-Frame revealed enough information to get started. The board has a single large LED near one edge, a row of five smaller LEDs (apparently a status display) down another edge, two socketed 2764 EPROMs and some 1488/1489 RS-232 drivers for the serial ports. Since the 1488s use +12V, +5V and -12V, I knew my board would need (at least) these voltages. I also found a large reset button below the serial connectors.

After perusing the board, I figured it was time to see if it worked. Using a VOM (and a pin-out of the 1488s), I located the pins on the nine-pin power plug for +5, +12 and -12 volts. When I connected my $30 surplus power supply and hit the board's reset button, the large LED lit red and stayed on. Not encouraging.

Looking over the board more carefully, I noticed that one of the two system EPROMs had been plugged in backwards. I reversed the chip and tried power again. This time I got ten seconds worth of dancing LEDs. After the LEDs stopped flashing, the large LED went out and only a single green status LED remained lit.

Encouraged, I connected a floppy disk drive, inserted a floppy disk and powered up the board. The drive came on and the heads stepped across the disk as the board attempted to boot the alien disk. I had a working board!

Writing A New Monitor

But this board wanted to boot and run UNIX — I wanted to rewrite the code in the EPROMs to do something more interesting. This meant figuring out where all the peripheral chips, EPROMs, and RAM resided in the CPU's 16 MB address space.

A partial disassembly of the EPROMs, provided by a friend who had also purchased a Mini-Frame board, revealed that the EPROM was addressed to start at $800000.

Since the 68010 typically uses address zero as the starting point for its exception table, it was a safe bet that the RAM ran from $0 to $7FFFF. (It turned out that wasn't the case, but I'll get to that in a bit.) But where were the peripheral chips? I had to find the address of the Intel 8274 serial chip before I could talk to my board.

There was only one way into the machine — through the ROMs where the boot code resides. I would have to burn a test routine into a pair of EPROMs, install them in the board and run the program.

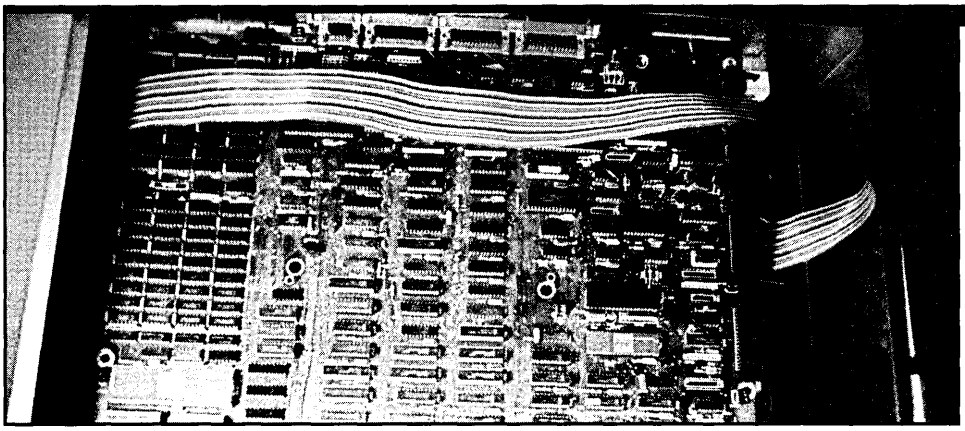
My routine would scan through memory, reading each address. When it hit the address containing the serial chip, the chip's select line would go low. Once I found the block which contained the port, I tested smaller and smaller areas within that block to locate the address.

The Host, The Target, And The Tools

Obviously, the Mini-Frame wasn't ready to be the development system, so I needed a host and some tools. I used a PC-clone running the X68K cross-assembler from 2500 A.D. Software. To burn the EPROMs, I spent weekends with my company's DATA-I/O Model 19 PROM programmer.

I knew I was going to be reprogramming and reinserting the Mini-Frame's ROMs many times, so I plugged machined-pin AUGAT sockets into the board's original ROM sockets. That way I could use the upper, removable sockets as much as I wanted, without damaging the lower, soldered-in sockets.

For hardware tools, I had only a simple LED logic probe. When touched to a circuit point, the probe would show a logic one, a logic zero, or a pulse. I used



By Karl Lunt
7349 W. Canterbury
Peoria, AZ 85345
(602) 869-6146 (days)

the logic probe mainly to track down the addresses of the different peripheral chips, by looking for a pulse on the chip select line.

Most of all, a large library of technical data made this project possible. I obtained data books on all the major chips in the Mini-Frame, knowing I would need the information at least twice. I used the pinout data as I hunted for the chip's address in the memory map. I later used the programming information when I wrote new code for my own monitor and programs.

Finding The Serial Port

Now began the hunt for the address of the 8274 serial port chip. What followed was a long, tedious cycle of coding, programming the EPROMs, running the code, noting the results and trying again. The program repeatedly read every address in a selected range, then started over again. As the program ran (if it ran), I would hold a logic probe on the serial port's chip select, looking for a select strobe.

The board crashed many times, leaving just the large LED on. This meant that I had read a forbidden address. Unfortunately, there seemed to be several such areas of memory.

Finally, I tracked down the serial port at address $C30000. Armed with the proper Intel manual, I wrote a routine which sent a stream of asterisks to the terminal. Then I installed the new code in the EPROMs and fired up the board.

Unfortunately the big LED stayed on, telling me the code had crashed. I rewrote that code a couple of times, to be sure I got it right, but the result was always the same — the board crashed.

The code was correct; there had to be a problem elsewhere. Since I used subroutines to do the initialization of the serial port, the RAM had to be working for the code to work. Perhaps the RAM wasn't there after all.

Activating The RAM

Figuring I had overlooked something, I went back to the disassembly of the original EPROMs. The code started out by writing a pattern of words to particular addresses. After I added this code to the beginning of my routine, I got my stream of asterisks on the terminal. Apparently, writing just those words to just those addresses activated RAM.

Later, I found that system RAM could be moved about in the CPU's logical address space by changing the data written into the mapping RAMs. This was verified later still when a friend acquired a set of schematics for the Mini-Frame. For now, it was enough to know how I could turn on the RAM.

The Beginnings Of A Monitor

By now, I had learned enough about the Mini-Frame to begin writing a small, EPROM-based monitor. I built some simple diagnostics into the first section of the monitor's code, using the five status LEDs. As the monitor went through its startup sequence, it would light different LEDs. A failure of any test crashed the system, leaving a distinctive status pattern. If all the tests passed, only a single green LED remained lit.

I included the system testing because several friends had also purchased surplus Mini-Frame boards and the tests would help them get their boards running.

Finally, before the monitor prompted for a command, it would activate all 512 KB of RAM and test it, displaying on the terminal the results of the test. The first time that baby came up and announced that all 512 KB tested good, I felt great.

Dumping, Scanning, And The RAM Loader

One of the first commands I installed in the monitor was a simple ASCII dump routine. This allowed me to quickly explore areas of the address space, looking for goodies. Considering the 64 KB memory of my 6809 systems, it felt strange to look at addresses above $10000; I wasn't used to numbers that large!

I tried using the dump command to locate the addresses of the other main chips, such as the WD2797 floppy disk controller (FDC). That got boring quickly; it was too much work to type in a dump command, hold the logic probe in place, and press RETURN just to read 256 bytes. I needed a way to check a lot of addresses quickly and repeatedly. So, I added the sweep command to the monitor.

The sweep command would read whatever size block I wanted, repeating the task until I interrupted it. Using this tool, I quickly found the locations of the FDC, the WD1010 hard disk controller (HDC), and the printer port. I also stumbled across at least one address that appeared to force a hard reset of the system. Earlier, this had caused me so much grief when I had to burn my searches into EPROM.

Which brings up a point — each time I changed the monitor or tested a program, I still had to burn a new set of EPROMs. This had two major drawbacks. First, it takes time to erase, program, test and install EPROMs. Second, since the EPROM programmer belongs to my company, I only had access to it on the weekends, limiting the time I could hack on my machine. I needed a way to download 68000 code directly from the host system.

Since the linker on the host system could output Motorola S1-S9 object records, I wrote a simple S1-S9 loader and burned it into the monitor EPROMs. This RAM loader command removed the last barrier to rapid progress on the Mini-Frame. I could now develop programs on the weeknights, then use the EPROM programmer on the weekends to upgrade the monitor.

Tracking Down The Disk DMA

I had made a lot of progress so far. I had full control of the serial ports, access to 512 KB of RAM, and a monitor strong enough to develop simple programs. But the biggest challenge remained. How did the floppy disk system work, and could I get SK*DOS running?

It was about this time that my friend acquired the set of schematics. The diagrams showed the DMA section, and my sweep routine located the DMA controller. Using the disassembly of the original EPROM code, I put together a simple set of routines to read data from a PC-formatted floppy disk.

With a little bit more work, I also wrote data to the disks. After all this, I figured it wouldn't be too difficult to generate code which would read and write SK*DOS disks.

Cleaning Up Little Things

What remained? I wanted to change the serial baud rate at will. A little research with the schematics and the sweep routine showed me how to use software to control the baud rate generator. The printer port needed fixing, as it was *not* PC-compatible. Using the schematics, I made a small adapter with a pair of DB-25 connectors — my line printer now works just fine on the Mini-Frame.

I still have not tackled the hard drive controller, mainly because I don't have a spare hard drive. That will make a good rainy day project. I also want to add some battery-backed static RAM to the machine. This will let me put SK*DOS' RAM disk in nonvolatile RAM. (Yes, I do have SK*DOS running on the Mini-Frame and it's a terrific DOS!)

I also intend to add an adapter so I can run PC-compatible cards on the Mini-Frame. The address and data busses come out on a 100-pin connector mounted on the board. I just need to find the matching connector and wire up the interface. The idea of EGA graphics (or better) on my 68010 system gives me tingles.

What Have I Gained?

I now own a fast 68010 system, running an excellent DOS. I learned a tremendous amount about 68010 hardware and system design from exploring this machine. I certainly saved a lot of money, but mostly I had a heck of a lot of fun doing a difficult hack on a terrific piece of hardware.

And Thanks To ...

Any project this difficult needs more than just one person on it; several people contributed time, talent and support to get the Mini-Frame to this point. Mark Loomis provided the disassembly of the original EPROMs and insights into the 68000 CPU. Mike Sutherlin scrounged up a great set of schematics. Bruce Beyerler offered encouragement and technical support.

Particular thanks are due Brian Bierer for getting me started. He found the Mini-Frame boards at the local surplus shop. He also added hardware expertise and enthusiasm to the project. A special "thank you" to Bill Lieske, Jr., for the support, encouragement and patience he has shown for the group of hackers who congregate each month at EMR Co.'s metal shop in north Phoenix.

About The Vendors

PT has been a leader in 6800/6809 technology for years. People whose judgement I trust give their PT68K 68000 single-board computer a very high rating for design and performance. Considering the basic kit only costs $200, it would be tough to find a cheaper way to enter the 68000 world.

Peripheral Technology
480 Terrell Mill Rd., Suite 870
Marietta, GA 30067
(404) 984-0742

Peter Stark of Star-K Software has developed first-class software for Motorola chips clear back to the 6800. He's turned out some superb programs over the years, but SK*DOS for the 68000 is something. Lately, he has worked with *Radio-Electronics* magazine (in the Computer Digest section) on a series of articles dealing with the PT68K board and SK*DOS. Check out back issues of the mag for solid information on both the 68000 and the PT68K.

Star-K Software Systems Corporation
Box 209
Mt. Kisco, NY 10549
(914) 241-0287

2500 AD Software wrote the 68000 cross assembler for MS-DOS. It sells for $299.50. They're also working on a C compiler that runs on a PC but compiles ROMable 68000 code. It should be available around February 1. Price should be around $600.

2500 A.D. Software, Inc.
P.O. Box 480
Buena Vista, CO 81211
(800) 843-8144
$299.50 MS-DOS

Editor's note: Karl had to search through memory for I/O addresses because Motorola maps ports and RAM in the same space. Intel, on the other hand, has separate I/O read and write lines so port addresses can overlay RAM addresses without conflict. Both systems have their proponents, but what's important is that the search for ports differs with different processors.

♦ ♦ ♦

Practical Fractals

And Other Flights Of Fractal Fancy

Larry obviously didn't understand the true import of last issue's Culture Corner (he thought I was encouraging his fracticilious schemes). And now, of all things, he thinks he's found a practical application for these twisted notions. (We're trying to humor him, really we are.)

Kicked back in my chair, feet up on the desk, I watch yet another Mandelbrot plot crawl across the screen. A sarcastic voice behind me says, "Fractals? *Again?* When are you going to do something useful around here?"

An annoying and utterly unfair remark. I do plenty of useful work at Micro C. In summer I mow the lawn (well, once anyway), and in winter I shovel snow from the walk. What more do they want?

As it turns out, plenty of folks are working on practical applications of fractals. But, before we get into that, let's go over some fractal theory from a less mathematical perspective — one that will make clear the potential for use of fractals in the real world.

Empirical Theory

Enough obscure esoterica has been written on fractal theory to supply the Micro C outhouse with paper products for years to come. (Indoor plumbing? In Central Oregon? Be serious.) Let's take the concept of fractal dimension and come up with a gut-level definition.

Many objects and processes in nature exhibit "self-similarity" at different scales. That is, they look the same under varying magnifications. England's coastline is a classic example.

From space we see an irregular line made up of bays, peninsulas, and amusement parks. A walk along the beach shows similar detail, but on a much smaller scale. And the view through a magnifying glass includes miniature baylettes, peninsulettes, and parklettes — again looking similar to the view from space.

This "scale invariance" of the coastline's appearance is a major characteristic of fractals. It makes the length of the coastline hard to pin down. A measurement taken from space misses much of the small detail and therefore will be smaller than one taken by a man with a yardstick. But we *can* measure the rate of increase of the coastline length as we zoom in.

It seems reasonable that more complex coastlines, like England's, would show a greater increase in length than, say, Southern California with its long, straight beaches. So this rate of increase directly measures the complexity of the coastline. It's also related to the coastline's fractal dimension.

In more specific terms, determination of the fractal dimension of England's coastline goes like this: Measure its length at various scales and draw a graph of the log of the length vs. the log of the unit measure. "Unit measure" means the smallest measurable unit at a given resolution — large scales have large unit measures.

Obviously, the slope of the plotted line will be negative since, as the scale increases, the length decreases. The fractal dimension then equals one minus the slope of the plotted line — a value greater than one since the slope is negative. It turns out that England's coastline has a fractal dimension of about 1.2.

We can test this notion with a geometrically straight line segment. Since it has no irregularity, measurements of its length will give the same value at all scales. Plotting length and unit measure as before shows a line of slope zero — *no* change in measured length. And (1-0) gives a dimension of one, like all good Euclidean lines should have.

Despite my good intentions, a bit of unexplained math has crept into the discussion. The faint of heart may want to skip to the next section, but for those with a penchant for completeness, the general case follows:

$$P(E) \propto e^{d-D}$$

where P(E) represents the value measured (here, the length of the coastline), e is the unit measure, d the topological dimension, and D the fractal, or Hausdorf-Besicovitch dimension. Think of d in the more traditional way — a dimension of integer value. (Coastlines have a topological dimension of one, and for planetary surfaces, d=2.)

By the way, one definition of fractals says that a fractal has D>d. Again, this makes sense: D=d gives us straight lines and flat planes — definitely not fractals.

Fractals In Oregon?

One of the nice things about living in Bend is having a wild river running through town. At least, the mink, otter, and beaver seem to like it. (Our own governor once refused to visit Bend — "The middle of nowhere" as he called it.) What better place to try my hand at fractalizing than on a nice crooked river.

No, I didn't actually pace the banks of the Deschutes for a small scale measure: I let my dividers do the walking on a map. An afternoon of diligent dividing in the Central Oregon Community College map room generated the data I needed. It's hard to come up with more than one order of magnitude in scale variation this way, but it does serve as an example of the technique. (See Figure 1.)

By my approximation, the wanderings of the Deschutes earned it a fractal dimension of 1.25 — not an unreasonable value. I'm sure the otters (and Governor Goldschmidt) will be glad to know.

Finding The Fractal Dimension

(I know — sounds like an episode of Flash Gordon: Flash Explores the Fractal Dimension.)

By Larry Fogg
Micro C Staff

Given an object in the "3-D" real world, The Real World for example, we can look at its diameter at different scales. Or we could consider the same scaling effect on its surface area. Again, the difference between mountains on the grand scale and molehills on a lesser scale gives us the information necessary to find the fractal dimension of the Earth's surface.

Gathering this data poses a difficult task, but researchers have developed methods for approximating the fractal dimension of a surface.

One possibility involves taking a horizontal slice of the surface (a contour line), finding its fractal dimension, and adding one. A fairly coarse approximation, and one that may depend on the elevation of the contour.

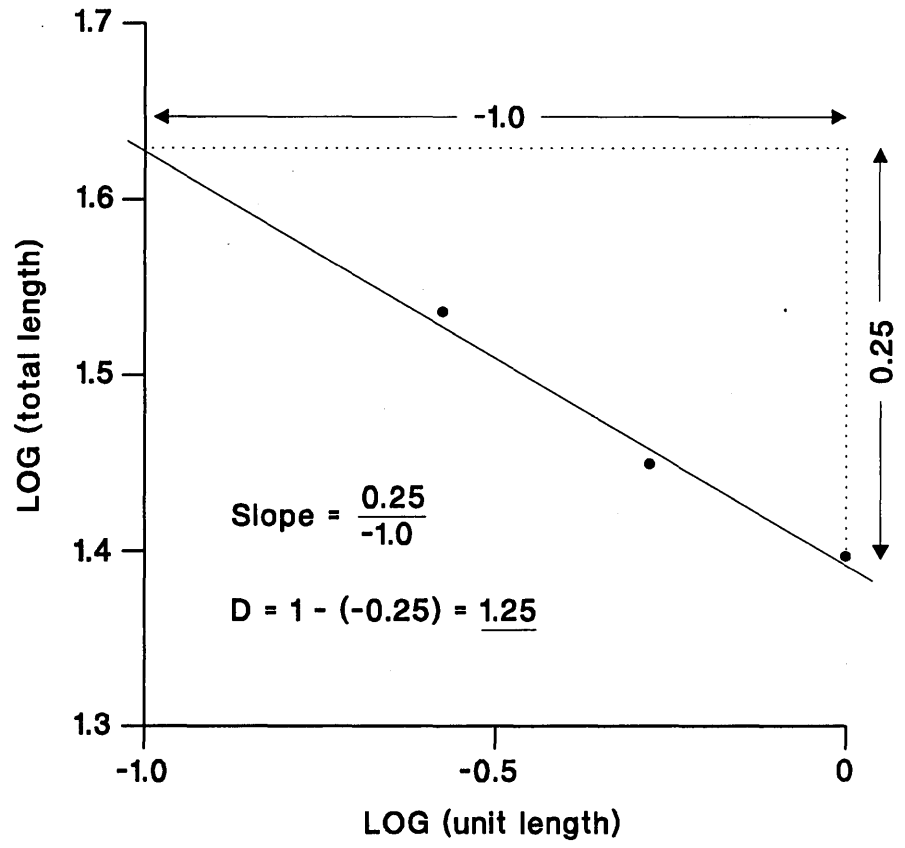
Or you could look at individual points on the surface and investigate their elevation difference vs. the distance between them. Or you can choose from other computationally intensive methods.

K.C. Clarke came up with a simpler method[1]. Overlay the surface with the largest square that will fit. Take the average of the elevations of the four corners and assign it to the center of the square. Now find the total area of the four surfaces of the resulting pyramid.

This gives a first large scale measure analogous to measuring a coastline from space. Next, overlay four smaller squares on the same surface and measure their areas in the same way. The smaller scale will yield a larger value for the total surface area since it will catch more of the surface's details.

Continue this process to the lowest limit of scaling (two pixels by two pixels

Figure 1 — Finding The Fractal Dimension (D) Of The Deschutes River



for a digitized image). Then plot the log of total area vs. the log of the area of an overlaid square, for each size of unit square. Very similar to the coastline method described above, Clarke's method saves a *bunch* of computer time over other fractal surface dimension techniques.

A Practical Example

Professor Philip J. Hopke (of the University of Illinois at Urbana-Champaign Environmental Research Laboratory) and his associates have used Clarke's method in an effort to characterize particles by their fractal surface dimension[2]. They hope to use the fractal dimension as a tool to identify the processes that form airborne particles.

Hopke captures the particle's image from a scanning electron microscope (SEM). The SEM bombards the surface of the particle with a tightly focused beam of electrons. The surface, in turn, reemits secondary electrons whose energy depends on the topology and composition of the surface. This digitized secondary electron image provides the data necessary to find the fractal dimension.

Particles of sodium chloride, sodium sulfate, ammonium sulfate, and one unknown compound have been subjected to this technique. Like most of the research I've seen, this application seems to work well over a somewhat limited range of scales. Hopke reports very consistent results at lower magnifications.

However, high magnification (>1000X) yields lower values for the fractal dimension. Nonetheless, a judicious choice of magnification may yield useful information.

To quote Professor Hopke, "If we can work out the remaining kinks in our methods, fractals could prove to be a valuable tool in investigating atmospheric particles."

Other Applications

Fractals have shown promise in a wide variety of fields, often in finding a quantitative measure of the irregularity of some property, as in Professor Hopke's research. Other areas are:

- Characterization of the roughness of the ocean floor[3].
- Distribution of earthquakes along fault systems[4].
- Optical diffraction analysis of particle clusters formed by diffusion limited aggregation (DLA)[5].

Fractal modeling of many processes may also be possible. Candidates for this application include:

- Clustering of stars[6].
- Spreading of diseases[7].
- AC behavior[8].
- Growth of DLA clusters[9].
- Simulation of lightning[10].

Data compression, linguistics, turbulent flow, population growth, and most anything else you can imagine have been explored with fractal techniques. The list is truly endless.

.COMplexity

I wanted to do something keen like a fractal measure of program complexity. But this points out a danger: Not all objects and processes are fractal in nature. If experimentation shows scale invariance of some property, then by all means explore its fractal possibilities.

I'll keep thinking about it, but it doesn't look like executable programs are fractals. (Except, of course, fractal programs.)

The Bottom Line

A few years back, John Maddox wrote, "...this is an interesting case, which seems to happen more and more often, of how an explanation in search of a phenomenon may lay claim to far more territory than it can handle[11]."

This still holds true for the most part. But while fractals remain the darling of a large portion of the scientific community, a healthy case of skepticism has set in. A myriad of applications tease researchers with promising results, but no one seems willing to claim absolute success in modeling, characterization, or any other application of fractals (except graphics).

Still, it won't surprise me in the least to see fractals become an increasingly important tool in the future.

Commercial Fractal Software

This here's a computer rag so let's finish up with a look at some software available for interdimensional explorations. I know of only two commercial fractal packages for the PC: Mandelbrot Explorer (ME), and FractalMagic (FM). Both do a fine job of tying up your computer at a bargain basement price.

According to the Micro C Fractal Accessory Review Team (Allan Chambers, Dave (that Thompson guy), and myself), ME and FM do just about anything you'd want with Mandelbrot and Julia sets.

Both packages allow storing work in progress for resumption at a later time. (Handy if you occasionally use your system for such mundane tasks as word processing.) They also let you define parameters for new plots by moving a window around an existing plot, or by entering parameters directly.

While both programs let you "animate," or dynamically change the color map of an image, FM gives you more control over the process. I could watch an animated image for hours.

Finally, both support math coprocessors — a definite must for serious fractal work.

On to the differences. FM has mouse support, ME doesn't. I don't consider this a deciding factor — the cursor keys work just fine.

If you have a CGA system, get FM since ME won't run on CGA. Better yet, buy an EGA or VGA setup. I used a VGA/386 system at MicroSphere to test these programs and I'm hooked. I've upgraded the recommended *minimum* system to EGA. CGA and Hercules fractals just don't cut it.

Mandelbrot Explorer

Mandelbrot Explorer sports online help, very powerful control over color assignment, some picture editing capability, and tips on photographing screens.

In the PC tradition, ME accepts command line parameters and operates with keystroke commands. The command line capability means that you can easily set up a batch file directing ME to do any number of fractals unattended. Wonderful! Head off on a month-long vacation while your computer sits home alone scratching out fractals.

Another nice feature called Escape lets you watch the dance of a point as it goes through the iterative process.

I do not like ME's method of screen boundary input. Instead of taking maximum and minimum values, ME wants to see the coordinates of the center of the region and a "magnification." Very unnatural.

Fractal Magic

Written in Turbo Pascal, FractalMagic comes in two flavors — regular, and 80x87. And if you want to create your very own fractal set, Advanced FractalMagic ($25) lets you write new calculation routines that can be linked in with the FractalMagic code. (You'll also need Turbo Pascal v.4.0.)

FM uses pull-down menus to create a very nice user interface. Screen parameter input follows the more standard method of specifying maxima and minima. During operation FM shows a bright dot at the point of calculation — a very nice touch that makes it easy to see where the heck you are.

The makers of FM also sell a screen dump utility, T-shirts, and another inter-

esting sounding graphics program called KaleidoScope.

Which One?

A rough comparison using the full Mandelbrot set gave FM about a 15% faster rating. I didn't get a chance to compare the two on an 80x87 system.

I'd be hard pressed to make a choice between the two programs. I prefer the user interface of FractalMagic, and it has the edge in speed. But Mandelbrot Explorer's batch ability and its slick Escape feature tip the balance back towards even.

Shoot, it's only money — buy 'em both.

Mandelbrot Explorer $30
Peter Garrison
1613 Alvito Way
Los Angeles, CA 90026

FractalMagic $35
Sintar Software
P.O. Box 3746
Belleview, WA 98009

References

[1] Clarke, Keith C., "Computation of the Fractal Dimension of Topographic Surfaces Using the Triangular Prism Surface Area Method," *Computers & Geosciences*, Vol. 12, No. 5, pp. 713-722.

[2] Hopke, Philip K., et al, "The Use of Fractal Dimension to Characterize Individual Airborne Particles," paper presented to the EPA/APCA Symposium on Receptor Models in Air Resource Management.

[3] Barenblatt, G.I., et al, "The Fractal Dimension: A Quantitative Characteristic of Ocean-Bottom Relief," *Oceanology*, Vol. 24, No. 6, pp. 695-697.

[4] Turcotte, D.L., "A Fractal Model for Crustal Deformation," *Tectonophysics*, Vol. 132 (1986), pp. 261-269.

[5] Allain, C. et al, "Optical Fourier Transforms of Fractals," *Fractals in Physics*, Pietronero, L. and Tosatti, E. editors, ISBN 0-444-86995-6, (Amsterdam: Elsevier Science Publishers B.V., 1986), pp. 61-64.

[6] Lucchin, F., "Clustering in the Universe," *Ibid.*, pp. 313-318.

[7] Grassberger, P., "Spreading of Epidemic Processes Leading to Fractal Structures," *Ibid.*, pp. 273-278.

[8] Liu, S.H., et al, "Theory of the AC Response of Rough Interfaces," *Ibid.*, pp. 383-389.

[9] Sander, L., "Fractal Growth Processes," *Nature*, Vol. 322, August 28, 1986, pp. 789-793.

[10] Tsonis, A., et al, "Fractal Characterization and Simulation of Lightning," *Beiträge zur Physik der Atmosphäre* (ISSN 0005 8173), Vol. 60, May, 1987, pp. 187-192, English language.

[11] Maddox, J., "Gentle Warning on Fractal Fashions," *Nature*, Vol. 322, July 23, 1986, p. 303.

♦ ♦ ♦

Hacking Sprint: Creating Display Drivers

Brett was one of the original architects of the IEEE 802.5 Token Ring LAN, and coauthored Living Videotext's ThinkTank 2.0. He's also an instigator of the annual Hackers' Conference, which makes him a natural for hacking Sprint, Borland's flexible new word processor.

Life in the Silicon Valley isn't quite as idyllic as it is in Central Oregon, but it has its pleasures: the challenge of identifying smog-shrouded friends from 6 feet and the romance of the regional car crawl during rush hour.

My desire to do something other than fight the gridlock one Friday evening led me to crack the imposing manuals I'd received with Borland's Sprint (the word processor).

As it turned out, the result justified the effort. If you're picky about your editing environment, learning how to hack Sprint will give you unprecedented control over the way you view or manipulate text.

A Complete Word Processor

Sprint is the progeny of a long and distinguished line of text editors and formatters that began at MIT and CMU with Richard Stallman's EMACS (Editing MACroS) editor and Brian Reid's Scribe text formatter.

Versions of these two programs, mainstays of the academic computing community, first appeared on microcomputers as MINCE ("MINCE Is Not Complete EMACS") and Scribble. These were the foundations for products such as Perfect Writer, Amethyst, and Final Word.

Sprint, derived from Final Word II, provides almost every feature a user could ask for. Imagine a text editor that comes with nine user interfaces, four built-in "languages," a text formatter, a spelling checker, a thesaurus, a file format converter, a mail merge, an automatic save feature, and support for dozens of printers, screens, and terminals.

Sound intimidating? It might be, had Borland not invested a great deal of time and effort to hide much of the complexity of this rich environment from the novice.

Sprint emerges from the box accompanied by a menu-driven setup, pop-up menus within the editor, and well-chosen defaults. Many of you will never need to delve any deeper into the mysteries of Sprint.

My desire to do something other than fight the gridlock one Friday evening led me to crack the imposing manuals I'd received with Borland's Sprint (the word processor).

However, if you do get into the "advanced" portions of the documentation (the Reference Manual and the Advanced User's Guide), you'll discover that Sprint has unmatched facilities for user customization and enhancements.

Figure 1 shows an overview of the Sprint system. Note that, at every place where there's an arrow in the diagram, a user-accessible "language" can create new commands and menu text.

A C-like macro language controls the interpretation of keystrokes and allows redefinition of EVERY KEY COMBINATION. So the underlying EMACS-like editor can emulate the user interfaces of WordPerfect, Microsoft Word, and SideKick.

A configuration "language" controls how Sprint interacts with the screen and printer. The Scribe-like formatting "language" helps to organize your output. And a record-oriented "language" lets you control the SprintMerge mail merge. This configurability makes Sprint a hacker's editor par excellence.

Adapting Sprint To A New Screen

Each of the languages and interfaces in Sprint deserves a small book of its own. In this article I'll focus on one of Sprint's more unusual features — its ability to run nearly any video display adapter or terminal.

During the course of my experimentation with Sprint, I've sent characters to the screen via direct memory mapping, software interrupts, the IBM PC BIOS, MS-DOS, a serial port, and I/O ports — all without major problems. Because you can load specific processor registers with selected values and call a software interrupt, you can write a TSR that does almost anything with Sprint's screen output.

To provide a real-world example, Earl Hinrichs of PC Tech was kind enough to let me borrow a high resolution monochrome video board and monitor to test Sprint's configurability.

The PC Tech video board, which uses a TMS34010 graphics CPU, produces a 736x1024 "portrait" display with up to 66 lines by 80 characters. (It looks as if it's possible to get still more characters on the screen, but this is the largest number currently supported without a special driver.)

By Brett Glass
P.O. Box 817
Palo Alto, CA 94302-0817

I started out by reading Appendix F of the *Sprint User's Guide* ("Build Your Own Screen and Printer Drivers"). This section describes the format of a Sprint "library file" — a file which describes screens and printers to the Sprint configuration utility (SP-SETUP).

A library file consists of ASCII "records" that define the characteristics of printers, screens, and I/O ports. If you're defining a printer device, there's also a construct called a "subrecord" that defines fonts, character widths, attributes, and character translation for a printing device.

The Library Record Format

A record in a library file must begin at the left margin and start with the word "printer," "screen," or "port." Each record consists of the name of a device, followed by a comma and one or more fields. Each field gives information about the device, and consists of a field name, a space, and a value (if required) for the field. Commas separate the fields.

A record can span more than one line; if it does, lines after the first *must* be indented. Comments, which begin with double semicolons, can appear in a record; if they appear in the first column, the setup program displays them as you configure Sprint.

Library records for screens have fields that specify a large number of options. Figure 2 shows an example of a library record (taken directly out of Borland's library of screen definitions) for a screen accessed via the IBM PC BIOS.

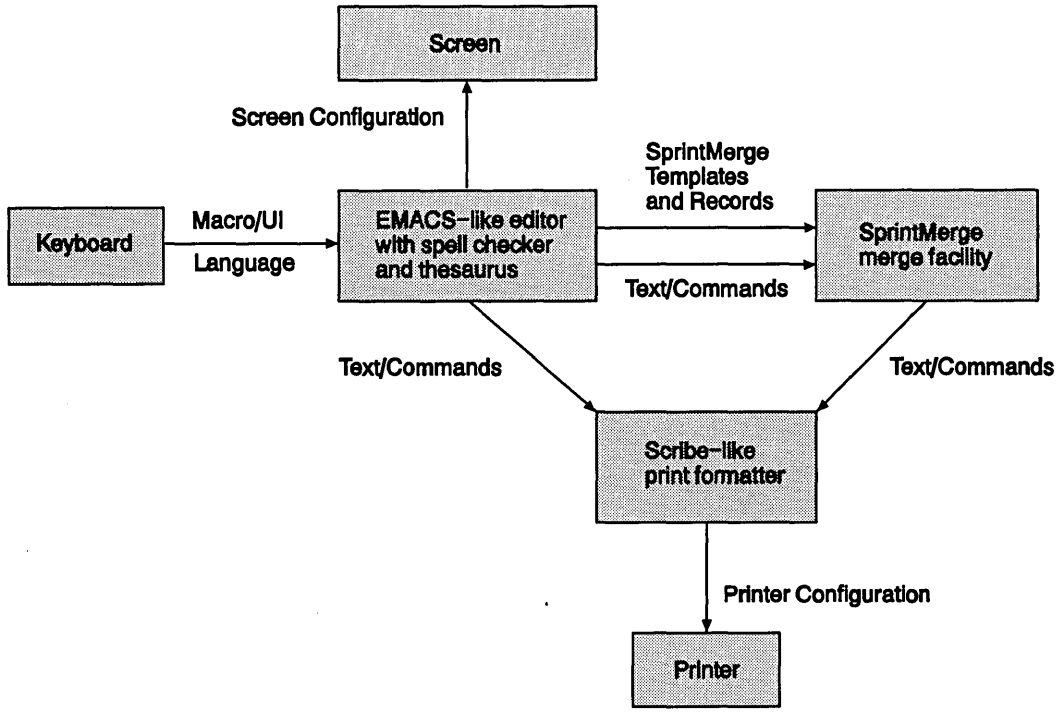
Let's examine this record bit by bit. First, the word "screen" in the first column indicates we're defining a screen device; the name of the device, ".BIOS," follows immediately thereafter.

The fact that the name of the device begins with a period has special significance. Devices whose names begin this way won't appear on the setup program's menu, but you can create devices that *are* visible on the menu with a record like —

```
screen IBM PC BIOS, as .BIOS
```

This record will cause an entry called "IBM PC BIOS" to appear on the screen setup menu, and the special "as" field

Figure 1 — Sprint System Overview



will invoke all the same field settings as the hidden ".BIOS" device would.

The first three fields of the .BIOS record, "cur," "up," and "down," are set to the capital letter "B." This setting has a special meaning: It causes Sprint to use the IBM PC BIOS to position the cursor and move it up or down. It's also possible to place strings in these fields which convert the row and column into a cursor positioning string to be sent to a terminal.

The next group of fields, which includes "help," "infobox," "menufirst," "menu," "error," "status," "select," "plain" and a series of control characters, selects the attributes used to paint the screen. You can modify these hex numbers for maximum visibility, or simply to suit your tastes.

Finally, the "reset" field tells Sprint what to do when you exit the editor. The string in our example manipulates the IBM PC hardware directly via the H (hardware string) construct. The sequence —

```
0>AL 0>CX 1850h>DX 7>BH 6 int 10h
```

puts 0 in AL, 0 in CX, 1850 hex in DX, 7 in BH, and 6 in AH, then invokes software interrupt 10h. This causes the BIOS to clear the screen.

Then, the sequence —

```
0>DX>BH 2 int 10h
```

puts 0 in DX and BH and 2 in AH, and invokes Int 10h again. This positions the cursor in the upper left-hand corner of the screen, in preparation for the DOS prompt to appear.

This example exploits only a small number of the facilities available through the Sprint library file "language." Other features include a case construct, control string formatting, binary arithmetic, and direct writes to memory and I/O ports. It's hard to imagine a screen control function you couldn't write with this language.

The PC Tech Monochrome Display

I installed the PC Tech Monochrome Display in my PC almost without glancing at the instructions. The monitor has an automatic on/off feature that causes it to turn on when your PC boots. It doesn't require a separate power switch — a very nice touch.

I turned my PC on and the board came up emulating an IBM CGA card. (The emulation was good enough, by the way, that even IBM's OS/2 Standard Edition 1.0 — known to be sensitive to minute hardware incompatibilities — didn't notice the difference.)

Figure 2 — ".BIOS" Screen Definition Record

```
screen .BIOS,cur B,up B,down B, ;; use "bios" calls only
        help 7h, infobox 7h, menufirst 0fh, menu 07h, error 070h,
        status 070h, select 070h, plain 07h,     ;; extra atributes
        ^A 070H, ^B 0FH, ^C 070H, ^D 01H, ^E 01H,
        ^J 020H, ^K 070H, ^L 07H, ^M 01BH, ^O 070H, ^P 070H, ^Q 070H,
        ^R 01H, ^S 070H, ^T 01h, ^U 01H, ^V 070H, ^W 070H, ^X 01H,
        ;; to prevent clear, remove the line below...
        reset H0>AL 0>CX 1850h>DX 7>BH 6 int 10h 0>DX>BH 2 int 10h
        ;; clear screen on exit - to set color instead of white
        ;; use "%>BH" instead of "7>BH"
/***/
```

Figure 3 — PC Tech Display Screen Definition Record

```
screen PC Tech\, 66 lines,as .BIOS,
;; PC Tech mono display (66 lines)
        map B800h,rows 66,init Cmovescrn\ C\ S\ 80\ 66,
        reset H0>AL 0>CX 4150h>DX 7>BH 6 int 10h 0>DX>BH 2 int 10h
/***/
```

The display, which uses two bits per pixel, maps colors to combinations of black, dark grey, light grey, and white, allowing most CGA programs to produce readable output despite the lack of color. However, because the display was acting like a CGA, it limited the text to 25 lines at the top of the screen — a terrible waste of high-resolution display space.

Fortunately, you can expand the PC Tech display to use more of the screen by loading a special TSR, called TSR10, that communicates with the TMS34010 graphics processor. Once you've loaded the TSR, the command —

```
MOVESCRN C S 80 66
```

maps more of the display area into the PC's memory, yielding an 80 by 66 display.

However, at this point Sprint still wouldn't have known that the display wasn't a CGA. So I created a new screen record (Figure 3) and inserted it into the MAIN.SPL file on my Sprint program disk.

This record contains several tricks I'd picked up by looking at other device descriptions in the MAIN.SPL file.

First, I wanted the menu entry on Sprint's configuration menu to show the string "PC Tech, 66 lines." However, in order to keep the setup program from mistaking the comma for the one that comes at the end of the device name, I had to preface it with a backslash.

I then included the field "as .BIOS" in the entry, since most of the IBM PC BIOS calls (cursor positioning and scrolling) work properly even on the larger display. This saved a lot of typing and kept me from having to respecify all the attributes.

The comment immediately after the first line begins at the left margin, so it appears when the device is selected in the setup program. This isn't absolutely necessary, but conforms to what I've done for other displays.

The map field tells Sprint the segment address of the display, so that it can bypass the BIOS and write characters directly to the TMS34010's dual-ported memory. Like the CGA, the PC Tech board resides at B8000h, unless it's specifically remapped to emulate the Hercules Graphics Card.

The rows field tells Sprint how many rows the display has. This, combined with the map field, defines the area of display memory used by Sprint.

The init field tells Sprint to invoke MOVESCRN as part of the initialization sequence for the board. The backslashes before the spaces in the command ensure that the setup program won't mistake them for the end of the command.

Finally, I set the reset field to a sequence *almost* the same as that for .BIOS — but not quite. By setting DX to 4150h rather than 1850h before invoking the clear screen BIOS call, I cause the BIOS to clear the screen down to line 41h (65, counting from zero) rather than line 18h (24) when I exit the program.

When I invoked the SP-SETUP program, lo and behold, the PC Tech display was on the menu — and after selecting it I was able to edit a whole typewritten page of text at one time.

Even nicer, I could open eight files on the screen at once and move back and forth between them — excellent for editing programs that span multiple files. (Then, of course, I wrote a simple macro that did a "make" on all the files I'd edited. Sprint can be addicting.)

Special Fonts And More

Just displaying the fonts as different shades of grey, black, and white (as in this configuration) is still barely scratching the surface of what can be done with the TMS34010 processor on the PC Tech display board.

The graphics CPU is capable of displaying boldface, underlined, subscript, superscript, strike-through, and other fonts on the screen pretty much as they're going to be printed, given only a few enhancements to the software on the board. (The Hercules RAMFont card already performs some of these functions with Sprint, but since it does them in hardware it may not be as flexible.)

At this writing, I haven't yet figured out how to modify the software on the coprocessor to display all the variations on the fonts, but it's certainly possible — and I hope the folks at PC Tech will lend a hand.

Macros And Other Business

Sprint's configurability in one small area (the development of display drivers) gives a hint of what it's capable of doing in others. I've seen Sprint editing macros that play tunes on the PC's speaker, edit your directory (veteran EMACS users will recognize this as an adaptation of a macro called "DIRED" on older EMACS systems), and perform other functions you might not usually see inside an editor.

The sheer size and scope of Sprint has bewildered a few reviewers. But there's no question that at least one class of user (the ingenious hacker) will revel in the possibilities. If you fall into this category, or even if you don't, you may want to put Sprint through its paces and see what you can create.

♦ ♦ ♦

Great C Comparison Of 1989!

By Scott Robert Ladd
P.O. Box 61425
Denver, CO 80206
(303) 322-7294

Here we go again. This is Version 2.0 of Scott's "everything you ever wanted to know about C compilers." In this updated version you'll get faster compiles, larger libraries, newer debuggers, and fancier manuals.

Yes, it's that time of year again ... when my family wonders about my sanity, my friends think I've disappeared, and I find myself buried in boxes, disks, and manuals. Once again, I'm reviewing the great hoard of MS-DOS C compilers.

The lineup has changed somewhat. Last year, I looked at 11 compilers; this year it's 12. Nearly all of last year's compilers (the ones still on the market) have undergone major upgrades. There are two significant newcomers — WATCOM C and MIX Power C. Zortech now sells what used to be the Datalight compiler.

Mark Williams' Let's C, a fine product I recommended in '88, is undergoing a major upgrade to full ANSI compatibility. The people at Mark Williams did not want their old compiler reviewed with a new product in the wings. As soon as I get the new version, I'll cover it in the regular column.

My methods have changed a bit. I've spent the past year honing the process, and at SOG VII several of you suggested things you'd like to see.

The Latest Tests

Figure 1 summarizes the results of my latest test. For the major benchmarks, I've listed the top five finishers. Note that, as in the coprocessor run-time for GRIND, the compilers at the top are only a few tenths of a second apart. A compiler which did not make the top five may still have been very close to the leaders.

All times are shown in seconds; EXE file sizes are in K (the size of the executable image divided by 1024). All times are the average of five tests. For QuickC and Turbo C, I used the command-line versions of the compilers. Compile and link times were done with optimizations turned off; after all, an optimizer usually only runs in the final compile. I compared run times after compiling for maximum speed.

"Em" means I used the software floating-point emulator, while "87" means I used the math coprocessor. All tests ran under MS-DOS v 3.21 on a 16 MHz 80386 PC with a 16 MHz 80387 math coprocessor and a 28 msec hard drive. I packed the drive so all compiler components and source files were contiguous. I used Microsoft LINK v 3.69 if the package didn't provide its own linker.

Figure 2 summarizes the compilers' features. It lists such things as ANSI compatibility, library features, memory models, and tools.

The Benchmarks

To begin with, the benchmarks have undergone *major* surgery. Where there were nine benchmark programs last year, there are now five. Only two of last year's benchmarks survived. The seven deleted benchmarks ran under fifty lines in length, and supposedly tested individual features of the compilers. In reality, they tested very little.

Real-world programs *do* something, whereas these tiny benchmarks were simple loops containing a few statements. Also, optimizing compilers cannot show their mettle on a tiny program. This year's benchmark suite consists of longer, more complex programs — a better test for optimizers.

DHRYSTONE 2 is a revision of the classic Dhrystone benchmark. The original author rewrote the program in C (the previous version was a C translation from Ada). He also cleaned up its logic and corrected some bugs. DHRYSTONE 2 contains a mix of statements designed to simulate a theoretically average program. Originally designed for comparing different hardware architectures, it's also very useful as a compiler benchmark.

FXREF was here last year. (See Figure 3.) This MS-DOS filter program creates a cross-reference listing of a text file. FXREF reads a file from standard input, and breaks it into text tokens. It then stores the tokens (along with line number references) in a binary tree. FXREF tests a compiler's I/O and dynamic memory allocation. The input to FXREF was its own source code.

GRIND has undergone very few changes. (See Figure 4.) It simulates a typical report pro-

Figure 1 — Benchmark Results

	Borland Turbo C v2.0B	C Ware DeSmet v3.1e	Comp.In C86 Plus v1.20D	EcoSoft Eco-C88 v4.16	Lattice C v3.31	Manx Aztec v4.10C	Metaware HIGH-C v1.4	Microsoft C v5.10	Microsoft Quick C v1.01	MIX Power C v1.2.0	WATCOM C v6.5	Zortech C v1.07
Dhry 2 Compile Time	6.1	7.3	55.5	10.5	15.4	13.7	24.5	14.7	6.6	16.6	19.7	6.4
Link Time	2.6	5.0	13.1	5.2	7.5	2.4	5.8	3.4	3.5	3.3	8.6	4.7
Em Run Time	20.3	23.6	20.7	23.4	23.7	16.9	19.3	16.6	19.6	21.3	17.9	17.5
Dhry / Sec	2463	2119	2415	2137	2110	2959	2591	3012	2551	2347	2793	2857
Em EXE Size	9.5K	16.5K	14.8K	12.8K	17.3K	7.7K	26.1K	10.7K	10.9K	18.6K	9.5K	10.0K
FxRef Compile Time	4.9	5.3	37.2	9.4	10.7	6.4	22.8	10.4	5.8	11.5	11.6	5.3
Link Time	2.5	4.7	14.3	5.4	7.4	3.4	5.8	3.5	3.5	2.8	9.7	4.5
Em Run Time	41.3	42.6	38.7	38.7	42.8	42.7	39.4	38.6	38.6	39.7	39.1	38.3
Em EXE Size	8.4K	13.5K	13.1K	11.6K	15.6K	8.8K	21.2K	8.9K	9.0K	12.2K	7.8K	7.7K
Grind Compile	4.1	6.5	34.0	8.2	9.1	7.1	22.5	9.8	6.0	10.6	12.3	5.0
Link Time	3.1	4.8	20.6	6.4	9.0	2.5	6.9	4.9	5.5	5.0	12.9	5.9
Em Run Time	28.8	34.2	77.4	32.3	30.4	30.6	34.8	26.9	27.6	27.2	24.5	25.1
87 Run Time	24.2	24.6	21.1	23.8	25.4	23.3	21.9	21.9	21.3	22.1	21.2	20.9
Em EXE Size	25.4K	16.5K	36.6K	16.0K	21.2K	11.1K	56.3K	26.1K	26.3K	22.3K	15.5K	19.2K
87 EXE Size	15.8K	14.0K	19.6K	16.0K	18.1K	9.8K	50.2K	18.4K	18.7K	19.7K	12.9K	19.1K
Sines Compile Time	2.8	4.0	13.7	4.8	5.5	4.1	17.6	6.4	4.7	5.4	6.5	4.2
Link Time	3.0	4.0	13.5	3.2	5.8	2.4	4.7	3.6	3.6	2.6	10.2	3.6
Em Run Time	138.4	69.5	302.2	62.4	62.7	76.0	114.6	120.0	156.4	37.5	25.1	46.4
87 Run Time	9.6	20.3	10.5	15.4	10.6	23.7	7.9	5.3	10.1	22.7	11.0	9.7
Em EXE Size	20.0K	7.5K	24.1K	4.3K	7.7K	5.7K	10.2K	16.9K	16.9K	5.3K	9.7K	6.3K
87 EXE Size	10.4K	5.0K	7.1K	4.3K	6.2K	4.6K	6.2K	9.3K	9.3K	3.7K	9.7K	6.1K
Empty Compile Time	2.4	3.7	8.6	4.1	4.3	2.8	14.7	4.9	3.7	4.4	3.5	3.7
Link Time	2.3	3.7	11.4	2.8	5.3	1.8	3.8	2.4	2.4	2.1	5.2	3.7
Em EXE Size	2.3K	4.5K	6.8K	1.7K	5.5K	1.9K	5.7K	2.2K	2.2K	2.1K	1.9K	3.4K

Emulator EXE Size Rankings

Rank	Dhry 2	FxRef	Grind	Sines	Empty
1	Manx	Zortech	Manx	Ecosoft	Ecosoft
2	Borland	WATCOM	WATCOM	MIX	WATCOM
3	WATCOM	Borland	Ecosoft	Manx	Manx
4	Zortech	Manx	C Ware	Zortech	MIX
5	Microsoft	Microsoft	Zortech	C Ware	Microsoft

Coprocessor Run Time Rankings

Rank	Grind	Sines
1	Zortech	Microsoft
2	Comp In	Metaware
3	WATCOM	Borland
4	Microsoft	Zortech
5	Metaware	QuickC

Compile Time Rankings

Rank	Dhry 2	FxRef	Grind	Sines	Empty
1	Borland	Borland	Borland	Borland	Borland
2	Zortech	C Ware	Zortech	C Ware	Manx
3	QuickC	Zortech	QuickC	Manx	WATCOM
4	C Ware	QuickC	C Ware	Zortech	QuickC
5	Ecosoft	Manx	Manx	QuickC	Zortech/C Ware

Coprocessor EXE Size Rankings

Rank	Grind	Sines
1	Manx	MIX
2	WATCOM	Ecosoft
3	C Ware	Manx
4	Borland	C Ware
5	Ecosoft	Zortech

Emulator Run Time Rankings

Rank	Dhry 2	FxRef	Grind	Sines
1	Microsoft	Zortech	WATCOM	WATCOM
2	Manx	Microsoft	Zortech	MIX
3	Zortech	QuickC	Microsoft	Zortech
4	WATCOM	Comp Inv	MIX	Ecosoft
5	QuickC	Ecosoft	QuickC	Lattice

gram. First, it reads a text file of 1000 floating point numbers. It then sorts values and performs calculations. Finally, it writes the calculated values to a disk file. GRIND tests I/O speed, loop optimization, and floating-point library speed.

SINES is a new benchmark. (See Figure 5.) It calculates the sine of every angle between 0 and 360 degrees. SINES is unique in many ways. First, the only variables in it are doubles. Second, it doesn't use a single library function. It primarily tests the speed of the floating-point code generated by a compiler. It's interesting that some compilers which have very fast math libraries create very slow standalone floating-point code. This test generates some surprising numbers.

EMPTY is *not* a practical program. The entire source for it is: main(){}. It was designed to show the size of the compiler start-up code, and how quickly a compiler could compile nothing. Like SINES, this test generated some unusual results.

Ah, now on to the meat of this review! First, I'll look at each compiler. Then, I'll discuss overall compiler performance, along with some recommendations and suggestions.

Borland Turbo C v2.0B

In two years, Borland has come from nowhere to take a significant portion of the MS-DOS C compiler market. While it has had problems, version 2.0 is a solid, complete product.

Borland gives you a lot of bang for your buck.

For $150, the Standard package gives you an integrated compiler/editor, a command-line compiler, and several programming utilities.

For $250, the Professional package adds a command-line debugger and macro assembler. As a beginner, a hobbyist, or someone on a budget, the Standard package is all you need. You can always add the debugger/assembler package for $150.

Installation is simple and the package doesn't take a lot of room. In fact, it's quite easy to use on a dual-floppy laptop.

The environment is nice with everything accessed via pull-down menus or hot-keys. Its biggest drawback is lack of mouse support. You can easily customize the editor, a Wordstar clone (like all Borland editors).

Debugger: The built-in debugger, while not as sophisticated as its standalone brother, certainly proves adequate for most debugging tasks.

The debugger can execute source-line by source-line. You can set watchpoints and breakpoints and you can change or examine variables. Once you solve a problem, you merely edit the source and recompile — all within the environment. It all works very easily and quickly.

Help: The package includes a memory-resident help facility. It loads into memory and can be accessed from within any editor via a hot key. The help is somewhat context-sensitive.

Errors: Turbo C does a better-than-average job of warning you about problems. For instance, it will yell if you use a variable before assigning it a value. It warns you about unused local variables, but not about unused formal parameters. You can control many of the warnings via compile options.

Documentation: The two paperback books are reasonably well-written and the tutorial section, friendly. You get lots of examples and lots of information about each function.

Unfortunately, Borland still puts function descriptions one-after-the-other, instead of beginning each on a new page. The habit of cross-referencing detailed function information to another, similar function can lead to *lots* of page flipping.

Library: The Turbo C library is large and complete. It supports most of the ANSI standard features and has all the MS-DOS extensions we've come to expect in a PC C compiler.

Borland's Graphic Interface (BGI) is one of the best graphics packages included with a compiler. It supports a wide variety of adapters and (at least to my mind-set) appears very well thought out and organized. I especially like the hardware-sensing option which lets you (with some care) write a very portable graphics program.

Finally: Turbo C is the fastest compiler in the review, although several others come close. None of its execution time or size performances were particularly exciting. On the SINES test, its emulator came out third from last.

C Ware DeSmet C v3.1e

This compiler has changed significantly since last year. Previously, DeSmet followed the K&R standard, ignoring many of the new ANSI features. Now, it supports the most commonly used ANSI features, including function prototypes.

This is the smallest compiler in the group; a minimum configuration will fit on a 360K floppy disk with some room left over! The complete package requires only 600K. While there is no installation program, the manual clearly describes the copying process.

Package: You get everything you need in the basic package — an editor, a linker, a compiler, an assembler, and a librarian. (The optimizer is extra.) The basic package only compiles to the small model. The large model option costs extra.

Environment: You can run the compiler from the editor and the editor will point out compile errors in the source. You can reconfigure the editor to work with non-IBM compatible MS-DOS machines.

The assembler is not MASM compatible, but should be useful for creating small routines and functions.

Debugger: You get a source-level symbolic debugger. Although not as fancy or sophisticated as some others, it does everything you need to track most bugs.

Documentation: The manual comes in a single three-ring binder. With it you get a sheath of update pages, which you must integrate. The manual has one big deficiency — no index.

Finally: While this compiler contains some of the ANSI features, many are still missing. For instance, it does not include any of the standard time functions so I had to write a special version of the DHRYSTONE 2 benchmark. On the plus side, DeSmet C has graphics and direct-video-display functions.

This compiler runs fast and produces very small .EXE files. The performance of resulting programs, however, ranged from average to below average.

Computer Innovations C86Plus v1.20D

C86Plus arrives with two three-ring binders. I wish I could recommend this compiler. It has the most enjoyable documentation to read, filled with anecdotes and humor. Unfortunately, compiler performance is the worst of the bunch.

You can tell that this compiler is a little different by the names they give the tools. For instance, they've named their linker program "Carol." Their object-module librarian is "Marion." The package has all the important tools except an editor and a debugger (an empty chapter in the manual waits for a future debugger).

Installation: You get an easy to use installation program which lets you set the locations of the compiler and tools.

Figure 2 — Compiler Features

Feature	Borland Turbo C v. 2.0	Comp.Inn. C86Plus v. 1.20D	C-Ware DeSmet v. 3.1e	EcoSoft Eco-C88 v. 4.0.16	Lattice C v. 3.31	Manx Aztec C v. 1.4C	Metaware High-C v. 1.4	Microsoft C v. 5.1	Microsoft Quick C v. 1.01	MIX Power C v. 1.2.0	WATCOM C v. 6.5	Zortech C v. 1.07
Utilities												
environment(1)	Y	N	Y	Y	Y	N	N	(8)	Y	(3)	(9)	Y
linker	Y	Y	Y	N	N	Y	N	Y	Y	Y	Y	Y
librarian	Y	Y	Y	N	Y	Y	N	Y	Y	N	Y	N
debugger	N	N	Y	Y	(3)	Y	N	Y	Y	(3)	Y	N
editor	Y	N	Y	Y	(3)	Y	N	Y	Y	(3)	Y	Y
assembler	N	N	Y	N	N	Y	N	(3)	(3)	N	N	N
make	Y	Y	N	(6)	N	Y	N	Y	Y	N	Y	Y
grep	Y	N	Y	N	N	Y	N	Y	N	N	N	Y
profiler	N	N	Y	N	N	Y	N	N	N	N	N	N
Models (4)												
Tiny (.COM)	Y	N	N	N	N	N	N	N	N	N	Y	Y
Small	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
Medium	Y	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	Y
Compact	Y	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	Y
Large	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
Huge	Y	Y	N	N	Y	Y	N	Y	Y	Y	Y	Y
Mixed	Y	Y	N	N	Y	Y	N	Y	Y	Y	Y	Y
Code Output												
.OBJ files	Y	Y	(5)	Y	Y	(5)	Y	Y	Y	N	Y	Y
OS/2 support	N	N	N	N	Y	N	(3)	Y	N	N	N	N
80x87 support	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
8087 sensing	Y	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	Y
186 support	Y	Y	N	N	Y	Y	Y	Y	N	Y	Y	N
286 support(2)	Y	Y	N	N	Y	Y	Y	Y	Y	Y	Y	N
386 support(2)	N	Y	N	N	N	N	(7)	N	N	N	N	N
ROMable	N	(3)	N	N	N	Y	N	Y	N	N	N	(3)
Assembler	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	Y
Source												
library	$150	Y	$89	$50	$500	Y	$2,000	$150	$150	$10	$225	$50
start-up	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	Y
Documentation												
tutorial	Y	N	N	N	N	Y	N	Y	Y	Y	Y	N
lang. ref.	Y	Y	N	N	Y	Y	Y	Y	Y	Y	Y	N
Support												
BBS	N	Y	Y	N	Y	Y	N	N	N	N	N	N
Commercial	CIS	BIX	N	N	BIX	N	N	CIS	N	N	N	N
Other												
Req's HD	N	Y	N	N	N	N	Y	Y	N	N	Y	N

Notes:

(1) Borland, Microsoft, and WATCOM provide integrated environments with the compiler, editor, and debugger built-in. The other products provide an editor which can also run the compiler (and possibly the debugger) while editing a program.

(2) Most of the compilers reviewed support protected-mode programming - only support for the special instructions available in real mode on these chips.

(3) Available at extra cost.

(4) Models
Tiny - code and data both in 64K (.COM model)
Small - small (up to 64k) code, small data
Medium - large (up to 1MB) code, small data
Compact - small code, large data
Large - large code, large data
Huge - large code, large data, individual data items can be 64k
Mixed - near and far items can be declared in one program unit

(5) Producing .OBJ files requires the use of a special utility.

(6) The compiler has limited, built-in make-like facilities

(7) 80386 protected mode support is available in a seperate product

(8) The large Microsoft Optimizing C compiler package includes Quick C. For example, the editor included with Microsoft C is actually the one from the Quick C environment. Also, although MS C cannot be placed on floppies, Quick C can

(9) WATCOM C comes with Express C, a integrated editor/compiler designed for the educational market.

C86Plus takes up a significant amount of disk space, making it difficult to use on a floppy-based system.

Library: C86Plus has a large, robust library. The language conforms to most of the early drafts of the ANSI standard. And you get typical MS-DOS extensions such as near and far pointers and direct access to video memory. The function, zork(), returns a weird string (apparently) significant to the popular computer adventure game.

Finally: Running C86Plus is simple; it supports the Microsoft C compiler switches. The product seems to be very stable.

Performance left much to be desired. Compiles were so slow I occasionally wondered if the program had locked up. The linker was no better. Programs ran in average times although they did very well on the coprocessor and FXREF execution times. The C86Plus floating-point emulator is *very* slow, often twice as slow as any other emulator.

Ecosoft Eco-C88 v4.16

Ecosoft has produced C compilers since the days of CP/M. Their MS-DOS offering was one of the first compilers to implement some of the ANSI features such as function prototypes and void. There have been few outward changes in this compiler since last year; most of the changes have been minor bug-fixes.

Documentation: This package's biggest weakness is the manual. Contained in a three-ring binder, it's haphazardly organized and doesn't contain much of the information I looked for. For instance, the package will link several different libraries with Ecosoft programs, but the manual provides little explanation of when to use the libraries.

Installation: The installation program lets you select language components and their location. It also lets you place the package on floppies.

This compiler uses more environment variables than any other. The environment variables show the locations and names of libraries, compiler components, and header files. One environment variable, EPICK, sets the default value for the "picky" flag (described below).

Package: Eco-C88 includes the compiler, a simple source-level debugger, and an editor. An object-module librarian is available separately. The package does not include a linker, which can be a problem if your version of MS-DOS doesn't include Microsoft's LINK. The debugger is nothing fancy — it doesn't have the windowed interfaces of some of the newer debuggers. Still, it does what it needs to — it tracks down bugs.

The compiler contains several interesting features. It can use "make" logic on the files specified, so that it will recompile any source files newer than their object modules. This was provided in lieu of a standalone make program.

Another compiler switch of interest, the so-called "picky" flag, behaves like the warning level switch found in other compilers. Eco-C88 includes several lint-like capabilities, and the picky flag determines how picky the compiler is about your code.

On the upside, Eco-C88 has a feature-rich library. It misses some of the ANSI functions but contains other interesting routines, including functions to manipulate binary trees and video displays.

It doesn't have a graphics library and some "standard" extended functions, such as the spawn() family, are conspicuously absent. Only Lattice provides less support for the proposed ANSI standard than Ecosoft.

Performance-wise, Eco-C88 runs about average. Though it generated smaller than average code, it took about an average amount of time to do it. The programs were reasonably fast, and Eco-C88 did come in fourth in the run time of the SINES benchmark (using the emulator.)

Lattice C v3.31

Lattice used to be king of the C compilers, but Microsoft passed them several years ago and Lattice never recovered. At the rate they're going, they never will.

Documentation: The manuals have seen the biggest improvement from last year. They're attractively typeset, well organized, and bound in two three-ring binders.

Installation: Installation is simple. The installation program lets you select which components to load to disk. The choices lie in a documented configuration file.

Package: This is a complete package with several tools, including the editor, debugger, and some unusual items like an OS/2 binder. Lattice is one of only two C compilers supporting OS/2 and Microsoft Windows (Microsoft, of course, being the other.) Lattice does not provide a linker. You can compile programs within the editor which will trap and track errors.

Debugger: The debugger is very primitive, basically a souped-up version of DEBUG. Lattice promised a CodeView-like symbolic debugger at COMDEX in 1987, but no one's seen it yet.

Lattice C probably supports more compiler switches than any other compiler in this review. Some of them are unique. For example, the -R switch automatically inserts the object modules into a library.

Finally: On the box they state that Lattice is an "ANSI" compiler. Unfortunately, nothing could be farther from the truth. Of all the compilers reviewed, Lattice is the farthest from compliance. It doesn't even support simple features, such as the new Pascal-like function headers. And it lacks many of the ANSI header files.

Lattice C's performance is nothing to brag about. Its only claim to fame in the benchmark tables was the performance of its floating-point emulator on the SINES run-time benchmark. Otherwise, it compiled slowly and produced average-sized .EXE files.

Manx Aztec C v4.10c

Manx puts out compilers for everything from Apple IIs to 68000-based machines. Its 80x86 compiler for PC compatibles comes in several flavors, ranging in price from $200 to $500. As the price goes up, so does capability. This review looked at the Commercial Package, the most expensive (and extensive) version available. This compiler has changed very little from last year.

Installation: Installation directions are sparse and there's no installation program, so you'll find it interesting figuring out which directories to create and which files to load. There's no excuse for not having at least a section discussing the recommended way to install the product.

Documentation: The manual still has two of the problems it had last year — a too small binder and poor organization. Manx adds additional documentation for each release level. Unfortunately, they don't integrate this information into the manual. The inserts have grown to the point that the three-ring binder will not completely close. It's often difficult to find information when it's divided among two updates and a two-part manual.

Package: This is one of the most complete packages I've seen. It includes every important utility. Though the editor won't call the compiler, the symbolic debugger, SDB, is nearly as capable as CodeView (or the Turbo Debugger). Unlike CodeView, it allows an un-

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
	CQL Query System (SQL retrievals plus windows)	$325
	GraphiC 4.1 (high-resolution, DISSPLA-style scientific plots in color & hardcopy)	$325
	PC Curses (Aspen, Software, System V compatible, extensive documentation)	$250
NEW!	Greenleaf Business Mathlib (exact decimal math, formatting, depreciation, interest, cash flow, statistics)	$250
	Greenleaf Data Windows (windows, menus, data entry, interactive form design)	$220
NEW!	Assembler Kit (by John Zarrella; includes listing generator & loader; requires signed license agreement)	$175
	Greenleaf Communications Library (interrupt mode, modem control, XON-XOFF)	$175
	TurboTEX (TRIP certified; HP, PS, dot drivers; CM fonts; LaTEX)	$170
NEW!	Sherlock (C debugging aid)	$170
	Greenleaf Functions (296 useful C functions, all DOS services)	$160
	Essential C Utility Library, Communications Library, or Resident C	$160
	Essential Communications Library (C functions for RS-232-based communication systems)	$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
	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
	CBTree (B+tree ISAM driver, multiple variable-length keys)	$115
	Minix Operating System (U**x-like operating system, includes manual)	$105
	PC/IP (CMU/MIT TCP/IP implementation for PCs)	$100
	B-Tree Library & ISAM Driver (file system utilities by Softfocus)	$100
	Tele Operating System (TeleKernel, TeleWindows, TeleFile, & TeleBTree by Ken Berry)	$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
	Wendin Operating System Construction Kit or PCNX, PCVMS O/S Shells	$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
NEW!	Kinetic Image Synthesizer (3-D animation system ... Saturday morning on your PC!)	$75
	XT BIOS Kit (roll your own BIOS with this complete set of basic input/output functions for XTs)	$75
	TE Editor Developer's Kit (full screen editor, undo command, multiple windows)	$75
	Professional C Windows (lean & mean window and keyboard handler)	$70
NEW!	Heap Expander (use LIM-standard expanded memory as an extension of the heap)	$65
	lp (flexible printer driver; most popular printers supported)	$65
	Quincy (interactive C interpreter)	$60
	PTree (parse tree management)	$60
	OBJASM (convert .obj files to .asm files; output is MASM compatible)	$50
NEW!	Icon-Tools (full-featured icon display and editing system)	$60
NEW!	Polyglot TSR Package (includes reminder, bookmark, virus catcher, cache manager, & speech generator)	$50
	HELP! (pop-up help system builder)	$50
	Multi-User BBS (chat, mail, menus, sysop displays; uses Galacticomm modem card)	$50
	Make (macros, all languages, built-in rules)	$50
	Coder's Prolog (inference engine for use with C programs)	$45
	Virtual Memory System (least recently used swapping)	$40
	C-Notes (pop-up help for C programmers ... add your own notes)	$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)	$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
NEW!	Polyglot RAM Disk (change disk size on the fly; includes utilities)	$30
	SP (spelling checker with dictionary and maintenance tools)	$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
NEW!	Install (automatic installation program; user-selected partial installation; CRC checking)	$25
	Pascal Compiler & Interpreter or Pascal-to-C Translator (P-codes, standard Pascal)	$25
	ICON (string and list processing language, Version 7.5)	$25
	FLEX (fast lexical analyzer generator; new, improved LEX)	$25
	LEX (lexical analyzer generator; an oldie but a goodie)	$25
	Bison & PREP (YACC workalike parser generator & attribute grammar preprocessor)	$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
	ANSI Forms (forms manager based on ANSI codes)	$20
	C Compiler Torture Test (checks a C compiler against K & R)	$20
	Benchmark Package (C compiler, PC hardware, and Unix system)	$20
	A68 (68000 cross-assembler)	$20
	List-Pac (C functions for lists, stacks, and queues)	$20
	XLT Macro Processor (general purpose text translator)	$20
	C/reativity (Eliza-based notetaker)	$15
	Data	
	DNA Sequences (GenBank 55.0 including fast similarity search program)	$150
	Protein Sequences (5,415 sequences, 1,302,966 residuals, with similarity search program)	$70
NEW!	SIC Codes (each SIC code with the name of the industry to which it applies)	$70
	Dictionary Words (234,932 words in alphabetical order, no definitions)	$60
	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 Floppy Almanac (high-precision moon, sun, planet & star positions)	$20
	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

acwlinfo@uunet.uu.net

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

Reader Service Number 4

limited number of breakpoints and can use keyboard macros. SDB does not support a mouse.

The Commercial Package has complete facilities for the creation of ROMable code. As the benchmark table shows, Aztec C produces very small executable files.

Finally: I found Aztec C easy to work with. The compiler has a simple syntax and only a few switches. Environment variables locate headers and libraries. My only problem came when trying to change the stack size. The manual documents three variables which must be set when there are actually four (_STKRED must be set to the minimum clearance between the stack and heap.) Again, the documentation is the weakest part of this package.

Aztec C did very well in the benchmarks. The compiler runs fast and generates small programs. While not amazing, the performance of the generated programs is at least average. Aztec C came in second in the execution speed of DHRYSTONE 2.

Metaware High-C v1.4

This product hasn't changed from last year. High-C's ads proudly proclaim that it was used to create several popular products (including dBASE III/IV and Q&A). However, it's quickly losing ground in that arena to Microsoft, WATCOM, and others.

Installation: Metaware's installation program copies the compiler components to any directory but there's no way you can run this package from a floppy. The compiler takes up over 600K!

Documentation: The manual barely fits into its three-ring binder. Divided into three sections, each with its own index, this is one of the better manuals in the group. It contains complete, clearly written information.

Only High-C supports "profiles," files which contain compiler directives and instructions. You create specific profiles for specific projects, thereby customizing the compiler's behavior.

Library: Metaware doesn't provide many utilities. It has no librarian or linker. The only utilities included are a GREP and a source code cross-referencer. While many experienced programmers don't use built-in utilities, beginners usually find them very useful.

High-C implements many common ANSI features, but it has a long way to go before it's in full compliance. The library doesn't include any extras. This compiler has one of the most boring libraries in the group with operating system access functions, but no graphics library.

Finally: High-C's performance is mediocre. Its compile times beat only C86Plus and it produces large executables. While High-C advertises as an optimizing compiler, it doesn't match the performance levels of some non-optimizing products. High-C turned in its best performances on the coprocessor run-time tests for GRIND and SINES, where it came in fifth and second, respectively.

Microsoft C v5.10

This is the industry standard. As a leader, it has its critics (who have some valid points). However, an overall view shows that Microsoft C has not become the standard merely because of the reputation of its parent.

Installation: Microsoft has continued to improve its installation program. It's very flexible, giving you control over what's put where. You can even change libraries without reinstalling the entire package.

Documentation: The documentation package includes three two inch thick three-ring binders containing six manuals, along with a paperback reference manual for QuickC. Microsoft's documentation is very thorough and liberally indexed.

Unfortunately they haven't integrated the upgrade for version 5.10 so you have to look in at least two places to be sure you have the correct information. I don't understand why companies with looseleaf binders can't integrate their updates.

Package: Microsoft includes a cornucopia (appropriate for *Micro C*) of utilities and tools with their C compiler. The most extensive of these is QuickC, an environment-based compiler sold separately (see the separate discussion of QuickC below.) Recent additions include a powerful editor and a grep utility.

Debugger: CodeView is Microsoft's windowing, source-level debugger. It looks a bit long in the tooth when compared to products like Borland's Turbo Debugger, but is still one of the best source-level debuggers available. CodeView lets you track and view data throughout the execution of a program. It does line-by-line and animated (slow motion) execution. You can store debugger commands in a script file.

One CodeView feature is sadly lacking in other debuggers — mouse support. I still think of CodeView as a fine product, in spite of its reliance on the archaic DEBUG and SYMDEB command set.

Editor: The Microsoft Editor (ME) would be a much better product with usable documentation (it's better for quick reference). However, if you spend the time, you'll find ME extraordinarily powerful. In many ways, it resembles Brief.

It does, however, have some unique capabilities. Instead of a built-in programming language, ME lets you write extensions in either C or Macro Assembler. You then compile them and they're linked in at run-time. (They're lightning fast.)

Using the Microsoft compiler is easy. A large number of switches control everything from code generation to the listing format. No other compiler fully supports OS/2, including multi-thread programs and dynamic-link libraries.

I've heard through the C community that Microsoft C 5.10 is buggy. It gained this reputation with 5.00. Microsoft rushed 5.00 to market, though that package had a significant number of bugs.

When 5.10 came out, it retained the stigma of its predecessor. I've had no problems with the compiler, although there still could be some.

Of course, misunderstandings about the optimizer could also generate bug reports.

For instance, the /Oa (included in the catchall /Ox switch) optimization switch tells the compiler to ignore the possibility of aliases in the program code. You get an alias when a memory location gets modified by more than one variable. (Often occurs when using pointers heavily.)

Aliasing makes C harder to optimize than other languages. The programmer needs to be aware of what aliasing is, to be sure it's safe to use the /Oa switch.

Finally: Microsoft C performs very well. It compiles slowly, but that shouldn't be a problem with QuickC included in the package. I use QuickC for most of my preliminary compiles, unless I'm using some features available only in its big brother.

Microsoft C generates fast code — it won the DHRYSTONE 2 benchmark by a considerable margin, though it produces larger than average .EXE files. Probably its biggest failure was the SINES benchmark, where its emulator performance was very poor. It's only fair to note that Microsoft offers an alternative math library — less accurate

than the emulator library, but considerably faster.

Microsoft QuickC 1.01

Microsoft's QuickC integrates editor, compiler and debugger functions. By the time this review reaches the stands, version 2.00 will be out. Version 2 is a complete rewrite of version 1.01, with many new features. Therefore, I'll keep this discussion short.

Quick C comes with a set of paperback manuals. Like the "big" Microsoft compiler, Quick C installs easily. It doesn't work well from floppy disks — it simply requires too much disk swapping, even with the special overlay provided.

Quick C offers most of the features of the big Microsoft compiler, though it doesn't support most of the optimizations. Microsoft produced QuickC for folks learning the C language. They figure those of you interested in an optimizing compiler will buy the larger package.

The package also includes a command-line version of the compiler — merely a driver which runs the integrated environment. It also includes a complete set of command utilities, along with a make and a librarian.

Finally: Quick C really performs. It compiles very quickly (as its name would imply), and produces very fast programs. QuickC out performs Borland's Turbo C (its most common rival) in most of the .EXE run-time benchmarks. Its worst performance came on the emulator run-time test for SINES, where it ranked second slowest. (Microsoft doesn't offer an alternative math library for QuickC.)

MIX Power C v1.2.0

You won't find a less expensive compiler available, for any language. At $20 (basic package), Power C costs less than most computer books.

Installation: Nothing fancy here, you just copy the compiler to floppy disks or your hard drive. Everything about this package is budget-level, except for the manual.

Documentation: MIX worked hard to create a manual useful to the novice C programmer. The tutorial contains one of the best explanations of pointers I've seen.

The manual does have problems, though. They've separated the information on compiler switches from the instructions on running the compiler. The manual includes compile options that don't work yet (the READ.ME file notes

Figure 3 — FXREF.C

```
/*
Program:  FXREF (File Cross-Reference)

Version:  1.10
Date:     21-Sep-1988
Language: ANSI C

Reads a file from standard input, and sorts and organizes each token
(word) found using a binary tree, keeping track of the number of oc-
curences of each token and their location by line and column. It then
prints a report to stdout.
Released into the public domain for educational purposes.
*/

#include "stdio.h"
#include "string.h"
#include "ctype.h"
#include "stdlib.h"

/* type definitions */
typedef unsigned short line_no;

typedef struct loc_s
     {
     line_no line;
     struct loc_s * next;
     }
     location;
typedef struct tok_s
     {
     struct tok_s * less, * more;
     char * text;
     struct loc_s *loc, *last;
     }
     token;
token * root;
char * err_msg[] = {
                   "token table root",
                   "token text",
                   "location references",
                   "token record"
                   };
/* function prototypes */
int main(void);
void parse_tokens(char *, line_no);
void add_tree(char *, line_no);
token * find_tree(char *);
void show_tree(token *);
void error(short);

int main()
     {
     char buf[256];
     line_no line=0;
     if (NULL == (root = ( token *)malloc(sizeof( token))))
          error(0);
     root->less = NULL;
     root->more = NULL;
     root->text = NULL;
     root->loc  = NULL;
     while (NULL != (fgets(buf,256,stdin)))
          {
          ++line;
          printf("%5u: %s",line,buf);
          parse_tokens(buf,line);
          }
     printf("\x0C\n");
     show_tree(root);
     return 0;
     }

void parse_tokens(char * buf, line_no line)
     {
     char tok[256];
     line_no pos;
     while (1)
          {
          while ((!isalpha(*buf)) && (*buf != 0))
               ++buf;
          if (*buf == 0)
               return;
          pos = 0;
          while (isalpha(*buf))
               tok[pos++] = *buf++;
          tok[pos] = 0;
```

continued next page

this). The error reference section merely lists the error messages, there's no explanation of them.

Package: For $20, you get the compiler and linker — that's it. MIX's optimizer is part of the compiler's code generator.

The compiler does support a wide variety of floating point options but has only one memory model — medium. This is why I did Power C's benchmarks with the medium model while I used the small models for other compilers.

Additional packages are available. The CTrace debugger costs $19.95, as does a database toolbox and a BCD math package. The library source code sells for $10 and contains a simple 8088 assembler as a bonus.

As with all advertising claims, the comparative performance chart in the Power C ads and on the back of its manual are a bit optimistic. It compiles fast, but nowhere near as fast as Borland and Zortech. It produces small, swift programs, but not usually the fastest. The most spectacular performance by Power C was on the emulator tests for the GRIND and SINES. Only WATCOM beats Power C's 37.5 seconds on the emulator run time test for SINES.

WATCOM C v6.5

WATCOM gives Microsoft a run for its money. A spinoff from the University of Waterloo in Canada, the company is well-known for its mainframe FORTRAN compilers (WATFOR and WATFIV). When they entered the PC market last year with version 6.0 of their C compiler, they made a splash that is still sending ripples through the MS-DOS C community.

Installation: WATCOM includes a simple program for installing the compiler and its components onto a hard drive. (Don't even bother trying it on floppies.) As with most compilers, it uses environment variables to direct the compiler and utilities to needed files.

Documentation: They've divided documentation into five spiral-bound books. You also get a stack of quick reference guides, one for each major program. All the manuals have complete, and logical, indexes.

Package: This package has everything, though the editor isn't particularly exciting. It emulates the old IBM mainframe editors by being slow and clumsy. But at least it exists.

Debugger: The debugger is not as easy to use as its competitors, but it has plenty of features. For instance, I

```
            add_tree(tok,line);
            }
      }

void add_tree(char * tok, line_no line)
      {
      token *temp_tok, *new_tok;
      location *temp_loc;
      short comp;
      if (root->text == NULL)
            {
            if (NULL == (root->text =
                        (char *)malloc((unsigned)strlen(tok)+1)))
                  error(1);
            strcpy(root->text,tok);
            if (NULL == (root->loc =
                        ( location *)malloc(sizeof( location))))
                  error(2);
            root->loc->line = line;
            root->loc->next = NULL;
            root->last = root->loc;
            return;
            }
      temp_tok = find_tree(tok);
      if (comp = strcmp(tok,temp_tok->text))
            /* comp is true (non-zero) if they don't match */
            {
            if (NULL == (new_tok = ( token *)malloc(sizeof( token))))
                  error(3);
            if (NULL == (new_tok->text =
                        (char *)malloc((unsigned)strlen(tok)+1)))
                  error(1);
            new_tok->less = NULL;
            new_tok->more = NULL;
            strcpy(new_tok->text,tok);
            if (NULL == (new_tok->loc =
                        ( location *)malloc(sizeof( location))))
                  error(2);
            new_tok->loc->line = line;
            new_tok->loc->next = NULL;
            new_tok->last = new_tok->loc;
            if (comp < 0)
                  temp_tok->less = new_tok;
              else
                  temp_tok->more = new_tok;
            }
        else
            /* if comp is false (0), the tokens match */
            {
            if (NULL == (temp_loc =
                        ( location *)malloc(sizeof( location))))
                  error(2);
            temp_loc->line = line;
            temp_loc->next = NULL;
            temp_tok->last->next = temp_loc;
            temp_tok->last = temp_loc;
            }
      }

 token *find_tree(char * tok)
      {
      short comp;
      token *node;
      node = root;
      while (1)
            {
            if (0 == (comp = strcmp(tok,node->text)))
                  return node;
            if (comp < 0)
                  if (node->less == NULL)
                        return node;
                    else
                        node = node->less;
              else
                  if (node->more == NULL)
                        return node;
                    else
                        node = node->more;
            }
      }

void show_tree(node)
 token *node;
      {
      location *lloc;
      short pos;
      if (NULL == node) return;
      show_tree(node->less);
```

```
     printf("%-32s: ",node->text);
     pos = -1;
     lloc = node->loc;
     while (lloc != NULL)
          {
          if (++pos == 7)
               {
               pos = 0;
               printf("\n%32s: "," ");
               }
          printf("%5d ",lloc->line);
          lloc = lloc->next;
          }
     printf("\n");
     show_tree(node->more);
     }

void error(short err_no)
     {
     printf("\nFXREF ERROR: Cannot allocate \
space for %s\n",err_msg[err_no]);
     exit(err_no+1);
     }
/***/
```

Figure 4 — GRIND.C

```
/*
Program:  Grind

Version:  1.11      Date:     26-Oct-1988      Language: ANSI C

Tests all aspects of a C compiler's functions, including disk i/o, screen
i/o, floating point, recursion, prototyping, and memory allocation. It
should be a large enough program to test the advanced optimizers in some
compilers.
Developed by Scott Robert Ladd. This program is public domain.
*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define MAXFLOATS 1000

struct tabent
     {
     double val, vsum, vsqr, vcalc;
     };

struct tabent table[MAXFLOATS];

char *errmsg[] = {
                 "GRIND.TBL cannot be created",
                 "GRIND.TBL cannot be closed properly",
                 "GRIND.IN cannot be found",
                 "GRIND.IN has been truncated",
                 "GRIND.IN cannot be closed properly"
                 };

/* function prototypes */
short main(void);
void  readfloats(void);
void  sortfloats(void);
void  quicksort(struct tabent *, short, short);
void  maketable(void);
void  writetable(void);
void  error(short);

short main(void)
     {
     puts("\nGrind (C) v1.10  -- A Benchmark Program\n");
     readfloats();
     sortfloats();
     maketable();
     writetable();
     puts("\7End run GRIND!!!");
     return(0);
     }

void readfloats()
     {
     register short i;
     char buf[12];
     FILE *fltsin;
     printf("--> Reading in floats. At #     ");
     if (NULL == (fltsin = fopen("GRIND.IN","r")))
```

continued next page

couldn't figure out how to remove the assembler trace window. The debugger doesn't remember the video mode of the program being debugged. If the debugger is in 43-line mode on an EGA, so is the output from your program. However, you can assign actions to watchpoints, something sadly lacking from CodeView and Turbo Debugger.

Express C is WATCOM's integrated compiler. Often, it produces .EXE files over 80K larger than necessary because it includes debugging information. However, as a prototyping tool, Express C is very good. The editor and debugger are versions of the standalone utilities, and work well enough for a classroom. Express C runs much faster than the main C compiler does.

WATCOM C has a very interesting code generator. One of the reasons for its sterling performance is that the compiler deftly uses AX, BX, CX, DX, SI, and DI to store function parameters. (You can disable this feature.)

Finally: The WATCOM C compiler runs relatively slowly but it produces small, fast programs. (WATCOM C's floating-point emulator is amazingly fast.) WATCOM, however, has a problem compiling the SINES benchmark for the coprocessor. When I forced it to use inline coprocessor instructions (the -7 switch), the resulting program went into an infinite loop.

Zortech C v1.07

Zortech now sells what used to be Datalight Optimum-C. In fact, they've managed to improve the product.

Installation: An automated installation program lets you select files and directories for this package. You can also set an array of environment variables to configure the compiler. Zortech C can easily run on a floppy disk-based PC.

Documentation: You get a 600 page paperback which includes a simple introduction to C, a description of compiler switches and components, and a look at optimization.

Package: The only tool missing from this product is a debugger. Zortech will have a symbolic debugger available sometime in early 1989. In the meantime, version 1.07 fully supports Microsoft's CodeView.

The ZED editor was designed for programming. You can compile programs from within it, and track the errors in the edit buffer.

Zortech provides a memory-resident, context-sensitive help facility. It will work from inside any editor. Pressing

the help hot key will bring up a subject related to the item at the current cursor location. This way you get help while working in your favorite editor.

Zortech C includes several utilities. Many of these, such as the timer and grep programs, come with source code.

Library: The library is quite complete. It includes several standard functions, such as strtok() and sleep(), not mentioned in the manual. (I also noticed they'd left out or misplaced examples in the function reference section.) Zortech has supplied a graphics library (as have other vendors) and their Flash Graphics are very fast.

Unlike Datalight, Zortech doesn't include the source to the library functions. Source costs you an extra $50.

Finally: On the benchmark tests, Zortech C did amazingly well. It compiled very rapidly and produced small .EXE files. Zortech showed up in almost all the "top five" charts in the benchmark table. Its file I/O and floating-point emulator performed admirably. Were I to rank compilers solely by performance, Zortech would lead by a wide margin.

Considerations

If you expect me to tell you which compiler to purchase, you're going to be disappointed. The compiler you need depends upon your resources and what you're doing. Remember that these are only benchmarks, not real life, and a group of comparisons won't be as thorough as individual reviews.

Microsoft C has slipped a bit (in comparison to the other products) since last year, but remains an excellent value. It provides a robust environment and good performance. You can pick up this package for under $300 from many mail order houses. (Watch for object-oriented facilities. They've scheduled a new release for mid-1989.)

WATCOM C has taken the MS-DOS C world by storm. If you work with floating point, there is no other choice. This compiler's biggest liability is compile time. Otherwise, it performs amazingly well. Watch for version 7.0, due out in early 1989.

Borland's Turbo C is blazingly fast. It doesn't produce the fastest executables, but it's still faster than average. Borland's produced an amazing debugger and an excellent graphics library.

Manx Aztec C remains a good value. It comes with a great debugger and is optimized for producing ROMable code. Anyone who works with embedded systems should have Aztec

```
            error(2);
      for (i = 0; i < MAXFLOATS; ++i)
            {
            printf("\b\b\b\b\b%5d",i);
            if (NULL == fgets(buf,12,fltsin))
                  error(3);
            table[i].val = atof(buf);
            }
      if (fclose(fltsin))
            error(4);
      printf("\n");
      }

void sortfloats()
      {
      puts("--> Sorting data");
      quicksort(table,0,MAXFLOATS-1);
      }

void quicksort(struct tabent * item,
               short left,
               short right)
      {
      register short i, j;
      struct tabent x, y;
      i = left;
      j = right;
      x = item[(i+j)/2];
      do
            {
            while (item[i].val < x.val && i < right) i++;
            while (x.val < item[j].val && j > left)  j--;
            if (i <= j)
                  {
                  y = item[i];
                  item[i] = item[j];
                  item[j] = y;
                  i++;
                  j--;
                  }
            }
      while (i <= j);
      if (left < j)  quicksort(item,left,j);
      if (i < right) quicksort(item,i,right);
      }

void maketable()
      {
      register short i;
      double sum = 0.0;
      puts("--> Calculating table values");
      for (i = 0; i < MAXFLOATS; ++i)
            {
            sum = sum + table[i].val;
            table[i].vsum  = sum;
            table[i].vsqr  = table[i].val * table[i].val;
            table[i].vcalc = sqrt(fabs(table[i].val)) *
                             log10(fabs(table[i].val));
            }
      }

void writetable()
      {
      FILE *fd;
      register short i;
      if (NULL == (fd = fopen("GRIND.TBL","w+")))
            error(0);
      puts("--> Writing Table to File");
      for (i = 0; i < MAXFLOATS; i = i + 10)
            {
            fprintf(fd,
               "val = %5.2f, sum = %5.2f, sqr = %5.2f, calc = %5.2f\n",
               table[i].val,
               table[i].vsum,
               table[i].vsqr,
               table[i].vcalc);
            }
      if (fclose(fd))
            error(1);
      }

void error(short err_no)
      {
      printf("\n\7GRIND ERROR: %s\n",errmsg[err_no]);
      exit(err_no);
      }
/***/
```

Figure 5 — SINES.C

```
/*
Program:  Sines

Version:  1.00
Date:     29-Nov-1988
Language: ANSI C

Computes all of the sines of the angles between 0 and 360 degrees.
Developed by Scott Robert Ladd. This program is public domain.
*/

#define pi2rad 57.29577951

/* prototypes */
void main(void);
double fact(double);
double power(double, double);

void main()
    {
    double angle, radians, sine, worksine, temp, k;

    for (angle = 0.0; angle <= 360.0; angle += 1.0)
        {
        radians = angle / pi2rad;
        k = 0.0;
        worksine = 0.0;

        do  {
            sine = worksine;
            temp = (2.0 * k) + 1.0;
            worksine += (power(-1.0,k) / fact(temp)) *
                        power(radians,temp);
            k += 1.0;
            }
        while (sine != worksine);
        }
    }

/* Note: this function is designed for speed; it ONLY works when n is inte-
gral */
double fact(double n)
    {
    double res;

    res = 1.0;

    while (n > 0.0)
        {
        res *= n;
        n -= 1.0;
        }

    return res;
    }

/* Note: this function is designed for speed; it ONLY works when
         p is integral */
double power(double n, double p)
    {
    double res;

    res = 1.0;

    while (p > 0.0)
        {
        res *= n;
        p -= 1.0;
        }

    return res;
    }

/***/
```

C. This compiler would also be a good choice in environments where code must run on different architectures.

Power C is an excellent performer at an unbelievable price. MIX has finally worked the bugs out of their product, making it one of the best values going. I recommend this compiler to students and beginners. For number crunchers on a budget, it performs nearly as well as WATCOM C.

However, with all those comments aside, one compiler stands out in my mind as an outstanding value. Zortech C has continued the reputation it earned as Datalight C, by providing excellent code quality and compile speed for a small amount of money.

Zortech appeared in the benchmark summary charts 15 out of 18 times; no other compiler matches this performance. Many programmers interested in the quality of the code they put out have chosen this compiler. Those programmers looking to the future may want to purchase the Zortech C++ package which includes this C compiler. While you use their powerful compiler, you can also learn about objects.

The Future of C

By this time next year, some vendors will have upgraded their compilers from C to C++ (or some other object-oriented variant of C). Zortech's C++ product is already on the market. Like the move to structured programming in the seventies, object-oriented programming will forever change how we write programs. In the issues ahead, I'll look at some of the reasons C is the basis for the languages of the 1990s.

Next Issue

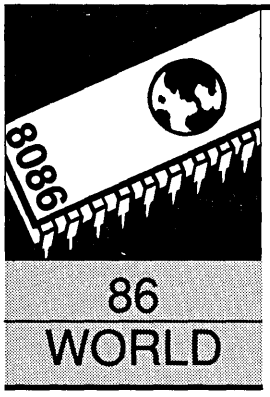
In the May/June 1989 issue, I'll talk about the use of some of the tools and utilities commonly associated with C. I'll also discuss some of the differences between the ANSI and K&R standards. Some are more significant than you might think.

Several of the manufacturers are readying major upgrades as I write this (the last week of November). So I'll talk about these.

Micro Assembly

On Memorial Day weekend 1989 (May 27-29), we're holding a Micro Assembly here in Denver, Colorado, at the Clarion Hotel (Airport). Registration costs $35 before the convention, and $50 at the door. Get in touch if you're interested.

♦ ♦ ♦



Screen Fonts For Windows & Gem

Unpacking Bags & Bits

By Laine Stump
PC Tech
907 North 6th Street
Lake City, MN 55041

Wow, Laine's body may be in Minnesota, but his mind certainly hasn't cleared customs. You'll hear more as he unpacks. Meanwhile, on to the body of his column.

So, I'm fresh off the boat, see? Well, off the plane anyway — you know, newly arrived in the country, ready to tackle some hot high tech. I've been dreaming of this for months. Yeah, get back to good ol' PC Tech and maybe do something new and exciting. Like write a multitasking kernel for a 34010 ... a video driver for a high res board ... a ROM BIOS for a 386. Maybe implement LIM 4.0 for a new version of the 16 Megger.

Anyway, I'm walking into the office, right? First stop after the airport and a night on the floor at La Guardia. Slept across the carpet from an old guy named "Jackson." He had two sweet rolls and a can of Blitz for breakfast and said someone was coming to pick him up.

First chance to relax since sharing overnight baggage watch duty with two Iranian senior citizens who came in on the same flight. First real moment of non-anxiety since being approached by a street dude asking me, "What kinda stuff you got there, man? Got enough for me?" (This happened while I was taking out my contact lenses, sitting in the lounge chair right across from the Piedmont check-in counter.)

Where was I? Oh, yeah. So, I'm walking into the office, okay? Well, "Earl," I ask, "What am I gonna work on? Whaddya got planned for me?"

So Earl says, "Well, I'm finishing the software for this new 34010 mono board with Group 4 FAX image decompression built in and ..." ("Oh yeah!" I'm thinking. "This is gonna be hot!!") "... and I'm still using the 9 x 15 font from the 736 x 1024 CPT monitor, but this new 2048 x 1536 Moniterm needs something larger, like about 12 x 23."

Whaatttt?? I just sold my X24/Rabbit Portable for plane fare! I've left my friends and life of sun and leisure! I've come 9,000 miles to click dots on and off with a mouse?!?!?

"I don't want you to waste your time clicking dots on and off with a mouse, though." Earl continues. "What I really want is a program that converts between Windows and Gem raster fonts, and our own internal 34010 format."

"Oh."

So, armed with the Windows Developer's Kit, the Gem Programmer's Toolkit, Zortech C++, and my favorite coffee cup (you must see it to appreciate it), I set off on my week long quest to understand binary images, discover new storage methods, to boldly figure out what the hell is going on.

So Where Was I?

Oh yeah... fonts. That's it! Well, see, it turns out that there are hundreds of fonts in the public domain, if you just know where to find them. Don't ask me; I don't know where to find them.

Anyway, knowing that there's lots of free fonts out there (seriously!) should give you the desire to read on. Please?

Here at PC Tech, we got several fonts included with "GFX Fonts," a font display library used with the GFX Graphics Library from a company called C Source. I can't tell you much about C Source or GFX Fonts, since I only wanted to translate their fonts into another format.

All I can tell you is that their font editor failed to run on a Hercules card (although it ran fine on an EGA). And their font files contained deviations from the Gem standard they were supposed to follow. But that's just first impressions, so ignore it, huh? Besides, it includes all the source code, so you can change it if you want.

Anyway, there are all kinds of fonts around. Balloon fonts, cartoon fonts, Olde Englishe Fonts. We even put a Cyrillic font into ROM on one of the 34010 boards just for fun (we're trying to land a contract with the KGB, but don't tell anyone).

If people just had enough information to use these fonts ... (Hah! I knew I'd squeeze a topic out of this somewhere!).

Figure 1 — Typical Simple Font Structure

```
typedef unsigned char[HEIGHT] character;
typedef character[256] font;

/***/
```

Figure 2 — Unpacked Linear Format

BYTE	bit 7	6	5	4	3	2	1	0
0	1A	1A	1A	1A	1A	1A		
1	1B	1B	1B	1B	1B	1B		
2	1C	1C	1C	1C	1C	1C		
3	1D	1D	1D	1D	1D	1D		
4	1E	1E	1E	1E	1E	1E		
5	2A	2A	2A	2A	2A	2A		
6	2B	2B	2B	2B	2B	2B		
7	2C	2C	2C	2C	2C	2C		
8	2D	2D	2D	2D	2D	2D		
9	2E	2E	2E	2E	2E	2E		

Figure 3 — Packed Linear Format

BYTE	bit 7	6	5	4	3	2	1	0
0	1A	1A	1A	1A	1A	1A	1B	1B
1	1B	1B	1B	1B	1C	1C	1C	1C
2	1C	1C	1D	1D	1D	1D	1D	1D
3	1E	1E	1E	1E	1E	1E	2A	2A
4	2A	2A	2A	2A	2B	2B	2B	2B
5	2B	2B	2C	2C	2C	2C	2C	2C
6	2D	2D	2D	2D	2D	2D	2E	2E
7	2E	2E	2E	2E	2E	2E		

Abstract

The programs I ended up with are too special-purpose (and too ugly) to put on public display in their entirety. But documentation and explanation of the font files themselves are worth talking about. So I'll explain the format of Windows raster font files and Gem font files, and give you a couple programming examples so you can use them.

Then, as an example of "something completely different," I will briefly describe PC Tech's own "PCT" font file, newly created by a committee of 1.5. (I asked Earl's opinion now and then, but mostly ignored what he said.)

First I'll discuss a few different possible font formats.

How To Represent A Character

When asked to come up with a data structure for a graphics representation of a character set (a font), most people will whack out the definition in Figure 1. This is fine, if your characters are all equal width, and this width is 8 or less. What if your font were 9 bits wide? Do you then switch from chars to ints?

You could, if you wanted to waste nearly 50% of your storage. This would only mean a loss of 256*Height bytes (3840 if your characters are 15 high). That's acceptable if you only use one font, but some graphics applications use more than 10 fonts at a time. Quite a few of those fonts are much larger than the font we're talking about.

Some other time I'll talk about the pros and cons of different font storage formats. For now let's just look at some (but I'm sure not all) of the different ways people have come up with to store font bitmaps.

Figures 2 through 5 give a graphic idea of the font formats, using a two character 6 x 6 font as an example.

Unpacked Linear

This is the format of the definition in Figure 1 (see Figure 2). The system stores all the scan lines of a character together. Each scan line starts on a byte boundary following the previous scan line of the character. Therefore, the bits for line y of character n start at the beginning of byte:

```
((n * Height) + y) * CharWidthInBytes
```

This format eliminates all those nasty bit shifts required for extracting characters in packed fonts (we'll discuss this momentarily), but wastes an average of NumChars*Height/2 bytes per font. This can add up.

Also, it takes a multiply to find the beginning of the character. While an integer multiply takes only around 30-40 clocks (on a 286), all these multiplies can add up (especially on an 8088, where a multiply is > 100 cycles). What if the font is large enough that the bitmap is greater than 64K? (Not an unreasonable idea.) However, a long multiply takes multiple instructions.

Packed Linear

In this format, the system stores the bits for a character together. *Very* together (see Figure 3). The bits for scan

Figure 4 — Unpacked Rectangular Format

BYTE	bit 7	6	5	4	3	2	1	0
0	1A	1A	1A	1A	1A	1A		
1	2A	2A	2A	2A	2A	2A		
2	1B	1B	1B	1B	1B	1B		
3	2B	2B	2B	2B	2B	2B		
4	1C	1C	1C	1C	1C	1C		
5	2C	2C	2C	2C	2C	2C		
6	1D	1D	1D	1D	1D	1D		
7	2D	2D	2D	2D	2D	2D		
8	1E	1E	1E	1E	1E	1E		
9	2E	2E	2E	2E	2E	2E		

```
To get a character:

  char* get = bitmapstart + n << 1;   /* location of line 1 */
  for (int ct = 0; ct < Height; ct++)
    {
    Put 16 bits from *get on screen;
    get += 512;
    }
/***/
```

Figure 5 — Packed Rectangular Format

BYTE	bit 7	6	5	4	3	2	1	0
0	1A	1A	1A	1A	1A	1A	2A	2A
1	2A	2A	2A	2A				
2	1B	1B	1B	1B	1B	1B	2B	2B
3	2B	2B	2B	2B				
4	1C	1C	1C	1C	1C	1C	2C	2C
5	2C	2C	2C	2C				
6	1D	1D	1D	1D	1D	1D	2D	2D
7	2D	2D	2D	2D				
8	1E	1E	1E	1E	1E	1E	2E	2E
9	2E	2E	2E	2E				

```
To get a character:

   int RowWidth = (NumChars * CharWidth);
   int ByteOffsetInRow = (n * CharWidth)/8;
   int BitOffsetInByte = (remainder of above);
   char* get = BitMapStart + BytOffsetInRow;
   for (int ct = 0; ct < Height; ct++)
     {
     put *get shifted left by BitOffsetInByte on screen;
     get += RowWidth;
     }
/***/
```

Figure 6 — Windows FNT Font File Header Definition

```
typedef struct
  {
  unsigned int  dfVersion;          /* version # of font file format (256) */
  unsigned long dfSize;             /* file size in bytes */
  char          dfCopyright[60];    /* vendor copyright notice */
  unsigned int  dfType;             /* type of font - bit 0 on if vector */
  unsigned int  dfPoints;           /* point size at which char looks best */
  unsigned int  dfVertRes;          /* dpi at which font was digitized */
  unsigned int  dfHorizRes;         /* see above */
  unsigned int  dfAscent;           /* no. of rows from top to baseline */
  unsigned int  dfInternalLeading;  /* Leading included in chars */
  unsigned int  dfExternalLeading;  /* Recommended leading outside chars */
  unsigned char dfItalic;           /* 1 if italic font, else 0 */
  unsigned char dfUnderline;        /* 1 if underlined, else 0 */
  unsigned char dfStrikeOut;        /* 1 if strikeout, else 0 */
  unsigned int  dfWeight;           /* stroke weight on 1-1000 scale */
  unsigned char dfCharSet;          /* char set standard, FF for IBM PC */
  unsigned int  dfPixWidth;         /* width of char cell, or 0 if variable */
  unsigned int  dfPixHeight;        /* height of char cells & bitmap */
  unsigned char dfPitchAndFamily;   /* low bit set if variable pitch */
  unsigned int  dfAvgWidth;         /* width of character 'X' */
  unsigned int  dfMaxWidth;         /* widest character in var. fonts */
  unsigned char dfFirstChar;        /* ASCII of first char in bitmap */
  unsigned char dfLastChar;         /* ASCII of last char in bitmap */
  unsigned char dfDefaultChar;      /* char to use if requested not there */
  unsigned char dfBreakChar;        /* char that defines word breaks */
  unsigned int  dfWidthBytes;       /* # of bytes in one row of bitmap */
  unsigned long dfDevice;           /* disregard */
  unsigned long dfFace;             /* offset in file of font name */
  unsigned long dfBitsPointer;      /* used internaly by Windows */
  unsigned long dfBitsOffset;       /* offset of bitmap in file */
  }  WinFontHeader;

/***/
```

Figure 7 — ReadWinFont

```
/*--- Read a Windows FNT file into global arrays ----------*/
WinFontHeader WinFont;
char bitmap[32767];   /* watch out you don't read anything bigger */
int  chofstable[256+1];
char facename[17];

void ReadWinFont(char filename[])
  { /* note: for brevity I have removed all file error checking */
  FILE* fin = fopen(filename,"rb");
  fread(&WinFont, sizeof(WinFont), 1, fin);
  if (WinFont.dfPixWidth == 0) /* indicates a variable pitch font */
    /* read character offset table */
    fread(&chofstable,
          (WinFont.dfLastChar-WinFont.dfFirstChar+2)*2, 1, fin);
  fseek(fin, WinFont.dfBitsOffset, SEEK_SET);
  int bitmapsize =  WinFont.dfWidthBytes * WinFont.dfPixHeight;
  fread(&bitmap, bitmapsize, 1, fin);
  fseek(fin, WinFont.dfFace, SEEK_SET);
  fread(&facename, 17, 1, fin);
  fclose(fin);
  } /* ReadWinFont */

/***/
```

line y of a character tack right onto the end of the bits for scan line y-1, regardless of byte boundaries.

The arithmetic for figuring out where a scan line is located gets more complicated, if you're still talking in bytes. Addressing according to bits simplifies things a little. However, except for the 34010, most processors have a tough time dealing with bits.

The bits for line y of character n start at byte:

```
(((n*Height) + y) * CharWidth)/8
```

and bit (7 - remainder) where remainder is the remainder of the division by 8. It's (7 - remainder) instead of just (remainder) because most PC graphics adaptors consider the "left" bit of a byte to be bit 7 (although bit 0 is the left on 34010s).

Packed Linear eliminates the waste seen in Unpacked Linear format, but makes for all kinds of unnecessarily complex bitshifts just to get a character out of the bitmap. The amount of shift is not only different for every character, but for every scan line of every character.

And, just like Unpacked Linear, you still have to do a multiply to find the beginning of a character.

Unpacked Rectangular

In this format, the system stores the first lines of all characters, followed by the second lines of all characters, the third lines, etc. Each piece of a character starts on a byte boundary. In this format, line y of character n is at

```
((n*CharWidthInBytes) +
(y*BitMapWidth))
```

where BitMapWidth is CharWidthInBytes*NoCharsInSet. I know this sounds silly, but there are reasons for it. Really.

The advantage is that you can successfully retrieve any character without doing any multiplies (remember how much time a multiply takes). For example, Figure 4 shows the algorithm to get the n^{th} character in an unpacked rectangular bitmap with character cells 16 bits wide.

Unfortunately, although fast, you waste just as much space in Unpacked Rectangular format as in Unpacked Linear Format.

Packed Rectangular

Windows and Gem use the packed rectangular format (see Figure 5). The system stores all the bits for a certain

row packed together (paying no attention to byte boundaries) followed by the bits for the next row of characters. The bits of scanline y of character n start somewhere in byte:

```
((y*BitMapWidth) + (n*CharWidth)/8)
```

where BitMapWidth is (CharWidth*NoCharsInSet)/8 rounded up to the next byte boundary (actually the next word boundary, for Windows and Gem).

Within that byte, you can find the bit offset of the start of the character by taking (7 - remainder) where remainder is the remainder of (n*CharWidth)/8. All bits from this bit to bit 0 are part of the character. So if remainder is 0, all the bits in the byte are part of the character (unless the character is less than 8 bits wide).

Now this looks screwy, doesn't it? It can't be as fast as Unpacked Rectangular format (you've got to shift the character data around to normalize it), but it is more compact. It only wastes an average of Height/2 bytes per font (Height*1.5 if each row rounds to a word boundary instead of a byte boundary).

Let's investigate how we would get a character out of this type of bitmap. The algorithm in Figure 5 will get all the lines of a single character, "n".

Realize that I'm simplifying things quite a bit when I say "get shifted left by BitOffsetInByte." I admit I haven't devoted enough time to this subject yet to come up with a detailed algorithm.

I wouldn't want to make you suffer through another bit-by-bit shift of the character onto the display. Heck, Larry already did that in the last issue. (If I was being threatened with a snowball down the throat of my X24 if I didn't show you something that worked, I would probably show you the same chunk of code.)

Back to the subject ... Well, this looks like the most speed inefficient storage method yet. Look at all those multiplies. At least the bit shift for all the rows of a character is the same.

Notice the first multiply is constant as long as you use the same font. You could do it once when you load the font and then forget about it. Both Gem and Windows include RowWidth as part of the font header record, so you never have to do it at all.

The second multiply is different for each character in the font. There are at most only 256 characters in any font, so we can easily build a "character offset" table when we load the font. Gem includes this table as part of the font file, and with Windows it's an optional addition to the file (used if the font has variable width characters).

The divide... well, everybody knows that a divide by 8 is nothing more than a shift right by 3.

So, by fooling around a bit, and by using 516 bytes more memory (514 for the character offset table and 2 for RowWidth), we have eliminated all multiplies from the operation of putting a character on screen.

I guess that's why the makers of the two most widely accepted graphics interfaces adopted the same format.

Figure 8 — Gem Font File Header Definition

```
typedef struct
  {
  unsigned int  gfID;           /* Font type ID  */
  unsigned int  gfPointSize;    /* size in points, don't use */
  char gfName[32];              /* name of font */
  unsigned int  gfFirstChar;    /* ASCII of first char in bitmap */
  unsigned int  gfLastChar;     /* ASCII of last char in bitmap */
  unsigned int  gfTopLine;      /* distance from baseline to top */
  unsigned int  gfAscentLine;   /* height of capital letter*/
  unsigned int  gfHalfLine;     /* height of small letter */
  unsigned int  gfDescentLine;  /* depth of descenders */
  unsigned int  gfBottomLine;   /* distance from baseline to bottom */
  unsigned int  gfMaxWidth;     /* widest char in font */
  unsigned int  gfMaxCellWidth; /* widest char cell in font */
  unsigned int  gfLeftOffset;   /* used for tilted fonts */
  unsigned int  gfRightOffset;  /* see above */
  unsigned int  gfThickening;   /* # of pixels to thicken for bold */
  unsigned int  gfUndlSize;     /* thickness of line for underline */
  unsigned int  gfLightMask;    /* BitMask for lightening, (5555h) */
  unsigned int  gfSkewMask;     /* ???? (5555h) */
  unsigned int  gfFlags;        /* bit 3 set if monospaced */
  unsigned long gfHorOfsTable;  /* ???? don't use */
  unsigned long gfCharOfsTable; /* offset of char offset table in file */
  unsigned long gfBitsOffset;   /* offset of bitmap in file */
  unsigned int  gfFormWidth;    /* # of bytes in one row of bitmap */
  unsigned int  gfFormHeight;   /* height of chars and bitmap */
  unsigned long gfNextFont;     /* ptr to next font, used internal to Gem */
  }  GemFontHeader;
/***/
```

Figure 9 — ReadGemFont

```
/*--- Read a GEM font file into global arrays ----------*/
GemFontHeader GemFont;
char bitmap[32767];   /* watch out you don't read anything bigger */
int  chofstable[256+1];

void ReadGemFont(char filename[])
  {
  FILE* fin = fopen(filename,"rb");
  /* read header info */
  fread(&GemFont, sizeof(GemFont), 1, fin);
  /* read horizontal offset table */
  fseek(fin, GemFont.gfCharOfsTable, SEEK_SET);
  fread(&chofstable,
        (GemFont.gfLastChar-GemFont.gfFirstChar+2)*2, 1, fin);
  /* read bitmap */
  int bitmapsize =  GemFont.gfFormWidth * GemFont.gfFormHeight;
  fseek(fin, GemFont.gfBitsOffset, SEEK_SET);
  fread(&bitmap, bitmapsize, 1, fin);
  fclose(fin);
  } /* ReadGemFont */
/***/
```

Windows Fonts

The Windows font files I'm talking about here are the ones that have an FNT extension, not the ones on the Windows runtime disk which have an FON extension. You produce FON files by combining several FNT files with the Windows Resource Compiler, which shuffles the information around. The FON files contain the same information, they just don't follow the format given in the Developer's Kit.

Also, I'm only talking about raster fonts — those represented in the file by bitmaps indicating which dots should be on and off within a character. Vector fonts (those represented by lists of vec-

Figure 10 — Font Conversion

```
/*-- Fill in a Windows Font Header from a Gem Font Header --*/

void ConvertG2W( )
  { /* assumes all data is in previously defined global arrays */
  int numchars = (GemFont.gfLastChar - GemFont.gfFirstChar + 1);

  /* Translate the header information */
  WinFont.dfVersion = 256;
  strnset(WinFont.dfCopyright,'c',60);
  WinFont.dfType = 0;
  WinFont.dfPoints = GemFont.gfFormHeight;
  WinFont.dfVertRes = 72;
  WinFont.dfHorizRes = 72;
  WinFont.dfAscent = GemFont.gfTopLine;
  if (WinFont.dfAscent == 0)
    WinFont.dfAscent = GemFont.gfAscentLine;
  WinFont.dfInternalLeading = 0;
  WinFont.dfExternalLeading = 0;
  WinFont.dfItalic = 0;
  WinFont.dfUnderline = 0;
  WinFont.dfStrikeOut = 0;
  WinFont.dfWeight = 200;
  WinFont.dfCharSet = 255;
  WinFont.dfPixWidth = 0;
  WinFont.dfPixHeight = GemFont.gfFormHeight;
  WinFont.dfPitchAndFamily = 1; /* low bits indicates proportional */
  WinFont.dfAvgWidth = chofstable['X'+1-GemFont.gfFirstChar]
                     - chofstable['X'-GemFont.gfFirstChar];
  WinFont.dfMaxWidth = MaxWidthInTable(chofstable, numchars);
  WinFont.dfFirstChar = GemFont.gfFirstChar;
  WinFont.dfLastChar = GemFont.gfLastChar;
  WinFont.dfDefaultChar = 0x60 - GemFont.gfFirstChar;
  WinFont.dfBreakChar = 32 - GemFont.gfFirstChar;
  WinFont.dfWidthBytes = GemFont.gfFormWidth;
  WinFont.dfDevice = 0;
  WinFont.dfBitsPointer = 0;
  WinFont.dfBitsOffset = (((sizeof(WinFont) && 1) == 1) ?
                          sizeof(WinFont)+1 : sizeof(WinFont))
                           + ((numchars+1)*2);
  WinFont.dfFace = WinFont.dfBitsOffset
                 + (WinFont.dfWidthBytes * WinFont.dfPixHeight);
  /* NOTE: The following is out of sequence because it depends on
           other computed values in the header */
  strncpy(facename,GemFont.gfName,17);
  facename[16] = 0;
  WinFont.dfSize = WinFont.dfFace + strlen(facename)+1;
  } /* ConvertG2W */

/*-- Fill in a Gem Font Header from a Windows Font Header --*/

void ConvertW2G( )
  { /* assumes all data is in previously defined global arrays */
  int numchars = (WinFont.dfLastChar - WinFont.dfFirstChar + 1);

  /* Translate the header information */
  GemFont.gfID = 2;;
  GemFont.gfPointSize = WinFont.dfPoints;
  strcpy(GemFont.gfName,facename);
  GemFont.gfFirstChar = WinFont.dfFirstChar;
  GemFont.gfLastChar = WinFont.dfLastChar;
  GemFont.gfTopLine = WinFont.dfAscent;
  GemFont.gfAscentLine = WinFont.dfAscent;
  GemFont.gfHalfLine = WinFont.dfAscent/2+1;
  GemFont.gfDescentLine = WinFont.dfPixHeight-WinFont.dfAscent-1;
  GemFont.gfBottomLine = WinFont.dfPixHeight-WinFont.dfAscent;
  GemFont.gfMaxWidth = WinFont.dfMaxWidth;
  GemFont.gfMaxCellWidth = WinFont.dfMaxWidth;
  GemFont.gfLeftOffset = 0;
  GemFont.gfRightOffset = 0;
  GemFont.gfThickening = 1;
  GemFont.gfUndlSize = 1;
  GemFont.gfLightMask = 0x5555;
  GemFont.gfSkewMask = 0x5555;
  GemFont.gfFlags = 0;
  GemFont.gfHorOfsTable = 0;
  GemFont.gfCharOfsTable = ((sizeof(GemFont) && 1) == 1) ?
                           sizeof(GemFont)+1 : sizeof(GemFont);
```

continued on next page

tors to draw) are a whole 'nother animal.

Yet another qualifier... Although I have used this method to look at the FNT version of files as recent as Windows 386 fonts, my Developer's Kit is for Windows 1.04. Microsoft has done "something" with its font format in newer versions of Windows, so this information may be out of date. Or not.

At any rate, you won't be able to take this information and use it directly to modify the fonts you got with PageMaker, or the fonts you bought from Bitstream.

If you have the Window's Developer's Kit, it has a passable font editor that runs under Windows. You can also use this information to create FNT files to compile into FON files yourself. (As long as the FNT file hasn't followed the FON file in changing format.)

See Figure 6 for a listing of the Windows font file header record. Figure 7 lists a C++ function to read a Windows font file into a global header record, a global bitmap, and a global character width table, if there is a width table.

In ReadWinFont, I first read the header record — always located at the beginning of an FNT file. Next, for variable width fonts (as indicated by dfPixWidth being 0), I read the character offset table (it always follows the header).

The dfBitsOffset field of the header record points to the offset in the file of the bitmap. (Windows uses dfBitsPointer after loading the font into memory.) My next step is to seek to that position in the file, then read dfWidthBytes (the width of each row) * dfPixHeight bytes into bitmap.

Finally, I seek to the offset of the font name (stored in the header in dfFace) and read it into facename. Then I close the file (I wasn't born in a barn, you know).

Gem Fonts

Yep. These are the same fonts you see on the screen when you run Ventura. Every stinkin' one of them. I don't know if it's legal, strictly speaking. It is certainly physically possible to use those slick-looking fonts, like Times and Modified Boston Slang and Watusi Bold and all the others, in your own applications.

Figure 8 shows the fields of the header record of a Gem font file. Just like the Windows font header, there's a lot of stuff in a Gem font header that most of us don't care about. Maybe people writing Venturas and

PageMakers care about internal leading and external leading. I just want to print something on the screen and don't care about one dot this way or that, usually.

Figure 9 shows a C++ function to read a Gem font into global memory buffers similar to the function for Windows fonts. Gem uses different field names (dfWidthBytes is gfFormWidth). Also, the character offset table is not in a fixed place in Gem fonts, while the font name is.

C++ Function??

An aside here. When I say "a C++ function," I mean "a C function that I compiled with Zortech C++, so it probably won't compile under normal C." I haven't gotten into C++ yet, so these examples don't look too "plussy."

Don't think of them as examples of C++ programming. Think of them as examples of dealing with fonts in a high level language.

Partial Character Sets

Both Windows and Gem support fonts with any number of characters up to 256. (It appears Gem will support up to 65,535, but I haven't tried it). The lowest character in the set is character dfFirstChar (gfFirstChar for Gem) and the highest character in the set is dfLastChar (gfLastChar).

To avoid wasting space on characters that aren't even in the font, the system stores the character "FirstChar" at the very first position in the bitmap, and the bitmap ends after LastChar.

Because of this, your character display function should always make sure the character you want to display lies somewhere in the range FirstChar – LastChar. If not, you should display DefaultChar (or nothing).

Don't forget that you find character "n" at position "n-FirstChar" in the bitmap.

From Windows To Gem And Back

The functions ConvertG2W() and ConvertW2G() in Figure 10 show how the fields of the two font types correlate. Use them if you want to read both font types, but write all your font processing routines to use a single font format. The bitmaps and character offset tables are identical for both fonts.

Speaking of multi-compatibility, the function IsWindowsFont() does a simple check on a font file and returns true if it fits the rules of a Windows font file. It only checks that the dfSize field equals the length of the file in bytes, but that's enough for mere mortals.

Figure 10 continued

```
  GemFont.gfBitsOffset = GemFont.gfCharOfsTable + ((numchars+1)*2);
  GemFont.gfFormWidth  = WinFont.dfWidthBytes;
  GemFont.gfFormHeight = WinFont.dfPixHeight;
  GemFont.gfNextFont = 0;
  if (WinFont.dfPixWidth != 0) /* if fixed pitch in Windows */
    MakeChOfsTable(chofstable, WinFont.dfPixWidth, numchars);
  } /* ConvertW2G */

/*-- determine size of a file --*/

long fsize(FILE* fp)
  { /* this was written because there is not an ANSI */
    /* standard fsize(). Every compiler has a different one */
    /* This one uses only ANSI routines */
  long tmpsize;
  long pos = ftell(fp);
  fseek(fp, 0L, SEEK_END);
  tmpsize = ftell(fp);
  fseek(fp, pos, SEEK_SET);
  return(tmpsize);
  } /* fsize */

/*-- Determine if file is a Windows FNT file --*/

int IsWindowsFont(char filename[])
  {
  int temp;
  long size;

  FILE* fin = fopen(filename,"rb");
  fread(&temp,2,1,fin);  /* read past version */
  fread(&size,4,1,fin);
  /* read long word in byte 2-5 and see if = length of file */
  temp = (fsize(fin) == size);
  fclose (fin);
  return(temp);
  } /* IsWindowsFont */

/*-- Construct a character offset table --*/

void MakeChOfsTable(int ChOfsTable[], int Width, int NumChars)
  {
  for (int ct = 0; ct <= NumChars; ct++)
    ChOfsTable[ct] = ct*Width;
  } /* MakeChOfsTable */
/***/
```

As long as you're decently careful, you can mix the use of Windows and Gem fonts transparently. Just call IsWindowsFont() on a file before you open it, then read it into the proper buffers and do the conversions. You decide which format to use internally. I would use the Windows format, but that's just because I figured it out two days before Gem, so I'm more familiar with it.

Even if you use Windows format internally, you should call MakeChOfsTable() for fixed width fonts in order to find characters faster. Since Gem requires a character offset table, even for fixed width fonts, you have to support character offset tables to be able to display Gem fonts, anyway.

Character Offset Table

Speaking of that, I almost forgot to explain the format of the Windows and Gem character offset tables.

The character offset table has (dfLastChar - dfFirstChar) + 2 word entries which give the offset in bits of each character from the left side of the bitmap rectangle. You can find the width of a character n by subtracting charofstable[n+1] from charofstable[n]. Hence the extra entry at the end, used to determine the width of the last character.

As with the bitmap, the offset for character n is at charofstable[n-FirstChar].

Putting It All Together

Okay, so I'm too proud to flick bits, am I? I think it makes me look stupid and old-world. I would just feel too guilty leaving you with absolutely no way of looking at those fonts you ripped off from your buddy's copy of Ventura.

The C++ function in Figure 11 writes a character from a font onto the screen

Figure 11 — DisplayChar()

```
/*-- Crudely display a character of a font --*/

void DisplayChar(unsigned char ch, char* bitmap, int Height,
                 int BitMapWidth, int chofstable[])
  {
  printf("char: %02x, %d bits wide\n", ch, chofstable[ch+1]-chofstable[ch]);
  int const bitoffset = (chofstable[ch] & 7);
  bitmap += chofstable[ch] >> 3;   /* byte offset of start of char */

  for (int r = 0; r < Height; r++)
    {
    char* ptr = bitmap;
    char workbyte = (*ptr++) << bitoffset;
    int bcount = (7 - bitoffset);
    for (int bitstoprint = chofstable[ch+1] - chofstable[ch];
         bitstoprint > 0; bitstoprint--)
      {
      putchar(workbyte & 0x80 ? '*':'.');
      workbyte <<= 1;
      if (!bcount--)
        {
        workbyte = *ptr++;
        bcount = 7;
        }
      } /* for b */
    putchar('\n');
    bitmap += BitMapWidth;
    } /* for r */
  } /* DisplayChar */

/*-------------------------------------------------------------------*/
main( int argc, char* argv[ ] )
{ /* print all chars in Gem Font file given on command line */
  ReadGemFont(argv[1]);
  for (int ch = 0; ch <= GemFont.gfLastChar - GemFont.gfFirstChar; ch++)
    DisplayChar(ch, bitmap, GemFont.gfFormHeight,
                GemFont.gfFormWidth, chofstable);
  printf("Done.\n");}
/***/
```

as a bunch of asterisks. This doesn't do much in practical terms, but at least it verifies that you understand what's going on. Two Brownie Points to the first one of you who figures out how to integrate this into Larry's printer graphics library in issue 45.

If any of you have an algorithm that's more efficient, I would be truly honored to take a peek at it. If I get any good ones, maybe I'll print them.

PCT Font Format?

Oh, yeah. I promised to tell you about our own font format. The Unpacked Rectangular format now stores the PC Tech font format. This is because we use it only for the text mode display on the 34010.

With our 34010s in text mode, there's loads of extra memory sitting around. (Even the smallest PC Tech 34010 has 256K of nondisplay memory and 256K of video memory.) So space isn't a concern, but the fastest performance possible is.

In PC Tech format, as with any other semi-intelligent 34010 font format, the systems stores all the bits within a byte in reverse order from the Windows and Gem fonts. That's because, as mentioned before, the 34010 is a bit addressed processor, so the left bit is bit 0.

The header for PCT format fonts contains more or less the same information that's in the Gem and Windows headers. Just less of it. The actual format doesn't matter, since you'll probably never see a file in this format anyway (unless you get a 34010 board). Mainly I just mentioned it to show you that no single format is best for all situations.

Oh, and by the way, the entire internal loop of the algorithm I gave for packed rectangular form is a single instruction on the 34010. Including the bit shifts and masks.

You wonder what's kept Earl glued to his screen for the last year ...

Next Time

... the magic of the 34010 PIXBLT instruction and other wonders of the electronic world.

P.S. Remember — no matter what they tell you, Raisin Bran is an addictive narcotic. Please take it only as your doctor advises.

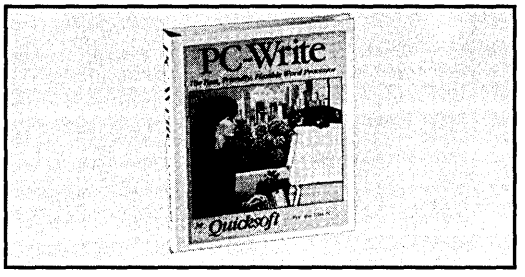
♦ ♦ ♦

PC-File:dB & PC-Write 3.0

Definitely New & Definitely Improved

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

For those of you tuning in late, Tony Barcellos is software librarian and newsletter editor for the Sacramento PC Users Group. In his spare time he teaches math and writes study guides, but it's his knowledge of shareware that has made him famous among software authors and has brought him to the pages of Micro C.



A shareware columnist has no excuse for writer's block. There's entirely too much to pick from in this corner of the computer industry. The usual problem is deciding what to write about.

A couple of giants in the shareware business made my job easier this month. ButtonWare released PC-File:dB and Quicksoft shipped version 3.0 of PC-Write. Both programs instantly jumped to the top of my topic list.

Single File

There are plenty of competing database products out there, even if one forgets the high-end products for a moment. (By "high-end" we mean in terms of price, of course.) Despite worthy competitors like ExpressWare's File Express, however, ButtonWare's PC-File continues to attract the most attention.

Rightfully so. Jim Button's famous database program was half of the dynamic duo (PC-File and PC-Talk) that established the credibility of user-supported software. While PC-Talk has faded, PC-File is still going strong.

So what has Jim Button done for us lately? Although his company's offerings have expanded over the years to include several other programs (see the May-June 1988 issue of *Micro C*), PC-File remains the centerpiece. Still stressing ease-of-use via menus, now enhanced by context-sensitive help screens, PC-File calls its new incarnation PC-File:dB. Guess what the "dB" stands for.

"By making PC-File dBASE compatible," says Jim Button, "I can now offer dBASE users an easy-to-use interface for their current data."

Since ease-of-use is conspicuously missing from the vocabularies of most dBASE users, PC-File:dB goes head to head with Ashton-Tate's dBASE III Plus. (Now that dBASE IV is finally shipping, Ashton-Tate can claim to having addressed the ease-of-use problem. However, since the file format remains the same, PC-File:dB should be adaptable to dBASE IV as well.)

Recent PC-File enhancements are still there, of course. PC-File:dB's predecessor, PC-File+, sported integrated graphics. Hence you can whip up pie charts or bar charts with your ButtonWare database.

The bar charts come in several different flavors, or you may choose line or scatter plots instead. The pies explode (for your enjoyment, of course) while bar and line charts offer razzle-dazzle logarithmic scales. This is heavy-duty stuff.

Keyboard macros, long a PC-File feature, can be created by example within PC-File:dB. Just turn on the macro feature and let PC-File record your keystrokes. With a little ambition, you can set up a macro that closes off data entry, sorts the data, prints a report (perhaps calling user-created report formats from disk), and puts you back where you began.

Not only can you save report formats (which PC-File has supported since much earlier versions), but now you can create them by painting the layout of your report on-screen.

The PC-File:dB laundry list of features is impressive. PC-File:dB has five data types: numeric, character, logical, date, and memo.

ButtonWare provides a few details about the program: Maximum number of records per database: 1 billion; maximum record length: 4000 characters; maximum field length: 254 (with exceptions for special fields like date fields, which have prescribed lengths); maximum memo field length: 5000 characters (even greater than the standard record length); maximum number of fields per database: 70; maxi-

mum number of calculated fields per database: 70; maximum number of relational fields per database: 70.

That should handle the Christmas card list.

The power isn't free, however. PC-File:dB requires 416K of RAM and a pair of 720K floppy drives. (A pair of 360Ks won't do it anymore.) As you might guess, a hard disk would be even better.

The other price is monetary, of course. Here ButtonWare continues to hold the line, offering a fully registered copy of PC-File:dB for only $89.95 plus $5 shipping and handling. This program conforms to the dBASE file standard — not the dBASE price standard.

PC-File:dB comes on three 360K diskettes. All three disks contain archive files (in ButtonWare's own special archival format). There's a Program disk, Utilities disk, and Documentation disk. The latter contains the entire user's manual, minus a few illustrations. The registered version comes with two nicely printed manuals — a standard user's guide and a special beginner's manual. (The documentation disk does not include the beginner's manual.)

Manuals are perfect-bound, attractively printed, and come in an amusing slipcase shaped like a giant B. The slipcase doesn't seem likely to hold up under much wear and tear, though the program's heritage argues that *it* will.

PC-File:dB
$89.95 + $5 shipping
ButtonWare, Inc.
P.O. Box 96058
Bellevue, WA 98009-4469
(800) JBUTTON, Toll-free order line

Write On!

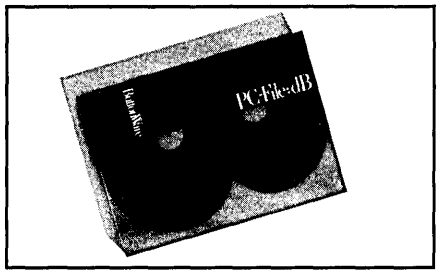
The other entrant in the geriatric shareware sweepstakes is PC-Write from Quicksoft. Now at version 3.0, PC-Write hardly seems to be suffering from old age. Big and beefy, PC-Write now comes on three diskettes with a complete user's manual occupying one of the disks.

What has Bob Wallace done to his warhorse word processor? After more than a year of teasing about what PC-Write 3.0 would offer, Quicksoft reports it has added over 500 features.

Here are a few of the major improvements:

- The old 60K file limit is gone. PC-Write 3.0 will use all available memory for files.
- It supports and displays multiple columns.
- You can grab rectangles of text and delete, copy, or move them.
- It has menu-selected fonts and special effects.
- New quick keys bypass menu operations for greater speed in switching files and printing.
- Bob's integrated the editing and printing modules. (And they remain in memory whenever possible.)

Besides the word processor, Quicksoft offers companion programs like the DCA utility ($29), which converts files



between DCA-RFT and PC-Write formats. Font Selector ($29) chooses and downloads soft fonts to your Hewlett-Packard LaserJet+ or compatible laser printer. Finally, there's the PageMaker Import Filter ($10), which saves PC-Write files in a format PageMaker understands. (It preserves much of PC-Write's formatting.)

"PC-Write 3.0 is a giant leap in capabilities," says Bob Wallace. He's right.

PC-Write 3.0
$89
Quicksoft, Inc.
219 First Avenue North, #224
Seattle, WA 98109
(800) 888-8088 or (206) 282-0452

Archival Arch Rivals

The continuing legal dispute between System Enhancement Associates (SEA) and PKWare over SEA's ARC utility and PKWare's rival software is a dismal thing to follow. As I mentioned in my last column, the details of the lawsuit's settlement became public despite the agreement of the two parties to refrain from commenting on it.

While PKWare's Phil Katz remains the beneficiary of sympathy from most of the user community, SEA's Thom Henderson has explained his grievances in detail. He says his lawsuit was necessary to protect the integrity of his ARC program. (I thank Michael Kaufman for sending along the latest archive of bulletin board messages on the topic.)

Henderson claims that Katz copied ARC's program code in creating PKARC and PKXARC, utilities that do all ARC's archiving functions (and then some) with major improvements in speed. Katz's defenders rebut Henderson by declaring that the compression algorithms in ARC and PKARC derive from the same public domain sources.

Henderson continues his argument by asserting that Katz would be in violation of copyright law even if he hadn't copied *any* of SEA's code (although Henderson is convinced that he did).

Henderson argues that Katz's admission of seeing SEA's program code for ARC establishes a presumption of infringement on SEA's copyright, even if Katz then created PKARC and PKXARC from scratch.

The original court settlement was in SEA's favor, while the court of public opinion seems to be coming down on PKWare's side. Whether Henderson's efforts to explain his position to users will shift the tide in his favor remains to be seen.

I'm sick of the entire business so I'll just wait for the tragicomedy to come to a conclusion. Somebody — anybody! — get the hook!

Editor's note: See the Letters Column in this issue for another point of view on the SEA vs PK controversy. And, stay tuned for more next issue.

Cards And Letters And Disks

My thanks to all of you who have contacted me. It's a struggle to keep up with it, let alone do justice to its volume. Besides the letters and notes, there are the stacks of disks and programs that I need to look over when I find the time. (Time? Anyone have the time?)

My forays into the stacks take on the aspects of an archaeological dig. Where's that program I wanted to write about that does graphics in Encapsulated PostScript format (no "jaggies" when you scale an illustration)?

How about that critically acclaimed spreadsheet package that was withdrawn from the regular commercial channels and relaunched as shareware? (It's around here somewhere, along with the nice cover note written by the author.) Then there's the program that purports to bring true 3-D CAD features to shareware for the first time. And the programs I talked about before keep getting updated.

There's no way I'll ever catch up. But the fun is in the trying. Keep sharing.

♦ ♦ ♦

PersonalWare

Shareware You'll Want To Register

By Dave Thompson

I received a small brown package the other day and inside I found a small black disk. Nothing more. On the label someone had scribbled "Personal Relationships, an Expert System for Adults Only."

I fired up my system, popped in the disk, and looked at the directory.

Aha, USEME.EXE.

Welcome to a new world of relationships. To take full advantage of this software you must:

1. Find someone compatible.

2. Find a bottle of bubbly and two glasses.

3. Turn up the heat.

4. Turn down the lights.

5. And hit ENTER.

I ran upstairs, grabbed Sandy, found the bottle of carbonated apple juice we keep for medicinal purposes, and headed back to the computer.

I hit ENTER.

Please enter your name.

DAVE

Please enter your partner's name.

SANDY

Please select the type of relationship you'd like to develop.

- *1. Non-touching, mostly platonic. (e.g. husband/wife)*
- *2. Good friends (cheek kisses and hugs on special occasions).*
- *3. Very Good Friends (e.g. quarterback/cheerleader).*
- *4. Consumate Roommates.*

I selected 1. We'd already practiced that one.

I'm sorry, that selection is not available.

I cranked up my courage and selected 2.

I'm sorry, that selection is not available.

I've never tackled football but after 20 years of matrimonial bliss, "Very Good Friends" sounded like something we could work toward. I selected 3.

I'm sorry, that selection is not available.

I bit my lip and selected 4.

Good choice.

From here on, I'll take the lead. Timing is critical so please do not begin any activity before I tell you. Continue the activity until I beep. At the beep, drop what you're doing and read this screen.

"Beep!"

First, get comfortable. Put on a quiet record, take off your shoes, loosen your tie or collar, and take a sip or two of the bubbly. The lights should be low. Remove any distractions.

I dumped the cat off my lap.

"Beep!"

Now sit together side by side. DAVE, reach over and grasp SANDY's hand. Gently.

Maintain that position. Do not speak.

"Beep!"

Now, DAVE, put your arm around SANDY. Notice the warmth, DAVE, that begins where your arm touches SANDY. Notice how that warmth travels back down your arm and spreads throughout your entire body. Notice especially how that warmth settles into your solar plexis.

I dumped the cat off my lap. Again.

"Beep!"

SANDY, notice the warmth from DAVE's arm. Notice how it spreads, making your breathing heavier. Almost audible.

"Beep!"

Now, before we begin the unbuttoning, DAVE and SANDY, you need to enter your Personal Relationships registration code. If you don't have a number yet, you can obtain one immediately by calling 1-800-NDR-WARE. Be sure to have your VISA or MasterCard handy.

Registration Fee:

For $100 you get the code immediately.

For $50 we call you back after you've cooled off for 5 minutes.

For $15 we call you back next week.

Note: Your shameless activity, DAVE and SANDY, has been recorded in the system, along with the time and date (you did, after all, select number 4). If you attempt to exit the program at this time (via reset, power switch, etc.) to avoid registering, this machine will, forever, at random, display the gory details of this tryst on the screen.

"Beep!"

*Please enter your registration code:*_____

"Beep!"

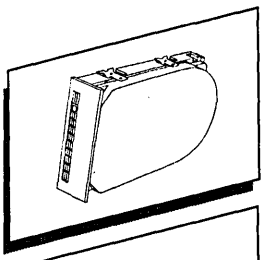
*Please enter your registration code:*_____

"Beep!"

*Please enter your registration code:*_____

"Beep!"

♦ ♦ ♦

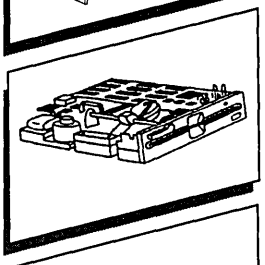


20 MEG HARD DRIVES

SEAGATE Model ST-225
(Full 6 Mo. Warranty)*less face plate* $219.95

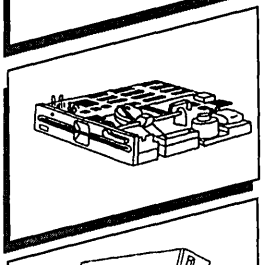
Western Digital Controller For Model ST-225 Drive
$69.95 w/cable

Yes!! These are for IBM compatible machines.



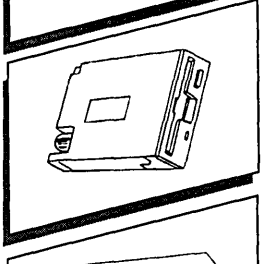
SHUGART # 465

QUAD DENSITY, 720 K, 96 TPI
1/2 HEIGHT
BLACK FACE PLATE
(NEW BUT HAVE COSMETIC BLEMISHES)
SOLD AS IS $49.95



TOSHIBA 5 1/4" FLOPPY DRIVES

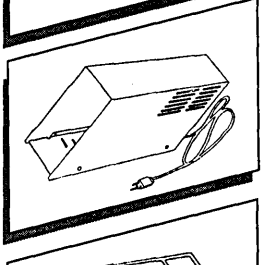
DSDD, 360 K
1/2 HEIGHT
BLACK FACE PLATE
(90 DAY WARRANTY) $89.95



TOSHIBA 3.5" DISK DRIVES

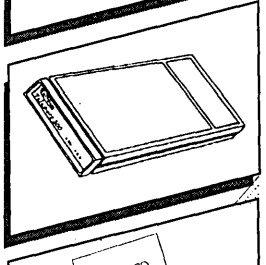
IBM compatible
720K Byte
Double sided
Mounting kit
Power & Data cable adapter
Model # FDD 4210GOK

FULL 90 DAY WARRANTY $119.95

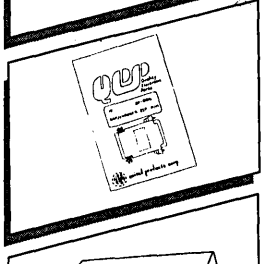


5.25" DISK DRIVE CABINETS
(with power supply)

For 1 full height or 2 half height
#CAB-25V5
$99.95 each

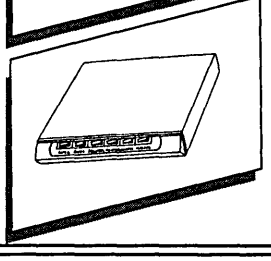


300 BAUD SMART MODEM (DIRECT CONNECT)
LOW COST SERIAL MODEM
7 x 10 x 1 1/2
$19.95



CABLES-CONNECTORS-SEX CHANGERS

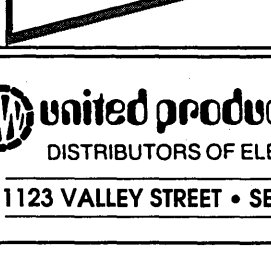
"XT" STANDARD PARALLEL (10', DB-25M TO CN36M)	#10-004	$12.95
AC POWER CORD (6' COMPUTER STANDARD, MOLDED RUGS)	#10-001	2.95
RS232 MODEM ADAPTER (DB-25 M-F, PINS 2 & 3 REVERSED)	#20-001	9.95
SEX CHANGER, DB-25 F TO F	#20-005	9.95
SEX CHANGER, DB-25 M TO M	#20-006	9.95



COMPUTER POWER SUPERVISOR (A LOW COST INSURANCE POLICY
PROVIDES SURGE/SPIKE PROTECTION
FOR YOUR COMPUTER!
ALL INCOMING POWER IS MONITORED & CONDITIONED BEFORE ALLOWING IT TO POSSIBLY TRASH YOUR COMPUTER OR OTHER SUCH VUNERABLE & EXPENSIVE EQUIPMENT.

- 5 OUTLETS, EACH WITH A LIGHTED SWITCH
- 1 MASTER ON/OFF SWITCH
- SYTLISH CABINET PROVIDES FULL SHEILDING
- RATED 15 AMP, 125 VAC, 1875 WATTS

DIM. 12.5" x 2.5" x 14" Regular price $69.95 Your price $49.95



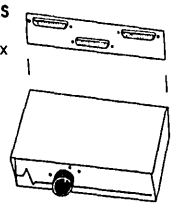
5.25" HARD DISK
CONTROLLER CARD
FOR ANY MODERN 5.25" HARD DRIVE

$89.95 W/CABLE

COMPUTER A-B SWITCH BOXES

#CN36 A-B Parallel Centronix
$24.95 each

#DB25 A-B Serial RS-232
$24.95 each



STEPPER MOTORS

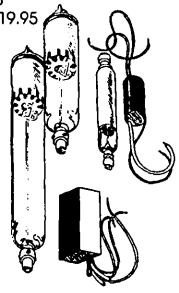
Copal #SP-57
1/4" Shaft, 7.5 deg./step, 36 Ohm, 12VDC $6.95
3/16" Shaft, 35 oz." torque, 2.1A., 5VDC
WITH BRASS GEAR, 20 TEETH, 1/2" DIAMETER $9.95

LASERS

5 MW Laser Tube $89.95
Power Supply Kit (115VAC) $69.95
Power supply (wired) (12VDC) $119.95

1 MW Laser Tube $119.95
Power Supply (12VDC) $99.95

(These lasers are brand new and guaranteed to have a cosmetic defect or not meet manufactures full specifications. All are tested in our lab to insure your satisfaction.)



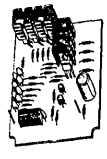
WARNING: Voltages present and used by lasers can be lethal...Permanent eye damage could result from direct exposure to an on coming laser beam. Only those persons qualified to handle such potentials should do so...

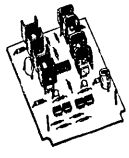
TECHNA-KIT

D-C Motor Controller

- Control 2 D-C motors with a computer or other logic source
- For motors rated 6-24 VDC
- Control forward/reverse/run/cw/ccw/stop
- Up to 6 Amp starting surge, 4 Amp cont.
- Dynamic breaking (capable)
- Will also run most 4-lead stepper motors

$29.95





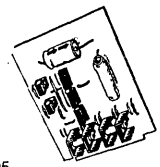
USMD-C

- Control standard 6-lead stepper motors with a computer or other logic source
- For motors rated 1.7 - 12.0 VDC
- Optical isolation
- Control: forward/reverse/step rate/stop
- Industry standard 22 pin edge card connector

$29.95

COMPUTER AUTOMATE

- Use your computer to provide automation
- 8 separate driver ports per card
- 8 TTL/Cmos inputs
- 1 user defined sense switch
- 6-24 VDC
- 4 Amps/driver (max current)



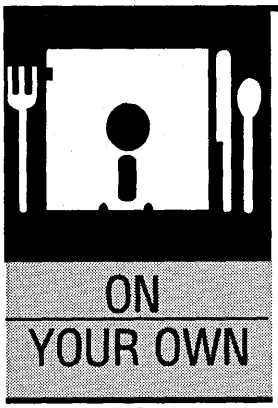
$29.95

united products corporation
DISTRIBUTORS OF ELECTRONICS SINCE 1968
1123 VALLEY STREET • SEATTLE, WA 98109-4425

VISA AMERICAN EXPRESS

***PHONE:* (206) 682-5025**
***FAX:* (206) 682-5593**
M-F 9-6 SAT 9-5

Reader Service Number 101



Building A (Small) Publishing Empire

From (Almost) Nothing

By Kent Peterson
Psion Interest Group
41 Greenridge Avenue
White Plains, NY 10605
(914) 761-2196

You asked for it, more personal stories from "On Your Owners." Well, this is personal. Kent talks about his new start-up, a newsletter for a very small computer. (Definitely not the Big Board.)

Last year I landed the kind of job my mother always told me I wanted. I became a systems analyst for a fairly large corporation which gave me job security, a pension plan, full medical, and a fast AT with almost enough disk space.

Still, the hacker in me wasn't content. I spent more of my time in meetings than I did in microcode. My jeans and Nikes didn't look quite right with my white shirt and tie. I knew there was more to life, but I also needed the money my "real job" provided. How depressing.

As usual, my wife, Christine, knew just how to snap me out of my depression. "You need a gadget," she said. She was right. Somewhere among the megabytes and meetings, life had stopped being fun. I had started in this business hacking on little machines and I missed the challenge of pushing a little machine to its limits.

Then, one fateful day, the boys down in accounting called up to tell me I hadn't claimed enough of my miscellaneous moving allowance. I tried to explain that it hadn't cost much for me to move, but they insisted that the company owed me a couple hundred dollars. Furthermore, if I didn't take it, it would screw up their bookkeeping. Of course I refused their offer (for about seventeen nanoseconds).

I spent the money on a funky little machine called the Psion Organiser II. *(Editor's note: Psion is pronounced SIGH-on, a psemi-psimple name.)* This eight ounce box, billed as the "pocket PC," was just what I was looking for. The unit contains a 1 MHz Hitachi 6303X microprocessor (a CMOS version of the Motorola 6809) and 32K of RAM. It has a calculator-style keyboard and a two line liquid crystal screen. It runs a proprietary language called OPL.

The Organiser doesn't use disk drives; it uses tiny packs that contain either EPROMs or battery-backed CMOS RAM. It even reads from and writes to these packs while communicating through an eight-bit (proprietary) parallel port.

Psion also manufactures a device called a CommsLink that converts this port to a smar' RS-232C. In short, the Psion offered me enough sameness along with enough weirdness to stave off boredom for awhile.

After developing a relationship with the little Psion, Christine and I wrote a few little music routines and I managed to squeeze in a text adventure. Then I got serious. I made it talk to every other computer we had — the AT, the Hewlett-Packard, and the Atari.

The Psion keyboard lacks some characters that I consider essential (like the question mark). So I dug through the tech manual until I figured out how to re-vector the keyboard lookup table into RAM to give me an alternative keyboard layout. What fun. I was having so much fun with the little machine that I decided to do something crazy.

Thoughts Of Grandeur

"I'm going to become the Peter Norton of the Psion Organiser II," I told my wife.

"Oh no," she said. "Does this mean you'll start wearing pink shirts and ties and stand around with your arms folded and your shirt sleeves partially rolled up?"

I clarified my position. I wasn't talking about a wardrobe change. I just figured I could become the acknowledged expert on the Organiser. This would require staying current on the latest developments in the Psion world, getting all the latest hardware and software, and spreading that information within the Psion community.

"Sounds expensive," Christine said.

"Not if we do it right," I countered. "I'm going to make this machine pay for itself." She just smiled. I've said that about every computer I've ever owned and it's seldom proven true.

Newsletteritis

I decided to start a users group for the Organiser II and publish a small newsletter of program tips, techniques, and reviews. I figured membership dues would cover the cost of printing and mailing the newsletter with

enough left over to keep me stocked with the latest hardware and software.

I checked, there was no Psion users group in the U.S. The British build the little computer so there's an active group there.

Jeff Etter, who'd sold me my Psion, thought the U.S. market was rich enough to support a group and offered to help any way he could.

The plan we formed was simple. I'd write the first issue of our newsletter. Christine (a former editor) would proof the copy and correct any misspellings. When everything passed her inspection, we'd print the script on a borrowed laser printer. Then we'd take the laser printouts to a local print shop. (The first print run totalled 100 copies.)

I planned to send these copies, together with subscription forms and pleas for contributions, to 100 of the people who'd bought Psions from Jeff. If we were lucky, we'd make back the money we'd spent on the first printing and be able to keep going. If not, Christine would hopefully remember she'd married me for poorer, too.

Really Cheap Desktop Publishing

Our plan depended on desktop publishing, but we had no budget. We didn't have Ventura, a scanner, or our own laser printer. We did have PC-Write. This forced us to adopt a very simple format for the newsletter: single column, no photos, and no graphics beyond the simple box-drawing characters that PC-Write supports. This no-frills approach gave us some unexpected benefits. Since we didn't have one of those wisi-wonderful graphics packages, I spent my time writing instead of playing around with graphics. (It seems like whenever I get hold of a mouse and a bit-mapped image, I spend more time tweaking than working.)

Another benefit was that we could print our proof sheets on the trusty old Proprinter rather than use the laser printer for everything. Finally, since PC-Write uses straight ASCII, the output could be used in a disk version of our newsletter. (Of course, most of this was just rationalization. When Ashton-Tate offered us a copy of BYLINE, we jumped at it.)

PIGs

We immediately realized our users group needed a name, so Christine and I discussed possibilities: "We can't call it the Psion Interest Group," I said.

"Why not?"

"The initials spell 'PIG.' We can't have a group called PIG. It isn't dignified."

"I think it suits you," she said sweetly.

I looked around my office at the scraps of paper covered with program fragments, the semi-random stacks of books and computer magazines, the empty Oreo bags and Pepsi cans.

"I still don't think it's a very good name," I said. "Can't we come up with something else?"

"You could always call it the Psion International Software Society," she chuckled.

"I guess PIG isn't such a bad name," I said.

"Oink, Oink," she squealed.

We set to work writing, editing, and programming. We worked ideas into programs and wrote articles to accompany the code. We pressed anything with a keyboard and an RS-232 port into service for the writing. The Psion's small size and portability made it a natural for note taking. Anywhere inspiration struck, the Psion was there. (It's amazing how many good ideas come to you when you're in the bathroom.)

The Psion also served as a bridge among different machines. They couldn't agree on disk formats, so the Psion served as an ASCII Esperanto that they all could talk to.

Finally we finished our first issue, sent out the 100 copies, and waited for people to make PIGs of themselves.

They did. Not all, not a majority, but enough. And, Jeff Etter kicked in some cash to help with postage costs.

PICO magazine did a review of the Psion and fortunately the review contained a technical error. I wrote a letter correcting the error and mentioned our users group. Lo and behold, more PIGs.

Greg Paul, the director of technical support for Psion, USA, learned of our efforts and directed people our way. One of our members contacted the British users group and we established a publication swap with them.

We not only gained members, we gained a few good writers and a couple of hotshot programmers. Ideas came pouring in. PIGs are mostly users, not programmers. But the users have plenty of ideas for keeping programmers busy.

I have always relied on the strangeness of kind people and the kindness of strange people.

Problems

Of course, not everything went smoothly. When I was ready to print the second issue of our newsletter, the laser printer was nowhere to be found. It seems the guy who owns the printer lent it to someone else. (That's the problem with relying on one of these generous types, you never know when they're going to be generous to somebody else.)

A few frantic phone calls connected us with a laser-printer owner who let us print the 24 pages of the second issue during lunch hour. To our relief, and our subscribers' amazement, the second issue hit the streets reasonably close to its intended date.

A Conference

Before the third issue, our editorial staff (and the kids) flew down to sunny Florida for three days of sun, fun, and tech-talk at Jeff Etter's Psion Conference. The conference was a meeting ground for Psion VARs (Value Added Resellers) and manufacturers of Psion-related hardware and software.

It was our first chance to meet many of the Psioneers face to face and discuss what they would like from a users group. We signed up subscribers, played with gadgets, and met some amazing people.

Shortly after the conference, interesting items began appearing in our mailbox. (Now I understand exactly how Jerry Pournelle feels.) Psion programs ranging from word processors and database managers to celestial navigators began piling up in our apartment.

The most significant piece of software wasn't for the Psion, however. It was a copy of BYLINE, Ashton-Tate's desktop publishing software. BYLINE came to us courtesy of Ashton-Tate and Jeff Etter.

Every Silver Lining Has A Dark Cloud

BYLINE turned out to be a mixed blessing. Our third issue had pushed PC-Write to its limits. (That issue included two simple illustrations and a very simple circuit diagram.) BYLINE promised to give us many more fonts and sizes, along with the easy addition of real graphics. The package was easy to learn, very powerful, and gobs of fun. Our fourth issue was only six weeks late.

In fairness to Ashton-Tate, I have to say that the delay wasn't BYLINE's fault. BYLINE performed gallantly; the problems I encountered are inherent in any desktop publishing system.

First, I had to figure out what the publication should look like. Should we use Times-Roman or Bookman? Which point size? How should listings look?

But I'd poured an issue's worth of text and programs into BYLINE when I discovered an even bigger problem. I only had half a newsletter. Our new typeface and multi-column format held a lot more information per page.

Another problem was the art. They say a picture is worth a thousand words, but I found out that it's much easier for me to write a thousand words than to mousepaint a picture.

> I have always relied on the strangeness of kind people and the kindness of strange people.

BYLINE could import art from a paint program or a clip art library, but the illustrations I wanted were not available as clip art. A scanner would solve the problem, of course, but my budget didn't include a scanner.

Unexpected Features

Like any sophisticated piece of software, BYLINE has a few undocumented "features." A personal favorite is the way it inserts a space following the hyphen in hyphenated words. After I'd tried the usual techniques of manual-reading and hair-pulling, I stumbled on the solution. I manually insert an invisible space after the hyphen.

Editor's note: Of course — an invisible space! Why didn't I see that? (All oxymorons are obvious.)

I used a similar technique to get BYLINE to print a program listing containing a less-than sign followed by a quotation mark. BYLINE uses the <" and "> as variable delimiters for things like page numbers. It refused to print the program listing until I inserted an invisible space between the < and the ".

BYLINE gave us enormous layout flexibility, but it did limit our printer options. BYLINE supports both dot-matrix and laser printers so we could still use the Proprinter for proofing.

Unfortunately a dot-matrix printer has to do a digital hand stand to print all those wonderful fonts and pictures. So we got a lot of snack breaks while the printer exercised. Of course, we did most of the proofing on the screen, but Christine insists on paper and red pen for her final pass.

After more writing, rewriting, editing, and playing, our fourth issue got shipped in PostScript form to Psion's U.S. headquarters. They dumped it out on their PostScript laser printer. (None of our local friends had laser printers with PostScript or the font support we needed.) Finally, for better or worse, the issue hit the streets.

So How Are We Doing?

We're now at work on our fifth bimonthly issue and we have about 200 members in a dozen countries. Our membership fees ($25 U.S., $40 foreign, and $15 for students, senior citizens and hard-luck stories) bring in enough to cover printing, postage, and the occasional gadget.

If I figured out what I'm making per hour on this venture, it would come to somewhere around six cents. But if I factor in the people I've met, the lessons I've learned, and the fun I'm having, I'm richer than I've ever been.

There is one little thing that bothers me, though. I was really looking forward to being the Peter Norton of the Psion Organiser II.

I neglected one little detail, however. When Peter formed his company, he called it Peter Norton Computing. When I formed mine, I called it PIG. My wife took note of the name and the number of nights I spent at the computer and christened me "The Big Boar." I have no idea what Peter Norton's wife calls him (or if she calls him at all), but at least I don't have to wear a pink shirt.

Psion Organiser II

Weight: 9 ounces
Dim: 6" long 3.8" wide 1.2" deep
Battery: 9 V transistor
Memory: 32K main memory
Language: OPL
Processor 1 MHz Hitachi 6303X
Price $250

Accessories

32K RAM pack (battery backed)
32K EPROM pack (EPROM burner built into Organiser)
128K RAM pack available soon.

♦ ♦ ♦

GEMS SUPER DISCOUNT* PRICES

MOTHERBOARDS

XT/8088
4.77/8mhz TURBO, 640K,OK $69
4.77/10mhz TURBO, 640K,OK 76
4.77/10mhz TURBO, 1mb, OK, S/W 79
4.77/15mhz SUPER TURBO, w/1mb MEMORY, 2/3 size, 110% faster than 6mhz IBM AT . 420

AT/286 AT SIZE
6/10mhz, 1mb OK, DTK, Ows 229
6/10mhz, 1mb OK, PC CALC, Ows 259
6/12mhz, 1mb OK, PC CALC, ows 289
6/12mhz, 1mb OK, EW, 12mhz CPU, Ows..289
6/16mhz, 1mb OK, PC CALC, Ows 399

AT/286 XT SIZE
6/10mhz, 1mb OK, DTK, Ows 239
6/10mhz, 1mb OK, ELTECH, Ows 229
6/12mhz, 1 or 4mb, OK, MS, Ows 265
6/16mhz, 2 or 8mb, OK, NOVAS 489
6/20mhz, 2 or 8mb, OK, NOVAS 549

AT/386 AT SIZE
16mhz, w / 1mb MICRONICS 1399
20mhz, w / 1mb, MICRONICS 1449
20mhz, w / 1mb, 287 + 387 SOCKET,MS.1415
20mhz, w / 1mb, up to 16mb on M/B,1 par, 1 ser w / 2nd opt ..... CALL
20mhz MYLEX motherboard ..... CALL
25mhz 38C w/1mb ..... CALL

HARD DRIVES

ST-225 Seagate (DRIVE ONLY) $215
ST-225 with CONTROLLER 255
ST-238 Seagate (DRIVE ONLY) 229
ST-238 with CONTROLLER 275
ST-251 Seagate 40ms HARD DRIVE 339
ST-251-1 Seagate 28ms HARD DRIVE ... 419
MR535 Mitsubishi RLL 65mb 28ms 465
ST-125 Seagate 3-½ format 20mb 245
Miniscribe 3650 40mb (61 ms) MFM 315
Miniscribe 3675 63mb (61 ms) RLL 335
MAXTOR....Call For Special Prices ... $CALL
TOSHIBA MK130 65mb RLL 3-½ form .. 575
HARD CARD 20mb Plus Development ... 545
HARD CARD 40mb Plus Development ... 729

FLOPPY DISK DRIVES

Fujitsu 360K BLACK FACE PLATE $64
Fujitsu 1.2mb BEIGE FACE PLATE 83
Teac 360K 73
Teac 1.2mb 86
Teac 720K with 5-¼ mount bracket 92
Teac 1.44 3-½ w/ 5-¼ bracket 110
Mitsumi 360K BLACK 63
Sony 720K 3-½ w/ 5-¼ bracket 94
Sony 1.44mb 3-½ w/ 5-¼ bracket 119
Toshiba 720K 3-½ w/ 5-¼ bracket 99
Toshiba 1.44mb 3-½ w/ 5-¼ bracket 119
Mitsubishi 360K 69
Mitsubishi 1.2mb 84
Mitsubishi 720K 3.5 79
Mitsubishi 1.44mb 3.5 95

CASE

XT SLIDE $27
XT/AT Like Case 30
AT 3 Floppy Drive Front / 2 Button 54
AT 3 Floppy Drive with Digital Disp 72
XT SLIDE Heavy Duty, quality 27
XT/AT like, slide, heavy duty, qual 30
AT DT 3DR FT, 2 button, SOL 48
AT DT 3DR FT, 2 button, AS 54
AT DT 3DR FT, 2 button, DIG DISPLAY ... 72
AT DT 3DR FT, DIG DISPLAY/PC CALC 75

SPECIAL CASES

Transportable (case PS KB Mono Monit)$435
Transportable EGA VER 1250
LCD portable 640X400 Bracklite 18lbs ... 759
Carrying Case for portables 35
Tower Case 230W PWR SPLY, 6 half ht . 250
Tower Case w/digital display, 3 Dr. Front will hold 7, ½ht Dr. 220W 225

POWER SUPPLY

150 watt XT Compatible $36
200 watt XT Compatible 49
200 watt AT Compatible 59
220 watt AT Compatible 63
250 watt AT Compatible 69

KEYBOARDS

84 KEY At Style Keyboard $36
84 Key At Style Maxi Switch Keyboard 49
101 Key Enhanced Monterey 42
101 Key Tronics Keyboard 47
101 Maxiswitch Enhanced Keyboard 64
101 Tactile Enhanced Keyboard 49

MOUSE

Logitech M-8 Button $35
Logitech C-7 Button 69
Logitech HI-REZ 86
Microsoft Bus Mouse + Paintbrush/Menu . 99
Microsoft Ser Mouse + Paintbrush/Menu . 99

VIDEO CARDS

Monochrome W/ Printer Port $39
Monochrome 132 Column & Print Port ... 53
Color Graphics Adaptor 39
Color Graphics Adaptor 1/ print port 53
Micro EGA Auto Switching 129
Video-7 VGA 269
Paradise VGA Plus 259
Paradise VGA Professional 389
Chips and Technologies VGA Card 249

MODEM

1200 BAUD with Software $52
2400 BAUD with Software 95
Zoom HC 2400 w/Procomm Software ... 116
External 1200 w/PC Talk III S/W 62
External 2400 w/PC Talk III S/W 109
Everex External 2400 Modem 179

I/O CARDS

Multi I/O Ser Par Cal Clk Game Disk $48
Multi I/O Above (DFI) 66
Magic I/O Ser Par Cal Clk Disk Ctlr 360K, 720K, 1.2mb 49
I/O Plus.. Ser Par Cal Clk Game 42
Parallel Card (XT) 15
Serial Card 2nd Serial Optional 18
Port Serial Card w/2 Port & Opt 66
2nd Serial for XT 17
Game Card (2 Ports) 17
AT I/O CARD par, game, ser, 2nd ser opt . 42
2nd Serial Port for AT 19

Floppy/Hard Disk Controller

2 Drive Floppy CTLR (360K/720K) $17
4 Drive Floppy CTLR (360K/720K) 34
2 Drive Floppy Master (360K/1.2mb) 39
4 Drive FDC (360, 1.2/720, 1.44)SEF 89
4 Drive Super Special FDC W/Cables 89
WD FOX 2 Floppy Drive CTLR For AT 57
WD WX 1 Hard Disk Controller 8 Bit 57
WD XT/GEN Hard Disk CTLR For XT 52
WD 27X RLL Hard Disk Controller 59
WD WA-2 FD/HD CTLR For AT (MFM) ... 112
DTC 5287 FD/HD CTLR For AT (RLL) 154
NCL 5425 FD/HD CTLR For AT (MFM) ... 112
ADAPTEC 2372 FD/HD CTLR AT (RLL) .. 179
16 Bit 4 Floppy/2 Hard Drive Controller Modified 16 Bit DTC CTLR MFM & RLL . 295

TAPE BACKUP

Teac 60mb MT2ST/45 (CASS) INT $539
Everex 60mb Wangtek INT (Cart) 679
Colorado 44mb (Cart) Internal 289

MONITORS

Goldstar Amber Monochrome 720X348 . $58
Samsung 1252 Mono 720X348 TIL/SW .. 66
Samsung 1464 EGB 219
Eversion 14 Amber Flat Screen 119
Amdek 410A 139
Nec GS 199
Taxan Composite Amber/Green 89
Relisys EGA with TILT & SWIVEL 339
Relisys Multi Scan 499
Mitsubishi 1410XC EGA 379
Mitsubishi 1381A Diamond Scan 539
Nec Multiscan II 569
Sigma Designs Laserview 1901-PC 1749

CAD PRODUCTS

MITSUBISHI HA3905 19V/20" MONITOR, ANALOG/TTL, 1024 X 1024, 15.7 35.5khz $1725
MITSUBISHI HL6905 19" MONITOR, 1280 X 1024, 30⅞ 64khz AUTO TRACKING 2415
HIGH RESOLUTION VIDEO CARDS FOR CAD
QDP VIVA 1280 1290 X 768 16 COLORS . $1289
QDP VIVA 2000 1024 X 1024 16 COLORS..1559
QDP VIVA 2000 2 2024 X 2024 16 COLORS.1895

CHIPS (Prices Subject To Change)

V-20 8mhz (replacement for 8088) $10
V-20 10mhz 16
8088-2 CPU 5
8087-2 MATH CO-PROCESSOR FOR XT . 149
8087-1 MATH CO-PROCESSOR FOR XT . 215
80287-8 MATH CO-PROCESSOR FOR AT . 229
80287-10 MATH CO-PROCESSOR FOR AT.279
80387-16 MATH CO-PROCESSOR FOR AT..425
80387-20 MATH CO-PROCESSOR FOR AT.CALL
64K 150ns DRAM 1.35
64K 120ns DRAM 3.25
64K 100ns DRAM 4.25
64K 120ns DRAM 4464 ..... 13.50
256K 150ns 10
256K 120ns 11.75
256K 100ns 11.95
256K 80ns 12.95
1mb 100ns 33
256K 100ns SIMM'S (SIP's) 125
256K 80ns SIMM'S (SIP's) 145
256K 100ns STATIC COLUMN RAM 17
256K 80ns STATIC COLUMN RAM 18

Desktop Publishing

Hi-Res 1024 15"MONITOR W/Video Controller 1024 X 768 Resolution $499

SCANNERS

Mitsubishi Hand Scanner Full Page MH216 200 DPI, 8.5" X 11" $638
Mitsubishi 400 DPI MH 130 ..... CALL
Mitsubishi Paper Feeder For Scanner 176
Abaton 300 FB Full Page Scanner 1349

ACCELARATOR CARDS

MICRO 286-10 OK $269
SOTA 286i 10mhz OK 298
SOTA 286i 12mhz OK 359

MOTHERCARDS

SOTA 5.0 MOTHERCARDS w/1mb, 10mhz.$779
SOTA 5.0 MOTHERCARDS w/1mb, 12mhz.879

Econo XT® Compatible

8mhz Turbo M/B,0K, case, Keyboard, 150w P/S, FLOP CTLR, 1 360K Drive, Mono video B.D.
MONITOR $339

SPECIAL

Lap Top Computer

The New Mitsubishi MP286L 12mhz,1.44,20mb.....$2475

Call for Discounted System Prices

***All prices shown are pre-paid or ordered by VISA or Mastercard. Charge card orders are subject to a 3% surcharge. For C.O.D. or term orders on parts above add 10%.**

XT® AT® & IBM® Are Registered Trademarks of International Business Machines.

GEMS computers

3446 De La Cruz Boulevard
Santa Clara, California 95054

In Business Since 1985
A member of the Better Business Bureau and Chamber of Commerce.

HOURS: M-F: 9 A.M. - 9:00 P.M. EST
7:00 A.M. - 7 P.M. PST SAT: 8 A.M. - 6:00 P.M. PST

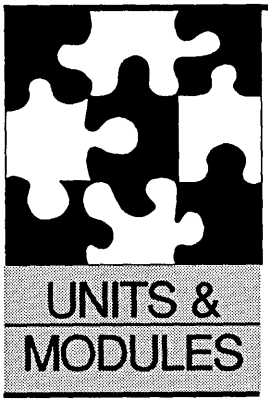
1-800-332-GEMS

IN CA 408-988-0161 TECH SUPPORT 408-988-0146
FAX 408-988-0609

Shipping: 4% plus $3.00 handling on all part orders (except cases, 9% + $3.00). APO/FPO orders add 8% plus $3.00 on part orders. Call for exact charges on Systems and Monitors. CA residents add 7% tax. Prices reflect 3% cash discount.

One Year Warranty On All Parts And Systems / All Orders Are F.O.B. Santa Clara.

Reader Service Number 130



Turbo Pascal 5.0

And A Rational Number Toolkit

By Michael S. Hunt
845 E. Wyeth
Pocatello, ID 83201
(208) 233-7539

Herein Michael covers Turbo Pascal 5.0 and unreal reals. I mean real integers. Or is it rational real integers? Ah well, he'll tell you more precisely than I can.

In this issue I'll briefly review Turbo Pascal 5.0. (This'll be brief compared to the extensive Modula-2 compiler review coming next issue.) Then for those of you tired of numeric roundoff error, check out the rational number toolkit. It contains the basic arithmetic functions to compute with rational numbers.

After convincing the post office that my mail should be delivered to my house and not E. B. Hunt three blocks away, I received some reader mail. Remember you can write and complain, explain, applaud or just shoot the breeze. I'll even answer. I welcome any suggestions for topics or directions for the column. I also check the Micro C BBS about once a week.

Turbo Pascal 5.0

Well, I've had Turbo Pascal 5.0 for about two months now. I like it, I like it. I upgraded from version 2.0 so many of the features mentioned are also in versions 3.0 and 4.0.

At last, units. I'm a fanatic about reusable code and problem solving tools. I've built and used modules in Modula-2, but now my 5000 line Pascal programs can be managed instead of manhandled.

Procedure passing is essential for truly flexible tools. The classic example is a sort procedure that can pass the appropriate comparison procedure in the parameter list. With a few sort procedures, several comparison procedures, and some carefully constructed selection routines, you can build a very versatile sort package. Hmmm, sounds like a possible column topic.

I have several programs in need of some assembly language speed ups. The ability to link .OBJ files and create .EXE files without overlays is welcome. (You can still do overlays, though.)

Ever since I purchased Logitech Modula-2, I've been addicted to debuggers. I could get along without them, but I'd hate to. Debuggers, even rudimentary ones, are tremendous time savers.

The integrated source level debugger in 5.0 is not full featured like CodeView. Borland recommends that you buy Turbo Debugger for industrial strength debugging. Scott Ladd reviewed Turbo Debugger and Turbo Assembler and took a look at what debugging is all about in his C'ing Clearly column. Look for it in *Micro C's* January/February issue (#45). You can purchase Turbo Pascal 5.0, Turbo Debugger and Turbo Assembler (as the professional package) for $250.

Turbo Pascal 5.0 supports the 80x87 numeric coprocessors. Compiled programs are smart enough to use the 80x87 if present, or emulate one if not.

This package can take advantage of EMS for overlay files and the integrated editor. The overlay manager can load the overlay file into EMS memory so overlays load faster.

If you have 64K of EMS memory, the edit buffer will be placed in EMS memory. This frees up conventional memory and allows you to debug larger programs. The integrated editor is WordStar compatible with some extensions for special program editing features. The editor is fully configurable so you can customize the commands to mimic your favorite editor.

Borland's Graphics Interface (BGI) unit contains more than 50 graphics routines. BGI supports CGA, MCGA, EGA, VGA, Hercules, AT&T 400, 3270 PC and IBM-8514. BGI will autodetect all but the 3270 PC and IBM-8514 graphics cards. You can link all the device drivers and have the program automatically choose the best resolution at runtime. BGI is quite extensive and can help give your program that polished look.

The Turbo Pascal 5.0 package also contains a make utility, a smart linker that removes unused code, an integrated environment with context sensitive help system, and well written reference and user guides. If you're writing in Pascal for IBM compatibles, I strongly recommend Turbo Pascal 5.0.

Get Rational

I do a lot of work with floating point numbers. I get tired of having my results turn out useless because of the accumulated roundoff error after repeated multiplication. Even more subtle is the error introduced when we subtract two numbers very close to each other (i.e., $|a-b|=0.0000001$) when we have only six decimal places of accuracy.

Even though we think of computers as being very accurate, floating point numbers are just approximations. Integer and cardinal values can be represented accurately as sums of powers of two. Since the computer represents everything in binary (powers of two), we can accurately represent integers and cardinals.

The trick would be to represent real numbers with integers. This is where rational numbers come to the rescue. A rational number is any number that can be represented in the form a/b where a and b are integers and b is not equal to zero. Since the rational numbers are only a subset of the real numbers, they're a bit limited. The square root of 2 and pi are not rational numbers.

If your math can be restricted to rational numbers, you can avoid roundoff error. The unit Rational (Figure 1) is a set of basic tools for arithmetic with rational numbers. Figure 2 lists the tools.

The Euclidean Algorithm

The real workhorse of the unit is the GCD function, called by all the other procedures and functions in the unit except ZeroRat. Using the Euclidean Algorithm, GCD finds the greatest common divisor for any two numbers.

The Euclidean Algorithm decomposes a pair of numbers x and y, where x and y are not both zero and x >= y, by calculating r = x mod y. It then lets x = y and y = r and repeats the process until r = 0. When r = 0, the previous r or the last nonzero r is the greatest com-

Figure 1 — Rational Number Unit

```
unit Rational;

interface

type baseType = longint;          (* shortint, integer, longint *)
     ratType = record
                 n : baseType;    (* numerator of rational number *)
                 d : baseType     (* denomenator of rational number *)
               end;

function GCD(x, y : baseType) : baseType;
function LCM(x, y : baseType) : baseType;
procedure LowestTerms(var a : ratType);
procedure ComDenom(var a, b : ratType);
procedure IncRat(var a : ratType);
procedure DecRat(var a : ratType);
procedure ZeroRat(var a : ratType);
procedure AddRat(a, b : ratType;var c : ratType);
procedure SubRat(a, b : ratType;var c : ratType);
procedure MultRat(a, b : ratType;var c : ratType);
procedure DivRat(a, b : ratType;var c : ratType);

implementation

function GCD(x, y : baseType) : baseType;

(* function GCD finds the greatest common divisor of the two numbers
   x and y by the Euclidian Algorithm. *)

var r, d : baseType;
begin
  if x < 0 then x := -1 * x;      (* set x = |x| *)
  if y < 0 then y := -1 * y;      (* set y = |y| *)
  if x < y then begin
    r := y;                  (* swap x and y if y > x    *)
    y := x;                  (* algorithm expects x >= y *)
    x := r
  end;
  if y = 0 then
    GCD := x
  else
  begin
    repeat
      d := x;              (* repeat Euclidian Algorithm until *)
      x := y;
      r := y;              (* a remainder of zero is obtained  *)
      y := d mod r;
    until y = 0;
    GCD := r               (* then set GCD to be previous remainder *)
  end
end; (* GCD *)

function LCM(x, y : baseType) : baseType;

(* function LCM finds the least common multiple of the two numbers
   x and y by determining the GCD of the two numbers and then applying
   the rule |a*b| = GCD(a,b)*LCM(a,b)    *)

var g : baseType;
begin
  LCM := abs(x*y) div GCD(x, y)
end; (* LCM *)

procedure LowestTerms(var a : ratType);
var g : baseType;
begin
  if a.n = 0 then            (* if numerator then set denominator *)
    a.d := 1                 (* to smallest positive number       *)
  else
    begin                    (* find the GCD of the numerator *)
      g := GCD(a.n, a.d);    (* and the denominator           *)
      a.n := a.n div g;      (* divide numerator by GCD       *)
      a.d := a.d div g       (* divide denominator by GCD     *)
    end
end; (* LowestTerms *)
```

continued next page

Figure 1 continued from page 69

```
procedure ComDenom(var a, b : ratType);      (* find a common denominator *)
var m : baseType;                            (* for a & b and adjust them *)
begin
  m := LCM(a.d, b.d);              (* find LCM of a.d and b.d *)
  a.n := a.n * (m div a.d);        (* set a.n to new value    *)
  a.d := m;                        (* set a.d to LCM of denominators *)
  b.n := b.n * (m div b.d);        (* set b.n to new value    *)
  b.d := m                         (* set b.d to LCM of denominators *)
end; (* ComDenom *)

procedure IncRat(var a : ratType);      (* increment a by 1 *)
begin
  a.n := a.n + a.d;       (* add denominator to numerator *)
  LowestTerms(a)          (* reduce a to lowest terms *)
end; (* IncRat *)

procedure DecRat(var a : ratType);      (* decrement a by 1 *)
begin
  a.n := a.n - a.d;       (* subtract numerator from denominator *)
  LowestTerms(a)          (* reduce a to lowest terms *)
end; (* DecRat *)

procedure ZeroRat(var a : ratType);     (* set a to 0 *)
begin
  a.n := 0;    (* set numerator to zero *)
  a.d := 1     (* set denominator to smallest positive number *)
end; (* ZeroRat *)

procedure AddRat(a, b : ratType;        (* add a and b and place *)
                 var c : ratType);      (* result in c, a+b = c  *)
var l, g : baseType;
begin
  LowestTerms(a);         (* reduce a to lowest terms *)
  LowestTerms(b);         (* reduce b to lowest terms *)
  ComDenom(a, b);         (* convert a & b to common denominator *)
  c.n := a.n + b.n;
  c.d := a.d;
  LowestTerms(c)          (* reduce c to lowest terms *)
end; (* AddRat *)

procedure SubRat(a, b : ratType;        (* subtract b from a and place *)
                 var c : ratType);      (* result in c, a-b = c        *)
begin
  LowestTerms(a);         (* reduce a to lowest terms *)
  LowestTerms(b);         (* reduce b to lowest terms *)
  ComDenom(a, b);         (* convert a & b to common denominator *)
  c.n := a.n - b.n;
  c.d := a.d;
  LowestTerms(c)          (* reduce c to lowest terms *)
end; (* SubRat *)

procedure MultRat(a, b : ratType;       (* multiply a and b and place *)
                  var c : ratType);     (* result in c, a*b = c       *)
begin
  LowestTerms(a);         (* reduce a to lowest terms *)
  LowestTerms(b);         (* reduce b to lowest terms *)
  c.n := a.n * b.n;       (* multiply numerators      *)
  c.d := a.d * b.d;       (* multiply denominators    *)
  LowestTerms(c)          (* reduce c to lowest terms *)
end; (* MultRat *)

procedure DivRat(a, b : ratType;        (* divide a by b and place *)
                 var c : ratType);      (* result in c, a/b = c    *)
begin
  if b.n = 0 then
    ZeroRat(c)            (* if division by 0 then c = 0 *)
  else
    begin
      LowestTerms(a);     (* reduce a to lowest terms *)
      LowestTerms(b);     (* reduce b to lowest terms *)
      c.n := a.n * b.d;   (* invert b and multiply by *)
      c.d := a.d * b.n;   (* a to get c               *)
      LowestTerms(c)      (* reduce c to lowest terms *)
    end
end; (* DivRat *)

begin
end.

/***/
```

Figure 2 — Rational Number Tools

```
baseType      shortint, integer or longint
ratType       record of numerator and denominator of baseType
GCD           returns the greatest common divisor of two numbers
LCM           returns the least common multiple of two numbers
LowestTerms   reduces fraction to lowest terms
ComDenom      converts two rational to a common denominator
IncRat        increments a rational by 1
DecRat        decrements a rational by 1
ZeroRat       sets a rational to zero
AddRat        adds two rationals
SubRat        subtracts two rationals
MultRat       multiplies two rationals
DivRat        divides two rationals
```

mon divisor, the greatest divisor of both a and b.

001 = 357 * 2 + 287
357 = 287 * 1 + 70
287 = 70 * 4 + 7
70 = 7 * 10 + 0

In this case, the Euclidean Algorithm applies to 1001 and 357. The last nonzero remainder is 7, so 7 is the greatest common divisor of 1001 and 357. For more on the Euclidean Algorithm, consult a book on Modern or Abstract algebra.

Two rational numbers must have the same denominator before you can add them together or find their difference. The function LCM finds the least common multiple of two numbers. The least common multiple of the denominators is the smallest number they will both divide into. This is the smallest common denominator. ComDenom uses LCM to convert two rational numbers to the same denominator.

IncRat and DecRat increment and decrement the rational number by one, respectively. ZeroRat sets the rational to zero. The numerator is zero and the denominator is one. AddRat, SubRat, MultRat and DivRat each perform the indicated operation on a and b and return the result in c.

These tools provide only the basic arithmetic for the rational numbers. Try writing a procedure such as ExpoRat that returns a rational to an exponential power. Or do operations on vectors and matrices of rational numbers. Hmmm, a matrix toolkit, maybe even sparse, that's another possible column topic.

Warning!

The unit rational does have some limitations. It doesn't detect integer overflow or underflow. It makes an attempt to avoid these errors by calling LowestTerms before and after operations on the numbers. This unit also assumes a nonzero denominator, ratType.d, and a nonzero deviser. You can change the size and range of the rational numbers by changing the baseType declaration. You can use the IEEE integer type comp, but all div's must be replaced with /'s.

I made no provisions for output or conversion to and from real data types. As Larry Fogg says: "But I leave these as exercises for the reader. (The ultimate academic cop-out.)"

Next Time

Next issue I'll have a Modula-2 compiler review, complete with benchmarks, compiler comparisons and programming environment summaries. So far I have one CP/M and four MS-DOS compilers, and I have a good lead on a shareware compiler. After the compiler review, I'll cover the graphics toolkit that I've been wanting to work on for a long time. Then on to ... well you decide. Tune in next time for the latest episode of "Compiler Wars." (Music fades. Lights come up.)

♦ ♦ ♦

Continued from page 4

help, to donate their time so we'll have another one. "I just attended my first SOG, people don't know what they're missing, it's like nothing else." ... "I'll help, though I'm not sure what I'd do from here."

Distribute SOG. We might make it like the SOG of old, only spread out. That way a club (or even an individual) could help by holding a smaller, more informal SOG. Ideally we'd wind up with three to five regional events ... Northwest, Southwest, Northeast, Southeast, and Middle-east (Midwest?).

If you've always dreamed of being surrounded by techies and you know of a wonderful campground, assembly hall, mountain top, farmer's field, picturesque community, seacoast, or whatever — call.

By talking we might both decide your place is crazy or it's perfect. Hopefully, we'll come up with locations people would enjoy visiting. Places where we can gather for two or three days of laid-back babble and technology, places I can write about in *Micro C*.

Anyway, call with your suggestions. And call soon. (I hope it's not too late this year to pull it off.) Anyway, we'll announce what we come up with in *Micro C*: times, dates, what people need to bring (sleeping bags or credit cards), who to contact, what to expect, lodging options and prices ...

If possible, I'll attend them all. After all, I haven't missed a SOG yet. (The last three weeks in July, September, or November would work best for me.)

Send Us Your Tech Tips

As strange as it might seem, Tech Tips is the best read portion of *Micro C*. (Yep, that tiny little column that's written by you.)

Unfortunately, each issue we scratch for good tips. I'd like the column to be the longest (okay, the second-longest) in *Micro C*. To do that we need your help.

The biggest question: *what makes a good tech tip?* Send us things you've discovered about: servicing PCs, putting together new hardware, locating cheap parts, surprises you've found in common assemblers or compilers, and things you do to make your work easier.

Send them to:

Tech Tips
Micro C Dept. Of Information
P.O. Box 223
Bend, OR 97709

Send disk, paper, or both. Or leave a file or message on the Micro C BBS, (503) 382-7643.

You'll be glad you did, because Tech Tip authors get: three extra copies of the magazine (at least one of those goes to your mother), and an author's T-shirt. (Non-authors have resorted to bribery in their attempts to get one of these exclusive chest covers.)

Theme Issues Can Also Be Interesting

After announcing the themes of upcoming issues, we've gotten wonderful response from folks working in those areas. For instance, an automated milling machine just offered to do a piece for the robotics issue (it's about a cousin of his who welds underbodies for GM). On the other hand, a number of you are waiting for a theme issue that fits your expertise.

Don't wait.

We're not limited to theme articles in theme issues. One of our strengths is our mix of the weird and the arcane. So, for instance, we have hardware articles in this (software tools) issue. And, of course, we ran the "Assembler Is Better Than C" article in the C issue. (Boy, did that generate mail.)

Anyway, if you have ideas, call us. Somebody's got to make these theme issues interesting.

The Theme Issues Are Already Interesting

Robotics is going to be hot. I mean *HOT*. I'm talking front burner. Loren Heiny is working on an article on visual recognition, and Bob Namsel is finishing up a cheap ($300) robot complete with an ME (Mail-in EPROM). Based on the 64180 (Z80 look-alike), it'll be a maze runner complete with IR range sensors, bumper switches, solid state compass, and so on.

He's proposing that groups build a single robot and have members design and burn ROMs. Each group will hold competitions as members try out their own ROM programming prowess.

Then, the top ROMs from each group would be sent to Bob for the national runoff (amazing). The best ROM wins.

Official Programmer's Diet

If you've been programming any length of time (i.e. since midnight, last Friday) you know the importance of maintaining your health. So you carefully plan your diet around the seven hacker food groups — twinkies, cola, coffee, chips, chocolate, cold pizza, and Rolaids.

However, I've got to pass along some of what I'm hearing in *Aviation Consumer*, a small (but expensive) newsletter on flying.

Pilots have to watch their health. They're not supposed to have seizures, fainting spells, heart attacks, migraines, shortness of breath and such while they're flying. Distracts from the job. Every year or two (depending on type of license), each pilot gets a physical and during the exam he gets checked for a whole list of conditions, any of which would send him to the unemployment office.

We all know about chocolate, chips, and pizza. They're good artery pluggers. But that's pretty slow and painless. Now, it appears, some pilots are having trouble with NutraSweet, the sugary tasting stuff residing in just about every "reduced calorie" product on the market.

The problem came to light when a pilot wrote in saying he'd spent several years and about $40,000 being tested. He'd had a whole list of problems (including seizures) that had ended his flying. He was a mess. Almost by accident he stopped drinking six bottles of diet cola a day and stopped using all other diet foods. In a week he was a new man.

Well, a NutraSweet official responded immediately, saying this person's experience wasn't scientific proof of a problem with their sweetener.

Then a physician followed up with a long treatise on the chemical makeup of NutraSweet and the possibility of allergic reactions. Another physician added a look at the effects of large doses of caffeine. (Many of the pilot's complaints matched the side effects of caffeine.)

Meanwhile, another pilot who'd been having similar health

problems reported he'd stopped using dietetic gum and candy after reading the original letter. He reported that his problems, too, disappeared after a week.

So, if you're weirder than you'd like to be (or weirder than someone close to you would like), you might want to modify your diet slightly. (Chitlins and sprouts with salsa?)

Hard Drive Relief

"Hello, Micro C?"

Yep.

"My 20 meg hard drive hasn't been booting dependably."

Yep.

"Should I get a new drive?"

Yep.

Actually you probably don't need a new drive — you just need to give the drive you have a little help. Most drive problems show up after a year or so, and usually it's because the unit's having a harder and harder time reading data. (Usually the head's in one place, the track's in another.) The only way to straighten things out is to do a real, low level, reformat.

Unfortunately low level formats are a pain. You back everything off the hard drive onto ten zillion little floppies, run DEBUG to get into the controller's formatter, run FDISK, run FORMAT, and then copy all that data back onto the hard drive. I'm talking hours, assuming everything works properly.

In issue #42 I wrote an article titled "Keeping Your Hard Drives Running." In it I mentioned a package called Disk Technician. Disk Technician watches for recoverable data errors. (Errors which MS-DOS ignores.)

When the package finds a track with errors, it stores the corrected information in memory, reformats the track, and then rewrites the data. You're supposed to run the program every day and over a year you'll probably get a pretty complete low level reformat (as it's needed).

I had two problems with Disk Technician. It was copy protected, and it worked best when run everyday. Also, it was designed for people who don't want any choice about what happens (micro-power users). (I should add that I just received a new copy of Disk Technician and they've upgraded it a bit, made it run on more machines, and they've removed the copy protection.)

Anyway, in that article I also talked about H-Test and H-Format. H-Format tests your hard drive for optimum interleave and then does the optimum low-level format without wiping out data. Wonderful package.

SpinRite

A month ago I got a copy of SpinRite. It's a pretty complete combination of Disk Technician and H-Format.

It'll tell you the optimum interleave and do a low level reformat using the interleave of your choice (without wiping out data). It'll thoroughly test your hard drive surface and (if you tell it to) even recover tracks that you locked out during the original format. Plus, it'll recover data from bad sectors (if possible).

It has super displays — tells you a lot about your hard drive — is trivial to use — detects, displays, and fixes bit errors (if possible) — and it runs fast. It's hard drive insurance and a lot more. (Plus it's not copy protected.) Disk Technician and H-Format sell for $99.95 each so you'd pay nearly $200 for the pair. SpinRite will cost you $69.95.

One note. Use DISKCOPY to make one or more copies of the SpinRite master disk before doing anything. Then use the copy. If you ever use the master to create a working disk, you'll only be able to use SpinRite on that system. (You'll find the same instructions in the installation section of the manual.)

Gibson Research
22991 La Cadena
Laguna Hills, CA 92653
(714) 830-2200

Better Graphics

As long as I'm talking about things I'm excited about I might as well tell you about three new graphics packages.

Adobe Illustrator has been running on the Mac just about forever and I'd heard serious rumors (press releases) that they'd be showing their PC version at Comdex.

I looked for Adobe. No Adobe. I made a beeline for the Apple room. No Adobe. (They were there last year.) I asked the Apple staffers where I could find Adobe.

"Adobe who?"

How quickly they forget. Illustrator must have sold thousands and thousands of Macs.

However, Digital Research (you remember them, the guys who wrote CP/something) snuck in with an announcement of Artline. I don't know Illustrator well enough to know whether Artline is better or worse, but it looks wonderful. (In the manual they show you how to draw an apple — draw your own conclusion.)

Anyway, it lets you bend type, create shadows, sketch over scanned images (you can't edit scanned images because Artline manipulates vectors, not pixels), draw, and fill, and all those fancy things. (You have to have expanded memory to store the pixel images you're going to trace.)

We've had a similar package called Designer, but Sandy and Carol find it very difficult to use.

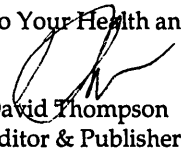
Finally, Arts & Letters also announced their Editor package. It's very, very similar to Artline but includes a much larger library of vector drawings.

Artline runs under Gem. Arts & Letters runs under Windows.

We've received both Artline and Editor and Sandy used both in this issue. Artline generated the Go board on page 96 (upper left-hand corner) and the diagram on page 37. The Arts & Letters Editor generated the books on page 95 (lower left-hand corner) and the illustrations on pages 33, 53, and 54. She had almost no trouble figuring out either program. (We have a more complete report in the works.)

(Carol used Publisher's Paintbrush to create and edit the robot on page 95. Paintbrush works very well for editing scanned and other pixel images.)

To Your Health and Your Hard Drive,



David Thompson
Editor & Publisher

Continued from page 22

Figure 3 — ISRHT.C isr handler template.

```
#asm

dataseg         segment para public 'data'
        extrn           _Dorg_:byte

        public  _ISRHT_LENGTH_, _ISRHT_PTR_

_ISRHT_LENGTH_    dw (offset p2 - offset p1)
_ISRHT_PTR_       dw offset p1
                  dw seg p1

dataseg ends

        ;This code gets moved later to a structure in RAM.
        ;The correct SS and SP values, as well as the
        ;address of the C function to call, are installed
        ;at that time.

codeseg         segment para public 'code'

p1:
        public  isr_template
isr_template proc far

                ;flags
                ;CS
                ;IP
        push    bp              ;save the callers registers
        push    si
        push    di
        push    ds
        push    dx
        push    ax
        push    cx
        push    es
        push    bx

        mov     ax, seg _Dorg_  ;get local data segm value
        mov     ds, ax
        mov     es, ax

        mov     ax, 0           ;push a pointer to the ISRH
        push    ax
        mov     ax, 0           ;0s will be replaced later
        push    ax

        call    isr_template    ;isr_template also replaced

        pop     ax              ;restore pointer we pushed
        pop     ax
        pop     bx              ;restore callers registers
        pop     es
        pop     cx
        pop     ax
        pop     dx
        pop     ds
        pop     di
        pop     si
        pop     bp
        jmp     isr_template    ;jump to previous handler

isr_template endp
        ;
p2:

codeseg ends

#endasm

***
```

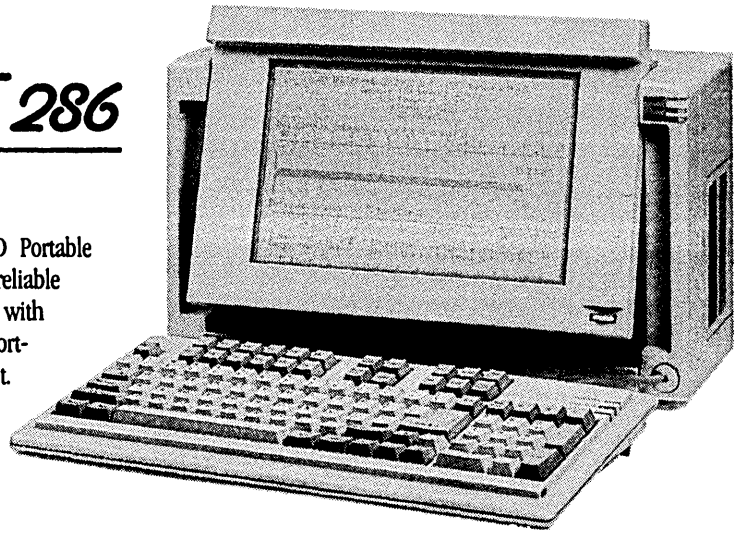

Continued on page 76

McTek

LCD Portable

(Also available in PLASMA PORTABLE)
(Including Hard Disk Only 19 lbs.)

RABBIT 286



The McTek Rabbit-286 LCD Portable combines the fastest, most reliable AT motherboard available with most visible full-size LCD portable screen on the market. Running at a switchable 8 or 10 MHz Ø wait state, it includes a 20MB hard disk, 720KB 3½" floppy drive, parallel & serial ports, Award 3.03 bios, 640k & turbo indicator LCD. The screen is a fantastically readable, electroluminescently backlit, 80-column by 25-line, high resolution 640x400 super twisted LCD with adjustable intensity and screen-angle. The screen size is 9.5"x6". It's as readable as a CRT. You can also plug in a digital or analog color monitor or a digital or composite monochrome monitor. Included also is an external 5¼" floppy port for reading and converting to 3½" disks (5¼" external drive w/case: $179 when purchased with LCD Portable). The McTek Rabbit-286 LCD Portable comes fully assembled with our one-year parts & labor guarantee, and sells for an amazing, complete price of only 386-20 MHz W/1MB

$1799!

$2799!

$1399.00
12MHz/0 Wait State

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

Options:..... Call

$2099.00
16MHz/0 Wait State

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 (28MS)**

Options:..... Call

$2399.00
20MHz/0 Wait State

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

McTek Systems, Inc. • 1411 San Pablo Avenue • Berkeley, CA 94702 • 415-525-5129

DISK DRIVES

Fujitsu 360k	$69
Fujitsu 1.2MB	$89
Teac	$75
Teac 1.2MB	$95
Toshiba 3½" 720K	$89
Teac 3½" 1.4MB	$105
20MB Hard Disk Kit	$279
30MB Hard Disk Kit	$309
ST-225	$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

Everex int. 300/1200	$79
Everex 2400 external	$195
Everex 2400 internal	$179

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

McTek286-20MHz	$449
Baby McTek 286B-AT 8/10 O-wait	$249
McTek 386-16MHz	$859
McTek 386-20MHz	$899
McTek 386-24MHz	$999
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 100MHz 640k 2 Drive System	$759

Reader Service Number 42

Continued from page 74

Figure 4 — ISR.H support for interrupt handlers

```
#ifndef uchar
#include <defs.h>
#endif

#ifndef OP_IRET
#include <cpu.h>
#endif

#ifndef AX
#include <call.h>
#endif

#define ISR_HDR_LOADED

/*        REGS  - Structure of a register file.

        A far pointer to this structure is passed to the
        interrupt handlers by the traps. The CPU registers
        are also passed in the order shown below.

        It's therefore possible to declare a handler like:

           void far int_24_handler( isr, r1, r2 )
           isrh *isr;
           _ _pregs r1, r2;
           {}

        Which is much easier than the alternative...
        This only works if your compiler supports passing
        structures by value (By now, there probably aren't
        many that don't...)
*/

typedef struct ISR_REGS {
  _ _regdp esbx;
  _ _reg c;
  _ _reg3r dsdxax;
  unsigned int di;
  _ _regdp bpsi;
  _ _pcreg pc;
  _ _regpsw psw;
} isr_regs;


/*        The code to intercept an interrupt and call a C
        function is copied into a structure of the follow-
        ing type for each handler that's needed. The
        required patches to the code are then made.
        isr_install() sets up the interrupt vectors, and
        returns. isr_restore() restores the previous
        handler, and isr_pass() calls the previous
         handler.
        The code for this trap is in ISRHT.C.
*/

typedef struct ISRH {
   uchar code1[17];
   uint isrhs;
   uchar code2[2];
   uint isrho;
   uchar code3[2];
   void far (*hndlr)();
   uchar code4[11];
   uchar code_jmpf;
   void far (*prev_hndlr)();
   int isr_num;
   int count;
   void far *stack;
} isrh;

#define isr_pass(i) ( (i)->code_jmpf= i->prev_hndlr ?
                        OP_JMPF : OP_IRET)
#define isr_iret(i) ( (i)->code_jmpf= OP_IRET)
#define isr_retf(i) ( (i)->code_jmpf= OP_RETF)

void far *isr_get_vector(int inum);
void isr_set_vector(int inum,void far (*farfuncp)());
void isr_install(isrh *isr,int isr_num,
void far (*handler)());
void isr_restore(isrh *isr);

/***/
```

Figure 5 — ISR.C support code for ISRs

```
#include <isr.h>


#pragma vpindex isr_get_vector
void far *isr_get_vector( inum )
int inum;
{
    return (void far *)(*((long far *)((long)
     (inum * 4))));
}


#pragma vpindex isr_set_vector
void isr_set_vector( inum, farfuncp )
void far (*farfuncp)();
int inum;
{
#asm
        mov     ax,word ptr 4[bp]
        shl     ax,1
        shl     ax,1
        mov     dx,0
        mov     es,dx
        mov     bx,ax
        mov     dx,word ptr 8[bp]
        mov     ax,word ptr 6[bp]
        cli
        mov     es:word ptr 2[bx],dx
        mov     es:word ptr 0[bx],ax
        sti
#endasm
}


#pragma vpindex isr_install
void isr_install( isr, isr_num, handler )
isrh *isr;
int isr_num;
void far (*handler)();
{
        extern int _ISRHT_LENGTH;
        extern unsigned char far *_ISRHT_PTR;
        int i;

        setmem( isr, sizeof *isr, 0 );
        /* use long form of movmem() to avoid
           problems with the different memory models...
        */
        for ( i = 0; i < _ISRHT_LENGTH; i++ )
           ((char *)isr)[i] = _ISRHT_PTR[i];

        isr->hndlr = handler;
        isr->isr_num = isr_num;
        isr->isrho = ((long)((void far *)isr)) & 0xffff;
        isr->isrhs = ((long)((void far *)isr)) >> 16;
        isr->prev_hndlr = isr_get_vector( isr_num );
        if ( !isr->prev_hndlr )
           isr->code_jmpf = OP_IRET;

        isr_set_vector( isr_num, (void far (*)())isr );
}


#pragma vpindex isr_restore
void isr_restore( isr )
isrh *isr;
```

```
{
        isr_set_vector( isr->isr_num, isr->prev_hndlr );
}


#ifdef MAIN

volatile long c;
volatile int k;

_ _regs _REGS;


main()
{
        isrh isr1, isr2, isr3, isrx;
        void far int_1c(), int_09(), int_23();
        int ascii;

        seg86( _REGSP );
        _ _regs_dump( _REGSP );

        isr_install( &isr1, 0x1c, int_1c );
        isr_install( &isr2, 0x09, int_09 );
        isr_install( &isr3, 0x23, int_23 );

        printf("\nBet ya can't ^C, ^BREAK or \
                CTRL-ALT-DEL out'a this one!\n");
        printf("(hit ESC when you give up!)\n\n");

        ascii = 0;
        do
        {  printf("Time: %lds, Last Scan code: %x    \r",
                   c * 101 / 1821, k );
           if ( constat() )
              ascii = conin();
        } while ( ascii != 27 );

        isr_restore( &isr1 );
        isr_restore( &isr2 );
        isr_restore( &isr3 );
}


int conin()
{
        pcdos( 0x08 );
        return _AL;
}


int constat()
{
        pcdos( 0x0b );
        return !!_AL;
}

void far int_09( isr, _REGS )
isrh *isr;
isr_regs _REGS;
{
        static int ctrl;

        k = inportb( 0x60 );

             if ( k == 29 )     /* control key pressed */
                ctrl = 1;
        else if ( k == 29+0x80 )/* control key released*/
                ctrl = 0;
        else if (ctrl && (k == 46 || k == 70 || k == 83))
             {  /* don't allow ^C, ^BREAK or ^DEL */
                outportb( 0x61, inportb( 0x61 ) | 0x80 );
                outportb( 0x61, inportb( 0x61 ) & 0x7f );
                outportb( 0x20, 0x20 );
                isr_iret( isr );
                return;
             };
        isr_pass( isr );
}

void far int_1c( isr, _REGS )
isrh *isr;
isr_regs _REGS;
{
        c++;
        isr_pass( isr );
}

void far int_23( isr, _REGS )
isrh *isr;
isr_regs _REGS;
{
        _CARRY = 0;
        isr_iret( isr );
}

#endif

***
```

Figure 6 — CPU.H CPU related #defs etc.

```
/*        8086 cpu opcodes
*/

#define OP_JMPF  0xea      /* far jump instruction     */
#define OP_CALLF 0x9a      /* far call instruction     */
#define OP_RETF  0xcb      /* far return instruction   */
#define OP_INT   0xcd      /* software int instr.      */
#define OP_IRET  0xcf      /* return from interrupt    */

/*        CPU flag register bit masks
*/

#define FLG_RESf        0x8000    /* unused            */
#define FLG_RESe        0x4000
#define FLG_RESd        0x2000
#define FLG_RESc        0x1000
#define FLG_OF          0x0800    /* overflow          */
#define FLG_DIR         0x0400    /* 0=inc, 1=dec      */
#define FLG_INTE        0x0200    /* Int Enable        */
#define FLG_TRAP        0x0100    /* 1=single step     */
#define FLG_SIGN        0x0080    /* 1=result NEG      */
#define FLG_ZERO        0x0040    /* 1=result ZERO     */
#define FLG_RES5        0x0020
#define FLG_AUXC        0x0010    /* Aux carry flag    */
#define FLG_RES3        0x0008
#define FLG_PE          0x0004    /* 1=even # 1 bits   */
#define FLG_RES1        0x0002
#define FLG_CARRY       0x0001    /* Carry Flag        */

#asm
        FLG_OF          equ     0800h
        FLG_DIR         equ     0400h
        FLG_INTE        equ     0200h
        FLG_TRAP        equ     0100h
        FLG_SIGN        equ     0080h
        FLG_ZERO        equ     0040h
        FLG_AUXC        equ     0010h
        FLG_PE          equ     0004h
        FLG_CARRY       equ     0001h
#endasm


/*        Structure of the PSW register...

WARNING:

"There are a number of caveats that apply to (bit)
fields. Perhaps most significant, fields are assigned
left to right on some machines and right to left on
others, reflecting the nature of different hardware.
This means that although fields are quite useful for
maintaining internally-defined data structures, the
question of which end comes first has to be carefully
considered..."
    - K&R, page 138.
*/

typedef struct _ _FLAGS {
   unsigned carry : 1;
   unsigned res1 : 1;
   unsigned pe : 1;
```

Continued on page 78

Continued from page 77

```
    unsigned res3 : 1;
    unsigned auxc : 1;
    unsigned res5 : 1;
    unsigned zero : 1;
    unsigned sign : 1;
    unsigned trap : 1;
    unsigned inte : 1;
    unsigned dir : 1;
    unsigned of : 1;
    unsigned resc : 1;
    unsigned resd : 1;
    unsigned rese : 1;
    unsigned resf : 1;
} _ _flags;

/***/
```

Figure 7 — REGSDUMP.C dumps a _ _regs structure

```
#include <call.h>

#define err_printf printf

#pragma vpindex _ _regs_dump
void _ _regs_dump( rp )
_ _regs far *rp;
{
#define _REGS (*rp)

     err_printf( "\ax-%04x  cx-%04x  si-%04x  bp-%04x \
     ds-%04x pc-%04x:%04x pswh-%02x %c%c%c%c %c%c%c%c\n\
     bx-%04x  dx-%04x  di-%04x           es-%04x  \
     stk-%04x:%04x     l-%02x %c%c%c%c %c%c%c%c\n",

               _AX, _CX, _SI, _BP, _DS, _CS, _IP,

               _PSW >> 8,
               rp->psw.bit.resf ? '1' : '.',
               rp->psw.bit.rese ? '1' : '.',
               rp->psw.bit.resd ? '1' : '.',
               rp->psw.bit.resc ? '1' : '.',
               _OF    ? 'o' : '.',
               _DIR   ? 'd' : '.',
               _INTE  ? 'i' : '.',
               _TRAP  ? 't' : '.',

               _BX, _DX, _DI,      _ES, _SS, _SP,

               _PSW & 0xff,
               _SIGN  ? 's' : '.',
               _ZERO  ? 'z' : '.',
               rp->psw.bit.res5 ? '1' : '.',
               _AUXC  ? 'a' : '.',
               rp->psw.bit.res3 ? '1' : '.',
               _PE    ? 'p' : '.',
               rp->psw.bit.res1 ? '1' : '.',
               _CARRY ? 'c' : '.' );
}

/***/
```

Figure 8 — CALLER.C does far calls

```
#asm

codeseg segment para public 'code'

        public  _AX_,_BX_,_CX_,_DX_
        public  _AL_,_AH_,_BL_,_BH_,_CL_,_CH_,_DL_,_DH_
        public  _SI_,_DI_,_BP_
        public  _ESBX_, _DSDX_, _DXAX_, _BPSI_
        public  _DS_,_ES_
        public  _PC_,_IP_,_CS_
        public  _PSW_
        public  _STKPTR_,_SP_,_SS_

_ESBX_  label   dword
_BX_    label   word
_BL_    db      ?
_BH_    db      ?

_ES_    dw      ?

_CX_    label   word
_CL_    db      ?
_CH_    db      ?

_DXAX_  label   dword
_AX_    label   word
_AL_    db      ?
_AH_    db      ?

_DSDX_  label   dword
_DX_    label   word
_DL_    db      ?
_DH_    db      ?

_DS_    dw      ?

_DI_    dw      ?

_BPSI_  label   dword
_SI_    dw      ?
_BP_    dw      ?

_PC_    label   dword
_IP_    dw      ?
_CS_    dw      ?

_PSW_   dw      ?

_STKPTR_        label   dword
_SP_    dw      ?
_SS_    dw      ?

sav_ss  dw      ?
sav_sp  dw      ?

codeseg ends

#endasm

#pragma vpindex caller
unsigned int far caller() /* returns CPU FLAGS */
{
#asm
        push    bx                  ;save the callers regs
        push    cx
        push    si
        push    di
        push    bp
        push    ds
        push    es

        mov     sav_ss, ss
        mov     sav_sp, sp

        mov     ds, _DS_
        mov     es, _ES_

        mov     ax, _AX_
        mov     bx, _BX_
        mov     cx, _CX_
        mov     dx, _DX_

        mov     si, _SI_
        mov     di, _DI_
        mov     bp, _BP_

        pushf   ;in case we're faking an interrupt
        call    dword ptr _PC_

        mov     _AX_, ax
        mov     _BX_, bx
        mov     _CX_, cx
        mov     _DX_, dx

        mov     _SI_, si
        mov     _DI_, di
        mov     _BP_, bp
```

```
        mov     _DS_, ds
        mov     _ES_, es
        mov     _SP_, sp
        mov     _SS_, ss

        pushf
        pop     ax
        mov     _PSW_, ax

        mov     ss, sav_ss
        mov     sp, sav_sp

        pop     es
        pop     ds
        pop     bp
        pop     di
        pop     si
        pop     cx
        pop     bx
#endasm
}

/***/
```

Figure 9 — CALLER.H header for CALLER modules

```
#ifndef ISR_HDR_LOADED
#include <sr.h>
#endif


/*        CPUREGS - Variables used as CPU reg copies.

        The module "CALLER.C" contains the function
        caller(), and a variable list which is in its
        code segment. Caller() loads the CPU registers
        with the vars, does the call, then loads the
        vars with the CPU registers. The way the vars
        are defined, their contents can be accessed in
        a number of ways; ergo the declarations below:

        The macros int86(), call86() & pcdos() also use
        caller().
*/

extern unsigned char far _AH,_AL,_BH,_BL,_CH,_CL,_DH,_DL;
extern unsigned int  far _AX,_BX,_CX,_DX,_SI,_DI,_BP;
extern unsigned long far _DXAX;
extern unsigned char far * far _BPSI;
extern unsigned char far * far _DSDX;
extern unsigned char far * far _ESBX;
extern unsigned int  far _DS, _ES;
extern          void far (* far _PC)();
extern unsigned int  far _IP, _CS, _PSW;
extern unsigned char far * far _STKPTR;
extern unsigned int  far _SP, _SS;

unsigned int far caller(void);

/* the following spaghetti is a work-around... */
#define _ _xx1(i) (+(ulong)((i) * 4))
#define _ _xx2(i) ((long far *)_ _xx1((i)))
#define _ _xx3(i) (*_ _xx2((i)))
#define isr_get_vec(inum) \
((void far (*)())_ _xx3((ulong)(inum)))

#define call86(farfunc) ( _PC = (farfunc), caller() )
#define int86(inum) ( call86( isr_get_vec( (inum) ) ) )
#define pcdos(func) ( _AH = (func), int86(0x21) )

/***/
```

Letters

Continued from page 6

might be a first. People who want a sense of perspective should read the article.

Don Taylor
13628 SE 20th Circle
Vancouver, WA 98684

SEA Defended

I normally welcome the delivery of my *Micro C* and read it voraciously from cover to cover. I appreciate your irreverent editorial style and make good use of the valuable information in your magazine. I feel compelled to write to you now because your irreverent style has become irresponsible. I'm talking about the Shareware column by Anthony Barcellos in issue #45.

The amount of misinformation flying around about Software Enhancement Associate's suit against PKWare is appalling. That you would allow the type of idle, misinformed speculation that characterized Mr. Barcellos' column is also appalling.

I don't know all the facts surrounding the suit between SEA and PKWare, but I do know that, quite properly, SEA has not entered the fray while a court action was pending. At the same time, Mr. Katz has apparently mounted quite an effort to try his case on the bulletin boards rather than in the courts.

The issue, as I understand it, is not whether Mr. Katz has developed a more efficient algorithm — it is one of theft, pure and simple. Mr. Katz was alleged to have stolen large amounts of copyrighted code from SEA without signing a licensing agreement.

I say allege because no one will ever really know what the truth of the matter is in this case. SEA (which, incidentally, is *not* a "much larger" entity than PKWare, as Mr. Barcellos claims) had an expert witness examine source code from ARC and from the PK series of programs.

The expert witness testified in court that substantial portions of the PK code, portions that had not, apparently, been released to the public, appeared to be identical to SEA's code. It was just after this witness had testified that, as I understand the situation, Mr. Katz approached SEA's lawyers and asked for an out of court settlement.

Again, as I understand it, it was Mr. Katz who insisted on language absolving him of any wrongdoing and who also insisted that portions of the settlement be sealed. It is indeed unfortunate that SEA is not a larger company with the resources to see the suit through to its bitter end. That way I would imagine a lot of questions, both legal and otherwise, would have been settled by the public record.

All of this, of course, is speculation. I would, however, claim that SEA has performed a great service to the microcomputing community. The concept of self-compressing archives is ingenious and has taken the bulletin board scene by storm. SEA is attempting to maintain a standard — one which will, eventually, operate on many different computers and operating systems.

Is SEA hogging the market? I would say no. SEA asks for no registration or money from private users. PKWare does. SEA publishes their source code so that programmers wishing to build utilities consistent with SEA algorithms may do so. PKWare does not.

Quite rightly, SEA asks for a licensing agreement to use that code in a commercial program. I have no problems with this — it is, after all, their intellectual property. In short, I feel that PKWare is the company that has done a grave disservice to the DOS community.

Further, I feel that responsible publications such as yours and *PC World* should go out of their way to cover both sides of an explosive issue such as this one fairly and evenly. Put the shoe on the other foot, Mr. Thompson. If I were to take your magazine and resell it with a different cover, one which was perhaps more appealing to the mass of appliance users out there, wouldn't I hear from your attorneys rather quickly?

I would hope in the interest of good journalism that you would cover issues more fairly in the future, or not at all.

Norman C. Saunders
The Osprey's Nest BBS
(301) 989-9036

Editor's note: Thank you for writing, Norm. I understand that SEA will be making a public statement now that things have been settled legally. (The statement should make the next issue of Micro C.*) And, on the other side, I have contacted the PK folks for additional information on Tony's column. So far, they haven't responded.*

Maxi/PC PCB Update

Since I wrote "Choosing a PCB Layout System" in *Micro C* issue #45, I've gotten some experience with the Racal-Redac Maxi/PC PCB layout package. To fix the problems that I was having with the software protection device (key), I had to purchase a second serial I/O card and dedicate it to the key. This was a major annoyance at the time and delayed me about a week from getting started with the package.

As a test for Maxi/PC, I entered the PD32 circuit. For you newcomers, a PD32 is a 32016 coprocessor board that plugs into an XT bus and runs UNIX. The board and software were featured at SOG V and in the October/November 1986 issue of *Micro C*.

The PD32 was designed by George Scolaro and the firmware and software were written by Dave Rand. The hardware design was donated to the public domain; the software port and UNIX, of course, were not.

The PCB is a full size XT board with 22 ICs and 8 256x8 SIPRAMs (for a total of 2 Megabytes on-board). It's a moderately dense layout and a good test for an autorouter.

I placed the ICs more or less in the same locations George chose, but moved a few of the smaller ICs to locations which gave a better looking ratsnest. Before autorouting, I allowed Maxi/PC to swap pins and gates. The PALs, in particular, were pin-swapped quite extensively. Therefore, the board is not wired exactly the same as George's board, but hopefully it's equivalent.

The results? Well, there's good and bad. On the good side: Maxi/PC routed all but 25 of the 662 connections on the board, and accomplished this in about 20 minutes on a 10 MHz AT clone with a 10 MHz 80287 math coprocessor.

However, those last 25 connections took me about eight hours. This is because a lot of the autorouted traces had to be rerouted since the autorouter had painted itself into a corner. Perhaps a rip-up and retry router could have completed the job.

I really have to give George a lot of credit. His layout had only about 75 vias and his traces follow data and

address paths in a very regular pattern. His traces flow in vertical, horizontal and diagonal directions on both sides of the board.

The Maxi/PC autorouter, on the other hand, uses an orthogonal router. Any time it wants to change direction, it drops in a via and continues on the other side of the board. Vertical traces are on one side of the board and horizontal traces are on the other side. As a result, the autorouter generated 277 vias!

Why is this important? Mainly, the issue is cost. There is a per-hole cost when fabricating a PCB. So the more vias, the more expensive the board will be to produce. For low volume production, the extra vias may be acceptable.

The PD32 project was one which generated a lot of excitement at the time, but never really got off the ground. I think this was partly due to the cost of building a system.

The parts cost for a 2 Megabyte board was about $400 (most of the cost in the memory chips) and the UNIX single user licence was another $500. This is cheap for a UNIX system, but apparently it was too expensive for the hobbyist market at which it was aimed.

At this point I have no plans to fabricate the PCB layout described above. However, I would like to revive the spirit of the PD32 and I'd like to correspond with anyone who has any ideas for a project like this.

Scott Baker
18185 West Union Rd.
Portland, OR 97229

Editor's note: Scott and I talked for quite a while about the possibilities for a high-performance public domain UNIX system. I suspect that a 68020 or 68030-based board with space for 4 megs of RAM would make a fine multi-user UNIX platform. There are several variations of UNIX, including some free versions. Contact Scott if you're itching to do an operating system port for a new piece of hardware.

Micro Museum

I just read my first issue of *Micro Cornucopia* and am impressed almost beyond words! Finally, a magazine that's not afraid to talk seriously about homebrew hardware and with scarcely a word on the latest and greatest software for non-technical users. Be forewarned — most computer magazines I have liked well enough to subscribe to have failed within 90 days. May you be spared this curse.

Several months ago I counted noses and found I owned 27 computers. Shortly thereafter I realized that my self-assigned task in life is to save as much hardware and documentation as possible defining the early years (pre-8088/32000) of the "home" computer.

To date I've added a number of units and hundreds of books and periodicals. I'm sufficiently serious about this to have purchased land for a museum I hope to build in 1990/1991. I can't believe the collection will be complete without all back issues of *Micro Cornucopia*.

Perhaps you have readers who would like the gift of immortality. I would be most grateful for help with my collection. The more obsolete, underpowered, and obscure the computer the better.

I'm especially interested in homebrew, kitbuilt or hacked commercial systems — preferably accompanied by something of the unit's history and the owner's experiences. Obviously, I also have an interest in periodicals and books from the Golden Era. Can anyone help?

Vernon L. Goodwin
7917 Douglas Dr.
Charlotte, NC 28217

Shipping
c/o Support Systems International
8700 Suite K Red Oak Blvd.
Charlotte, NC 28217

Editor's note: Find a large lake on a windy day and look at what's tethering the boats. At the end of those waterlogged ropes you'll find 8" drives, linear power supplies, and S-100 mother boards. Best of luck with your search.

By the way, I'm looking for a few good copies of Micro C *issue #27 myself. It turns out I didn't put any aside before they sold out.*

♦ ♦ ♦

CP/M NOTES

Who's Alive In CP/M?

Part II

By Samuel Vincent
CP/M Special Interest Group
Pittsburgh Area Comp. Club
P.O. Box 6440
Pittsburgh, PA 15212

Last issue we presented the first part of this list of active CP/M commercial software vendors (compiled by Samuel and his friends at PACC). Read on for the rest of that list along with public domain software suppliers and Z-Nodes that sell CP/M products.

As before, a complete version (with prices when available) lives on the Micro C BBS and the Issue #46 disk. Please send any corrections or additions to this list by writing to the attention of Samuel Vincent at the address shown above. You can also phone at (412) 845-8613 (evenings only).

Emerald Microware
P.O. Box 1726
Beaverton, OR 97075
(503) 641-0347
* Verified 8/31/88 *

Dealer-upgrades, add-ons for Kaypro, Xerox, Big Board, etc. QP/M, MicroSolutions products, and others.
Free catalog available.

Everton Publishers
P.O. Box 368
Logan, UT 84321
(800) 453-2707
(801) 752-6022
* Verified 8/15/88 *

MY FAMILY RECORD genealogy program
Free brochure available

EZ Systems, Inc.
P.O. Box 23190
Nashville, TN 37202
(615) 269-6428
* Verified 8/31/88 *

Church membership/ finance software.
Demo vers available.
DateMate by FreshWare
Free catalog available.

Flashlite Software
P.O. Box 3535
Daly City, CA 94015

FLASHPRINT
Printing utility

FreshWare Creative Sftwre
522 Glenpark Dr.
Nashville, TN 37217
(615) 360-6181
* Verified 8/31/88 *

DateMate - calendar program, appointment organizer and telephone list.
Free brochure available

Future Communications
128 Midway Rd.
Decatur, GA 30030-4429
(404) 373-4831
* Verified 4/15/88 *

CHECKZ - Checking account maintenance
Free brochure available

HURD Computer Systems
6330 Lincoln Ave.
Cypree, CA 90630
(714) 525-0879
* Verified 7/29/88 *

PRO MAIL, QWIZ WRITER, HOME INVENTORY PLUS, RECIPE INDEX.
Free brochure available

Hoyle & Hoyle Software,Inc.
111 Sparrow Dr.
Isle of Palms, SC 29451
(803) 886-5802
* Verified 8/15/88 *

QUERY III Dbase Manager
QUERY III Calculator/ Report Writer. Adventure
Free brochure available
Leave msg on tape.

Irata Software
2562 East Glade
Mesa, AZ 85204
(602) 892-0015
* Verified 4/15/88 *

ALIST and ALIST Plus - Simple database programs.

James River Group
125 North First St.
Minneapolis, MN 55401
(612) 339-2521
* Verified 4/15/88 *

Accounting - G/L, A/R, A/P, Payroll, and Inventory. TMAN-General data file manager.
Free catalog available.

Kamasoft
P.O. Box 5549
Aloha, OR 97007
(503) 649-3765
* Verified 8/15/88 *

OUT THINK - outline processor program.
Free brochure available
Leave message on tape.

L/Tek, Inc.
P.O. Box 2399
Davis, CA 95617
(916) 758-3630
* Verified 8/15/88 *

SPELLBINDER Word Processor. Scientific and Desk-Top Publisher versions also.
Free brochure available

Lexisoft, Inc.

See L/Tek, Inc.

Lionheart
P.O. Box 379
Atburg, VT 05440
(514) 933-4918
* Verified 4/15/88 *

Statistical & Business programs.
Free catalog available.

Logic Associates
1433 W. Thome Ave.
Chicago, IL 60660
(312) 274-0531
* Verified 8/31/88 *

MEGABACK-hard disk backup. TutorI/O-learn BDOS I/O functions, etc
Free catalog available.

Magnolia Microsystems
2818 Thorndyke Ave. West
Seattle, WA 98199
(206) 285-7266
* Verified 7/29/88 *

Dealer-Health/Zenith software/hardware add-ons. Most CP/M software being sold out.
Free brochure available

Manx Software Systems
One Industrial Way
Eatontown, NJ 07724
(201) 542-2121
(800) 221-0440 outside NJ
* Verified 8/31/88 *

Aztec C II compiler in educational, pro, and commercial versions. Cross compilers to or from CP/M available.
Free brochure available

Mendocino Software
P.O. Box 1564
Willits, CA 95490
(707) 459-9130
* Verified 4/15/88 *

EUREKA - Disk Catalog program

MicroSolutions Inc.
132 West Lincoln Hwy.
DeKalb, IL 60115
(815) 756-3411
* Verified 4/15/88 *

UNIFORM - Disk conversion program.
UniDOS - Run CP/M programs on MS-DOS.
Free brochure available

MicroPro
Update Order Department
P.O. Box 7079
San Rafael, CA 94901-7079
(800) 227-5609 ext. 761
* Verified 7/29/88 *

WordStar Version 4.0
CP/M Edition

MIX Software
1132 Commerce Dr.
Richardson, TX 75081
(214) 783-6001
* Verified 7/29/88 *

MIX C compiler and linker. MIX Full Screen Editor.
Free brochure available

Mountain View Press, Inc.
P.O. Box 4656
Mountain View, CA 94040
(415) 961-4103
* Verified 4/15/88 *

Forth language products
Free catalog available.

NightOwl Software
Rt. 1 Box 7
Ft. Atkinson, WI 53538
(800) 648-3695
(414) 563-4013
* Verified 8/15/88 *

MEX Plus-communication software. MEX Pack-MEX Plus with terminal emulation and remote capability.
Free brochure available

O'Neill Software
P.O. Box 26111
San Francisco, CA 94126
(415) 398-2255
* Verified 4/15/88 *

ELECTRA FIND - Text retrieval.

Oasis Systems
6160 Lusk Blvd.
Suite C-206
San Diego, CA 92121
(619) 453-5711
* Verified 8/15/88 *

The Word Plus-spelling checker and Punctuation and Style - grammar checker.
Free brochure available

Paradigm Consultants
39243 Liberty St., Suite L
Fremont, CA 94538
(415) 794-8977
* Verified 4/15/88 *

MINI-BILLING, MINI-LEDGER, MINI-INVENTORY, TIME & BILLING, MINI-PAYABLES & RECEIVABLES

Plu*Perfect Systems
P.O. Box 1494
Idyllwild, CA 92349
* Verified 7/29/88 *
BBS 213 670-9465 leave msg for Bridger Mitchell.

BACKGROUNDER II, DATE-STAMPER, DosDisk and JetFIND. Z3Plus (a Z-System OS for CP/M Plus

Poor Person Software
3721 Starr King Circle
Palo Alto, CA 94306
(415) 493-3735
* Verified 4/15/88 *

WRITE HAND MAN-Resident desk accessory program
Note: Doesn't work with Wordstar v4.0 - does with earlier versions.

QuikData, Inc.
P.O. Box 1242
2618 Penn Circle
Sheboygan, WI 53081
(414) 452-4177
(414) 452-4345 (BBS - 300/1200/2400 bps)
* Verified 8/15/88 *

Dealer-Heath/Zenith software/hardware. Anapro, Software Toolworks, etc. Disk conversion available.

Free catalog available.

R & L Micro Services
P.O. Box 15955, Station F
Ottawa, Ontario
Canada, K2C 3S8
(613) 225-7904
* Verified 8/15/88 *

BOBCAT - Disk cataloging program.
Free brochure available.

Sage Microsystems East
1435 Centre St.
Newton, MA 02159
(617) 965-3552
(617) 965-7259 (BBS-300/1200/2400, PC-Pursuit)
* Verified 7/15/88 *

Dealer for NightOwl Software, Plu*Perfect Systems, SLR Systems, and Z-System software.

SLR Systems
1622 N. Main St.
Butler, PA 16001
(800) 833-3061
(412) 282-0864
* Verified 7/29/88 *

Z80 assemblers,linkers, librarian, and dis-assembler-full screen. CP/M software emulators and co-processor cards.
Free catalog available.

Software Research Technologies
c/o Heritage Software,Inc
3757 Wilshire Blvd.
Suite 211
Los Angeles, CA 90010

SMARTKEY - Key redef, SMARTPRINT - Printer Control, TOUCH 'N' GO - Typing tutor

Software Toolworks
13557 Ventura Boulevard
Sherman Oaks, CA 91423
(800) 223-8665 or (818) 907-6789 (Customer service)
* Verified 8/31/88 *

C, Lisp, games, utils, spreadsheet,editors etc
Free catalog available.

Somogyi Software
P.O. Box 1009
Redondo Beach, CA 90278
(213) 318-2769
* Verified 4/15/88 *

PUSH 'N' PULL - Outline processor. Version for Osborne

Spectre Technologies, Inc.
22458 Ventura Blvd., Ste. E
Woodland Hills, CA 91364
(800) 628-2828 ext 918
(818) 716-1655
* Verified 6/15/88 *

Desk accessory,
Sideways printing,
MEDIA MASTER,
REMBRANDT,
Business planner.

Spite Software
4004 SW Barbur Blvd.
Portland, OR 97201
(800) 237-9111
(503) 228-8238
* Verified 8/15/88 *

An eclectic assortment
of software/hardware
products for CP/M.
Free catalog available.

StatSoft Inc.
2832 East 10th St., #4
Tulsa, OK 74104
(918) 583-4149
* Verified 4/15/88 *

PSYCHOSTAT 3 - Data
analysis and statistics

SWC
P.O. Box 706
Santa Teresa, NM 88008
(505) 589-0999
(800) 862-2345 - at 2nd dial tone, dial 792
* Verified 7/29/88 *

FEDERAL INCOME TAX
SYSTEM. Tax Preparer's
version and yearly
updates available.

Free brochure available.

T/Maker Research Company
812 Pollaid Rd, Suite 8
Los Gatos, CA 95038
(408) 866-0127
* Verified 7/29/88 *

T/MAKER-Integrated
software/word processor
database, spreadsheet.
Also avail for TRS-DOS.
Free brochure available

The Software Store
706 Chippewa Square
Marquetta, MI 49855
(906) 228-7622
* Verified 8/15/88 *

Communication, disk
manager, disk editor,
integrated data
manager, program cross-
reference generator.
Free brochure available

Workman & Associates
1925 E. Mountain St.
Pasadena, CA 91104
(818) 791-7979
* Verified 8/15/88 *

FTL Modula-2,WRITE word
processor, BDS C, MITE-
80 telecommunications,
POURNELLE ACCOUNTING.
Free catalog available.

Worswick Industries
4898 Ronson Ct.
Suite H
San Diego, CA 92111
(619) 571-5400
* Verified 7/29/88 *

Osborne software w/docs
Personal Pearl,
SuperCalc 1 and 2,
dBASE II, Forth,
most software by
Spectre Technologies

Xpert Software
8865 Polland Ave.
San Diego, CA 92123
(619) 268-0112
* Verified 4/15/88 *

XTRAKEY - Key redef
XTRAPRINT-Printer fonts
SIDE 2 - Print sideways
XSCREEN - Copy screen
to printer or file

Zedcor Inc.
4500 East Speedway,Ste 22
Tucson, AZ 85712
(800) 482-4567
(602) 795-3996
* Verified 8/31/88 *

ZBASIC interpreter -
compiler.
Free brochure available

The following companies and user groups distribute Public Domain Software.

Boston CP/M User Group
c/o Susan Schluckebier
Forsyth Dental Center
140 Fenway
Boston, MA 02115
(617) 262-5200 ext. 342
* Verified 4/15/88 *

Public domain software.

The C Users' Group
P.O. Box 97
McPherson, KS 67460
(316) 241-1065
* Verified 6/15/88 *

Public domain C software
in source code form, for
CP/M, DOS and UNIX.
Catalog available.

Canada Remote Systems
4198 Dundas St. West
Toronto, Ontario
Canada M8X 1Y6
(416) 231-2383 (weekdays 5pm - 9pm)
* Verified 4/15/88 *

Free mini catalog
Complete catalog on disk

Comal User's Group, USA, Ltd.
6041 Monona Drive
Madison, WI 53716
(608) 222-4432
* Verified 4/15/88 *

Comal Programming Language
Information booklet and
product listing with
prices - both free.

FOG, International Computer User Group
P.O. Box 3474
Daly City, CA 94015
(415) 755-2000
* Verified 8/15/88 *

Public domain software
for CP/M. Complete
catalog on disk.

INCA
1249 Downing St.
P.O. Box 789
Imperial Beach, CA 92032
(619) 224-1177
* Verified 4/15/88 *

Public domain software
tested for compatibility
with the C-128.
Free catalog available.

International Software Library
c/o U.S. Computer Supply
511-104 Encinitas Blvd.
Encinitas, CA. 92024
(800) 669-2699
(619) 942-1627 (Customer service)
* Verified 8/15/88 *

Public domain software for
CP/M, DOS and others.
Catalog available on disk.

M & T Books
501 Galveston Dr.
Redwood City, CA 94063
(800) 533-4372
(800) 356-2002, in CA.
* Verified 6/15/88 *

Small-C Handbook and
Compiler. Small-Mac
Assembler. Small-Tools for
text processing.
Z80 Toolbook with disk.
Free catalog available.

Micro Cornucopia
P.O. Box 223
Bend, OR 97709
(503) 382-8048
(503) 382-7643 (BBS - 300/1200/2400 bps)
* Verified 8/15/88 *

Public domain software for
CP/M and DOS.
Free catalog available.

NAOG/Z-SIG
North American One-Eighty Group
P.O. Box 2781
Warminster, PA 18974
(215) 443-9031
* Verified 8/15/88 *

Z-System public domain software. No catalog available. (info available in back issues of group's newsletter.)

NorthStar Computer Society
P.O. Box 311
Seattle, WA 98111
(206) 525-9487
(206) 523-5355 (BBS-300/1200/2400)
* Verified 8/15/88 *

Public domain software for NorthStar computers. Free printed catalog available.

Poseidon Electronics
Ralph S. Lees, Jr.
103 Waverly Place
New York, NY 10011
(212) 777-9515
* Verified 4/15/88 *

Public domain software. Catalog lists compatibility with C-128, C-64 CP/M cartridge, or not at all. Catalog ($5)

Public Domain Software Copying Company
33 Gold St., Apt. L3
New York, NY 10038
(800) 221-7372
(212) 732-4942
* Verified 8/31/88 *

Public domain software for CP/M and DOS. Printed catalog available ($4). Some commercial CP/M software.

SIG/M
c/o ACG of NJ, Inc.
P.O. Box 135
Scotch Plains, NJ 07076
* Verified 6/15/88 *

Public domain software for CP/M. Catalog available on disk ($6). Make checks payable to the SIG/M Disk Librarian.

TUG Products
c/o Turbo User Group
P.O. Box 1510
Poulsbo, WA 98370
(206) 697-1151 (BBS-300/1200)
* Verified 8/15/88 *

Public domain TURBO Pascal programs in source code form, for CP/M and DOS.

The following Z-Nodes sell products for use with CP/M computers.

The Cedar Mill Z-Node
12275 NW Cornell Rd, Suite 5
Portland, OR 97229-5611
(503) 641-6101 (voice)
(503) 644-4621 (BBS - 300/1200/2400 bps; PC Pursuit access)
* Verified 8/15/88 *

CP/M software and hardware Catalog available for down loading-PRICES.DZT (short file) or PRICES.TZT (long).

Lillipute Z-Node
1709 N. North Park Ave.
Chicago, IL 60614
(312) 649-1730 (BBS - 300/1200/2400 bps; System I)
(312) 664-1730 (BBS - 300/1200/2400 bps; System II)
* Verified 8/15/88 *

Z-System, NightOwl, SLR SLR Systems, and others. Download catalog.

Newton Centre Z-Node

See Sage MicroSystems East

♦ ♦ ♦

Graphics Of A Better Sort

By Gary Entsminger
1912 Haussler Dr.
Davis, CA 95616

Gary's discovered presentation graphics, real presentation graphics. They're not a bad sort of thing to discover if you've got presentations to make.

All too often we need to convince someone (a client, a boss, a research institution) that we have significant information about something. In business for example, we might collect data (numbers, that is) which correspond to some monetary function, such as sales of Dr. Wooly windshield wipers.

A researcher in pathology might want to show a correlation between unsanitary living conditions and the proliferation of the Bubonic plague. (Fortunately, this one we already know about.)

A programmer might want to compare the efficiencies of algorithms.

In short, the sample and show scenario pervades most of the "real world." And, an excellent way to present data is to show it, as a graph or a picture.

Although computers have been very good at calculating, until recently they haven't been very good at making graphs or pictures. It's no laughing matter to PC dealers that potential customers often choose Apple Macintoshes primarily for their "better graphics."

But PC graphics are improving and have neared parity (I believe) with the Macintosh in many applications. One reason for my enthusiasm stems from a terrific program I started using this week — SlideWrite Plus, from Advanced Graphics Software in Sunnyvale, California.

Performance Analysis

Let's suppose you and your colleagues are analyzing the potential performance of critical functions in a large program. Today you're comparing algorithms for sorting objects.

Many smart (and shudder, "smarter") folks have already tackled sorting, so you won't have to figure everything out yourselves. If you're lucky you can draw most of your conclusions from someone else's data, and spend your time forging ahead.

Specifically, your program needs to sort strings which represent names in a database. So you face this situation —

(1) You haven't selected a specific sort nor written it;

(2) You have the results of tests of sorts (gleaned from programming books and journals);

(3) You don't know exactly how many names you'll be sorting at a time, but you know the range;

(4) You don't know what kind of order the names will be in. You can probably manipulate the order, if you become convinced there's a need for it. That is, you can maintain the names either randomly, in order, or possibly somewhere between, but it means revising your thinking about some other parts of the program;

(5) The program must be portable, which doesn't rule out assembly language techniques, but encourages you to find a solution in a high-level language;

(6) You've designed a prototype, but you haven't gotten into the programming yet.

Unfortunately, you find that the data you've collected on sorts have a rather relative nature.

Some of the tests were written in C, some in Pascal, some in Prolog. Some use iteration; some recurse. The codes for Quicksorts, insertion sorts, bubble sorts, tree sorts, and variations of all sorts are confusing. At this point, you want to compare ideas first; then (assuming you get the approval of higher ups) you'll give yourself a chance to write your own blazing-fast sorting functions.

What to do? What to do? One benchmarker, for instance, doesn't tell you what kind of computer he used. No PC, XT, AT mention. No clock speed.

You think you see a trend, but you're not sure you could convince anybody. Maybe you could with a picture.

Maybe SlideWrite Plus could help.

SlideWrite Plus

SlideWrite Plus does presentation graphics. In other words, you can present data as: bar graphs; charts; scatter, line, and area plots; pies; etc.

Performance and versatility for your CP/M or MS-DOS computer

QP/M

QP/M by MICROCode Consulting
Fed up with the message **"BDOS error: R/O"**? With QP/M, you'll never lose another file because you changed a diskette. QP/M offers full CP/M 2.2 compatibility with outstanding performance and more commands WITHOUT eating up precious program space. Get such features as automatic disk relogging, simple drive/user selection using either a colon or semi-colon, 31 user areas, drive search path, multiple program command line, archive bit maintanence, and transparent time/date stamping; all in the same space as CP/M 2.2. Installs from a convenient customization menu, no software assembly required. Bootable disks available with CBIOS for Kaypro, Xerox (8" or 5¼", -1 or -2), & BBI.

QP/M Operating System, bootable - specify system.... $ 64.95
QP/M without CBIOS (installs on any Z80 system)..... $ 49.95

Networks

QP/M Network File System by MICROCode Consulting
QP/M Network File System is an efficient local area network allowing up to seven CP/M computers to share peripherals and data resources.

- Transparent operation at speeds up to 11,000 bytes/second in syncronous mode
- Speeds of up to 1,920 bytes/second in asyncronous mode
- Local/remote disk drive and printer support
- Remote peripheral support for modems and real-time clocks
- All stations need not be on the network even though connected
- Local drive access protection and control
- Simple menu oriented configuration utility
- Extended DOS calls are provided for addition of custom network utilities.

Works with interrupt driven Z80 systems such as Xerox 820, Kaypro (KayPLUS & Advent ROMs), Eagle, and other computers running QP/M, or CP/M 2.2

QP/M Network File System ..... $ 39.95

Hard Disks

Need more speed and storage on your system?
Improve the productivity of your Z80 computer with a hard disk.

HDS Host Board
This daughter board provides a convenient interface for connecting a Western Digital WD1002-05 hard disk controller to your computer.

- Plugs into the Z80 socket, no other wiring required
- 40 pin interface for a WD1002-05 (or HDO) controller board
- Switch selectable I/O port addressing
- Comes as bare board or assembled & tested
- Kaypro '84 host board also available

Winchester Connection by MICROCode Consulting
The most simple and comprehensive hard disk software package available for CP/M.

- Designed for use with the WD1002-05 controller board
- Works with one or two hard disks - 5 to 64 meg
- Menu installed, no software to assemble
- Complete hardware tests and error handling
- Automatic swap, for warm boots from hard drive
- Software drivers install above or below CP/M
- Allows custom partition sizes and mixed drive types
- Independent block and directory sizes on each partition
- Includes manual, format, test, park, and swap utilities

Winchester Connection Software only ..... $ 39.95
HDS Board with Winchester Connection Software..... $ 79.95
HDS Bare Board with software ..... $ 59.95
HDS Board, WD1002-05, and software $245.00
Call or write for other pricing options

WD1002-05 HARD DISK CONTROLLER BOARD by Western Digital

- Standard ST506 drive interface
- Same size as standard 5¼" drive
- 40 pin interface to host computer
- WD2797 floppy disk controller interface on board
- Can control up to three hard drives
- Direct replacement for **Kaypro 10** controller

WD1002-05 Controller Board $185.00
Other Western Digital boards available

Prices subject to change without notice. VISA and Mastercard accepted. Include $5.00 shipping and handling, $7.50 for COD, UPS-Blue or RED Label additional according to weight. Please include your phone number with all correspondence.

Kaypro

KayPLUS ROM Set by MICROCode Consulting
Want more performance and flexibility from your Kaypro? With the KayPLUS ROM set you can have the advantages of a Kaypro 4 or 10, even on your Kaypro 2.

- Install up to four floppies and two hard drives
- Boots from floppy or hard disk
- Supports 96 TPI and 3½" disk drives
- Can use any ST506 type hard drive - 5 to 64 Meg
- 32 character type-ahead keyboard buffer
- Automatic screen blanking (not avail. on 83 series)
- 12 disk formats built-in, unlimited configurable
- Full automatic disk relogging with QP/M
- Internal real-time clock support
- No software assembly required

Includes manual, format, configuration, diagnostics, sysgen, diskette customization utility, AND hard disk utilities. Available for '83 and '84 series Kaypros.

KayPLUS ROM Set, specify model ..... $ 69.95
KayPLUS ROM Set with QP/M $125.00

Parts and accessories for the Kaypro

Kaypro 2X Real-time Clock parts kit ..... $ 29.00
Kaypro 2X Hard disk interface parts kit ..... $ 16.00
Kaypro 10 or '84 series Hard Disk host board ..... $ 49.00
Kaypro four drive floppy decoder board ..... $ 35.00
Complete parts and repair services available

Xerox 820

PLUS2 ROM and X120 Double Density Board by MICROCode Consulting and Emerald Microware
About had it with single density diskettes on your Xerox 820-1? Get unsurpassed versatility with our X120 Board and PLUS2 ROM package.

- Run up to four floppy disk drives at once
- Mix 8" and 5¼" at the same time
- Software compatible with Kaypro and Xerox 820
- Built in drivers for most serial and parallel printers
- Get mini-monitor functions and auto-boot capability
- 19 built in disk formats, including Xerox and Kaypro
- Includes custom disk format definition program
- Banked ROM BIOS for more space in your TPA
- Composite video adaptor on X120 board
- Runs 48 TPI diskettes on 96 TPI drives
- Supports real time clock from Z80-CTC
- Works on the Xerox 820-1 and Big Board I
- Both ROM and X120 board are required for operation

PLUS2 ROM Set and X120 Board A&T $114.95
PLUS2 ROM Set and X120 Bare Board ..... $ 49.95
PLUS2 ROM Set only ..... $ 39.95
120 Bare Board only ..... $ 15.00
★★★ **Special** ★★★ 2 boards for $25, 5 for $50
Other kits, parts, and packages available

Parts and accessories for the Xerox 820

Xerox 820-2 CPU Board - new ..... $ 75.00
Xerox 820-2 Floppy Controller board - new ..... $ 65.00
Xerox 820-2 CPU board w/ Floppy Controller $125.00
Xerox 820-1 CPU board - new ..... $ 75.00
Xerox 820 complete high profile keyboard ..... $ 65.00
Xerox 820 bare high profile keyboard - new ..... $ 25.00
Xerox 820 5¼" drive cable ..... $ 9.00
Xerox internal video cable w/brightness control .... $ 9.00
Xerox 820 power supply ..... $ 35.00
Power connector, specify board or cable ..... $ 2.50
Xerox parallel printer interface cable ..... $ 35.00
Dual Half Height 5¼" Disk Drives - DSDD,
in cabinet with standard Xerox cable $265.00
Complete parts and repair services available

EMERALD MICROWARE

P.O. Box 1726, Beaverton, OR 97075

VISA **(503) 641-0347** MasterCard

30 day money back guarantee on all products.

IBM PC

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.

UniForm-PC by MicroSolutions
How often have you wished you could use your CP/M diskettes on your PC? Now you can access your CP/M disks and programs on your MS-DOS computer just as you would a standard MS-DOS diskette. Once the UniForm driver is installed, you can 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, dual format 3½" (720k/1.44 meg.- PS/2), and even 8" drives.

UniForm-PC by MicroSolutions ..... $ 64.95
Uniform for Kaypro and other machines ..... $ 64.95

CompatiCard by MicroSolutions
Meet the CompatiCard, THE universal disk drive controller card. This half card will let you run up to 16 disk drives (4 per CompatiCard) on your PC or XT, including standard 360K, 96 TPI, high density (1.2 meg, dual speed), 8" single or double sided (SD or DD), and dual format 3½" drives (720k/1.44 - PS/2). The combinations are almost unlimited. Comes with its own MS-DOS driver and format program for high density and 3½" diskettes. Use it with UniForm-PC for maximum versitility. 8" adaptor and additional cabling available.

CompatiCard Board $169.95
CompatiCard with UniFORM-PC.. ★★ **Special** ★★.. $225.00
CompatiCard with UniFORM-PC & high density or
3½" drive ..... ★★★ **Special** ★★★ $350.00

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 $169.95

MatchMaker by MicroSolutions
Now you can copy your Macintosh diskettes right on your PC/XT/AT with the MatchMaker. Just plug your external Macintosh drive into the MatchMaker board and experience EASY access to your 3½" 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

Frustrated because your PC can't speak CP/M?

UniDOS by Micro Solutions
Run CP/M programs on your PC? Of course. UniDOS is a memory resident program that can use the NEC V20 CPU chip to actually RUN your favorite 8080 programs. Use UniDOS with UniForm-PC, and automatically switch to CP/M mode as you log on your CP/M diskette. Switch to emulation mode to run Z80 code programs or for systems without a V20. UniDOS directly converts video and keyboard emulation for Kaypro, Xerox 820, Morrow, Osborne, VT100, and eight other displays. All standard CP/M system calls are supported. Note: The NEC V20 CPU is a fast, low power, CMOS replacement for the 8088 CPU chip that includes a full 8080 instruction set as well as the standard 8088 set. Systems using an 8086 may substitute a V30 chip.

UniDOS by MicroSolutions ..... $ 64.95
UniDOS w/UniForm and V20-8 chip $135.00

UniDOS Z80 Coprocessor Board by MicroSolutions
This **8 Mhz. Z80H** half-card will run your Z80 and 8080 code programs at LIGHTNING speed on your PC or AT. Functions just like the UniDOS program, except NO V20 or emulation mode is required to run your programs. Now includes UniForm-PC!

UniDOS Z80 Coprocessor Card $169.95

You either enter or import data into columns, and SlideWrite shows your picture of choice. You can edit data, zoom in and out of a picture, draw on it (freehand), manipulate it, change fonts, add text and legends, and in many other ways fine tune your creation (vary tics, scale, orientations, rotations, and other peculiarities).

When you want hard copy, you can print, plot, or upload to a camera and shoot a roll. If you've got the right equipment (polorchrome film, a mounter), you can make overheads and slides right at your desk, or go into the overhead and slide business. In a worst case scenario, in most cities you could shoot a roll of Ektachrome and have slides the next morning.

Figure 1 shows a chart reproduced from a slide created with Image Maker (a camera) and SlideWrite Plus. Don't let the black and white reproduction disappoint you; the slide, itself, is in near-living color.

Alternately, you can export your charts to desktop publishing software (like Ventura) or to slide service bureaus. In other words, you can make a pretty impression and maybe convince a few folks.

Plots Of Sorts

Take those sorts, for example. I tested SlideWrite with the results from several sorting algorithms from two useful programming books — *The C Toolbox* by William Hunt and *Prolog Programming In Depth* by Michael Covington, et al.

Covington, et al, tested five sorts —

Insertion
Hoare's famous Quicksort (from 1962)
Difference-list Quicksort
Improved Quicksort
Treesort

Hunt tested —

Insertion
Hoare's famous Quicksort

Here's the pseudo code for the Insertion sorts —

```
repeat (for each element after the
        first)
 (1) compare to sorted elements
     to determine where it should be
     inserted;
 (2) move sorted elements to make
     room for it;
 (3) insert it.
end.
```

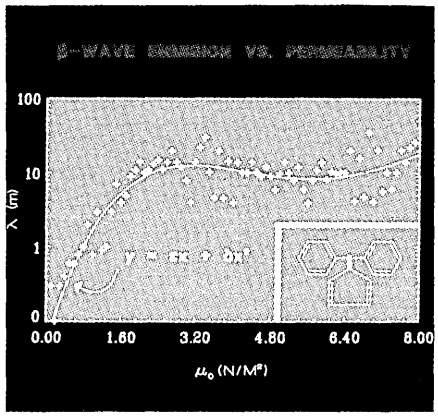


Figure 1 — Image Maker Slide

and the pseudo code for Quicksort —

```
(1) select a test value;
(2) partition the array or list
    using the test value;
(3) sort the left subarray (or list);
(4) sort the right subarray (or
    list).
```

To complicate matters, I entered Hunt and Covington's data into a Paradox database (of columns), then exported it as an ASCII file. SlideWrite Plus read the ASCII file correctly and created a chart in PCX format. Finally, I zipped the PCX file via modem to Micro C where Ventura imported it directly as Figure 2. Phew.

The y-axis represents time and the x-axis the number of elements in the database or array. The greater the slope of the regression line (a power curve in each of these cases), the more time it takes the function to sort the data.

A glance at Figure 2 shows an already ordered insertion sort as by far the most efficient (the curve is far less steep). A randomly ordered insertion sort is the most inefficient with an almost vertical curve. Random or ordered quicksorts are similarly efficient (both lie between).

In this example, I tried to fit several kinds of curves (linear, polynomial, exponential, etc.) to the data before I found the best fit — a power curve. SlideWrite's statistical function (F5) informed me that the power curve was a very good fit (accounting for 99% of the variation).

You can refit and redraw very quickly until you're satisfied with a description (or picture) of the data. Then you can embellish, modify, and print the charts you like.

Details & Features

SlideWrite Plus was written in Lattice C (and utilizes a few optimized assembly language functions). It's been an ongoing project for programmer Larry Daniel (now V.P. of Research & Development) for about five years.

Larry told me that when he began the SlideWrite project, his choice of a programming language was more or less obvious. Neither Microsoft nor Borland were C contenders then.

The excellent documentation (440 ring-bound pages) and numerous features are indicative of the work that's gone into this program.

Operation —
- all functions available through menu, and most through Fkeys also

Drawing —
- object-oriented (includes a picture and symbol library)
- rotates objects
- zooms images (in and out)

Graph —
- creates bars, lines, pie, text, scatter, area, 3-D, mixed, multiple graphs on one screen, free positioning of graphs, labels, and titles on screen

Colors —
- 16

Fonts —
- 16 typefaces, scalable to any size. Supports laser-printer cartridge & downloadable fonts

Output —
- postscript
- HP laserjet & compatibles
- various dot matrix printer
- film recorder
- plotters

Other —
- Greek/math characters
- curve fitting
- equation plotting
- error bars
- user-contolled line thickness
- automatically senses & uses a mouse

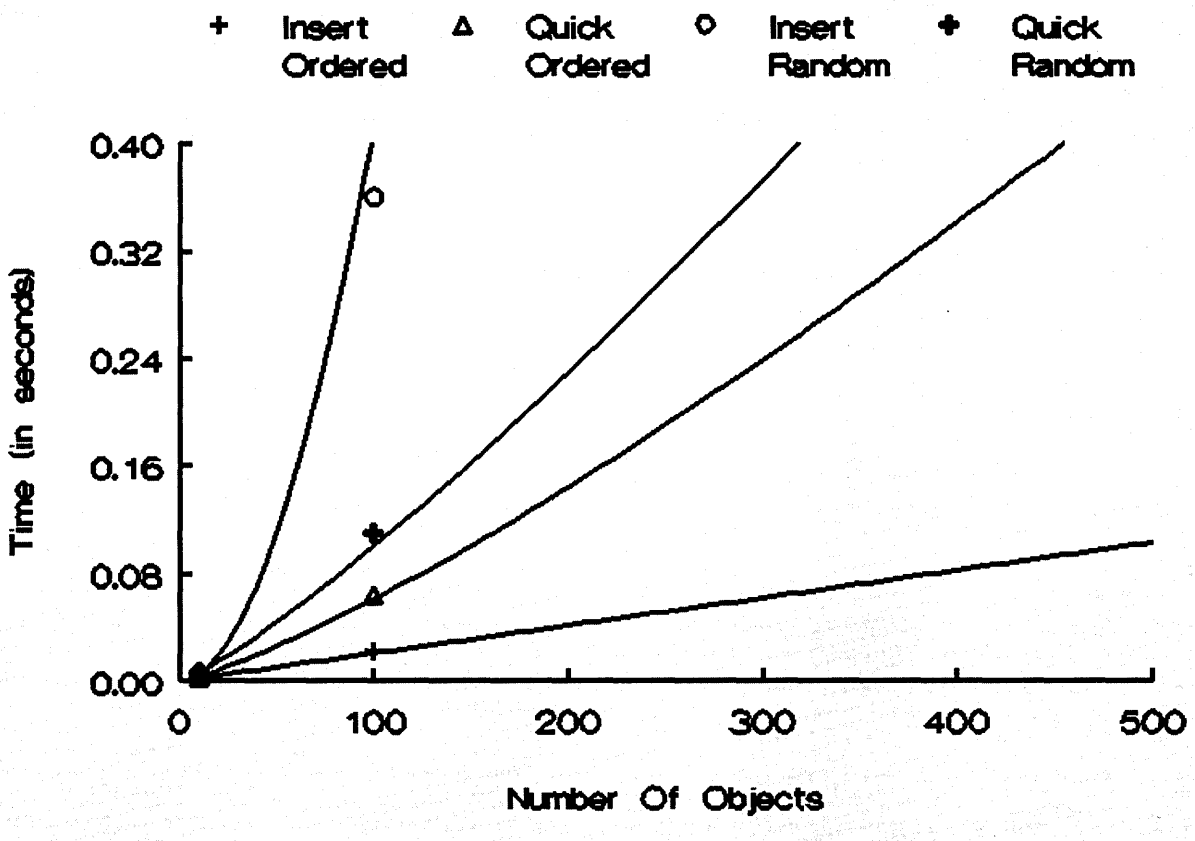
Limits —
- 4000 data points
- 12 graphs in one drawing

Requires —
- 390K available memory

Data formats —
- TIFF
- PCX
- CGM
- ASCII
- WKS
- WK1
- WRK
- WR1

Price — $445.

Figure 2 — Graphics Of Sorts Generated By SlideWrite



It sounds like I'm impressed with this program, and that's about right.

If you want to try out SlideWrite Plus with no commitment, call Advanced Graphics and they'll send you a little (and very useful) manual and a SlideWrite Plus demo disk which seems to do everything the real McCoy does.

The demo is free.

For more information —

Advanced Graphics Software, Inc.
333 W. Maude Ave., Suite 105
Sunnyvale, CA 94086-4367
(408) 749-8620

References

Covington, Michael, et al. *Prolog Programming in Depth.* Scott, Foresman, & Co. 1988.

Hunt, William. *The C Toolbox.* Addison Wesley. 1985.

A Good Place To Shop

I can't resist a good pitch —

"Isn't it the truth! Hardware engineers always blame the problems on software while software engineers always blame the problems on hardware. And marketing just wants to ship product.

"Well when I told my sister (she's a programmer) about these mugs, she said, 'That's exactly what they say!' So I know these will be perfect. Buy one for yourself or give one to your favorite opponent."

Picture two hardware and two software mugs, their handles shaped like ears, alongside mini-reviews of "The Fractal Geometry Of Nature" by Mandelbrot, and "Chaos" by the Royal Society of London.

In the same pages (28 in all), I found Escher puzzles and calendars, discussions of fuzzy math books, a consulting column, rubber stamps that reject everything, a code listing from a Build Your Own Assembler Project, OS/2, 80386, MS/DOS, and Unix programming guides.

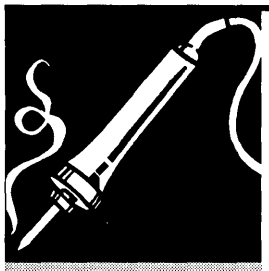
There are "far out" opinions and things to buy.

If you want bizarre and interesting access, subscribe to *John's Picks*, a "review catalogue" of books and things. Even the index is good reading! $5 per year from —

Microcomputer Applications
P.O. Box E
Suisun City, CA 94585-1050
(707) 422-1465

And that's Tidbits.

♦ ♦ ♦



TECHTIPS

Diagnosing A Slipped Disk

Disk Change Solution

Issue #45 arrived yesterday, and as usual I spent most of the evening with it. I found that the letter from John Innes of New South Wales, Australia, described some symptoms that sounded very familiar.

I'm using an AT clone with the AMI 286 BIOS. When I installed a 3.5 inch 720K drive for drive B and ran the BIOS SETUP routine, everything seemed fine. The format command worked, files could be read and written, etc. However, upon changing disks, the directory did not change. Of course, attempts to read from the disk produced garbage. I wasn't brave enough to try writing to the disk, but I would expect a write to destroy its contents.

Since I use MS-DOS 3.2 (which supports DRIVPARM in CONFIG.SYS), the solution was easy once I figured it out. It appears that the AMI SETUP utility sets up the disk parameter table to look for the door open signal. This causes a problem if the disk doesn't have a door open switch or if the switch doesn't work.

DRIVPARM is described in the DOS manual as having an option (/C) that "specifies that change-line (doorlock) support is required." What it doesn't say is that if the SETUP parameters specify change-line service, but the drive doesn't support it, then it should be disabled by using DRIVPARM without the /C option. I use:

```
DRIVPARM=/D:1
```

since the other default parameters are correct for a 720K drive. Mr. Innes should use —

```
DEVICE=DRIVER.SYS /D:0 /F:1
```

in his CONFIG.SYS file (unless his drive has a broken door-open switch, which should be fixed).

John R. Smith
609 E. Sacramento St.
Altadena, CA 91001

Editor's note: John presents a simple solution to the "phantom directory" problem. But those who just can't resist the opportunity for a little hardware hacking should read on.

... And A Hardware Version

John Innes' directory woes come from MS-DOS' method of handling disk swaps. The operating system actually reads a directory only once for a disk, storing this information in memory for subsequent DIRs. Only when a disk change is detected (through the disk-change or door-open signal from the drive) will DOS reset the drive.

Until recently, most drive controllers ignored the wishes of DOS and did a reset with each drive access. But newer controllers and drives often use signals from the drive to detect a disk change. So it's entirely possible to come up with a combination which can't detect a new disk. (^C will always reset the drives, but who wants to type endless ^Cs.)

You can easily fool the controller into resetting with each access. If it has seen any activity on the disk-change line (#2 on the ribbon cable), the controller assumes a disk change and resets accordingly. So just connect line #2 to the step line (#20) and the controller will see plenty of "disk changes." Do this by cutting line #2 and connecting the controller end to line #20.

The mod couldn't be much simpler. But if you're feeling lazy, Fujitsu and other companies sell a small adapter.

Editor's note: The preceding was gleaned from John Heilborn's "Ask Dr. John" column in Computer Currents, *December, 1988, Boston Edition. The article came to us from:*

Bruce S. Campbell
56 Birchwood Rd.
Windsor, CT 06095

Yet Another Hardware Fix

My 720K and 1.44M drives have XT and AT jumper options. I've found that jumpering the XT pins (even when the drive resides in an AT) solves the directory problem.

I suspect that XT disconnects pin 34. According to my documents, disconnecting pin 34 between the floppy and the controller tells the system to ignore pin 2 (the door-open line). So the system would have to reread the floppy each time it needs the directory.

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.

Unfortunately, the disconnection might also force an AT to treat the drive as a low density unit. Thus, a 1.2 meg drive would only read and write 360K. A 1.44 meg drive would only read and write 720K. (Check this on your system.)

Rob Aprato
323 75th St. #17
Everett WA 98203

MASM Tips

I just read the article on Microsoft's Macro Assembler, version 5.1. ("86 World," *Micro C*, issue #45) I completely agree with the author's positive comments. The new features make it a real pleasure to use. However, I'd like to mention some "gotchas." The first problem has to do with the new LOCAL directive, which allocates stack space and generates macros for reference of local (automatic) variables. Do not try to use a DWORD as the first local variable.

That's because the assembler always uses [BP-2] for the first local variable. This is fine for a WORD, but if you store a DWORD in [BP-2], it gets stored in [BP-2] through [BP+1], thus clobbering the saved BP at [BP]. If I need a DWORD, I declare the first local variable with the name DUMMY and the type WORD. The DWORD declaration then follows the dummy variable.

The second problem is not a bug, but a technique. In C, it's quite easy to pass the address of a local variable to another function, so that the second function can modify the local variable.

For instance, suppose you have a routine that displays a menu. One of the first things it does is save the contents of the screen memory:

```
show_menu()
{
 /* local variable */
 char screen_buffer[4000];

 /* pass address of array */
 save_screen(screen_buffer);
 ...
}
```

I ran into problems when I translated that type of C function into assembly. Unfortunately, automatic variables are allocated in the stack segment (referenced by SS), while static variables are allocated in the data segment (referenced by DS).

If you program in the large model, there's no problem because both segment and offset of the variable get passed. However, the small model only passes the offset and the function has no way of knowing whether the offset refers to the data segment or stack.

The only way around the problem is to make sure that SS equals DS. In high level languages, this is usually set up by the compiler. In an assembly language program, you do it yourself:

```
_main:
    mov   ax, @data
    mov   ds, ax
    cli
    mov   ss, ax
    mov   sp, OFFSET STACK
    sti
    ...
```

The CLI/STI instructions should not be necessary, but a few early (buggy) versions of the 8088 failed to disable interrupts when modifying SS. Note that SS must be changed, then SP.

Richard Lamb
3016 Waverly Dr., Apt. #311
Los Angeles, CA 90039

♦ ♦ ♦

PC Software as low as $2.49

Thousands of IBM/Compatible Public Domain and Shareware Programs
Are Available from the Micro Star Library and at Incredibly Low Prices!
We feature the best and most up-to-date shareware available.
Our software is guaranteed against bugs, defects, viruses, etc.
And We Offer FREE Technical Support For Our Customers

ORDER TODAY on our TOLL FREE PHONE LINES • ORDERS SHIPPED OUT SAME OR NEXT DAY

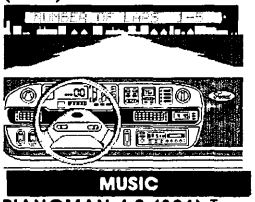
Incredibly Low Prices

1-9 Disks	$2.99 ea.
10-19 Disks	$2.69 ea.
20 or more	$2.49 ea.

GAMES

ARCADE GAMES (106) Has Kong, 3-D Pacman, Bricks, Pango. (Requires color.)
BASIC GAMES (107) Pac-man, Lunar Lander, Startrek, Meteor, Breakout, and others.
CARD GAMES (109) Canasta, hearts, draw poker & bridge.
STRIKER (110) Defender-like game. "Top Gun" in space.
FLIGHTMARE (112) Futuristic fighter pilot game. (Requires color graphics adapter.)
SLEUTH (117) Who done it?
DND (119) Like Dungeon and Dragons.
ROUND 42 (120) Better than Space Invaders. 42 levels.
GAMES IN BASIC (124) Lander, biorhythms, desert, Phoenix, Star Wars, others.
QUEST (152) Role playing adventure fantasy game. (Requires CGA.)
SPACE WAR (158) Dogfight in outer space, using phasers, photon torpedoes, etc.
BRIDGE PAL (171) Complete game of contract bridge, with tutorial.
FENIX (193) Just like the famous arcade game.
PINBALL GAMES (197) Pinball, Rain, Twilight Zone, Wizard, etc.
KID-GAMES (GAM8) Animals math, clock game, alphabet, etc.
CHESS (GAM9) Incredible. 2D and 3D. Many levels. Play back moves, store games.

EGA RISK (GAM11) World domination in great color. Includes EGA Asteroids.
PC PRO-GOLF (GAM27-28) Great graphics. Complete 18 hole, 72 par course. (CGA)
PEARL HARBOR (GAM32) Shoot down Jap Zeros before they destroy U.S. Fleet. (CGA)
ULTIMA 21 DELUXE (GAM34) Best Blackjack game around. Includes Video Poker.
FORD SIMULATOR (GAM37) Great driving simulation. (CGA)



MUSIC

PIANOMAN 4.0 (301) Turn your keyboard into a piano.
PC-MUSICIAN (302) Compose, save, and play music.

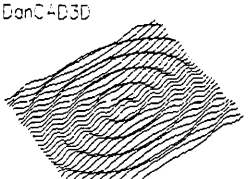
WORD PROCESSING

PC-WRITE 3.0 (434, 435, 436) (3 disks) Newest version! Very popular and complete. Includes spelling checker.
PC-TYPE+ (421-423) (3 disks) Excellent. Includes mail merge, 100,000 word spelling checker. Interfaces with PC-File+, PC-Style.

GRAPHICS

KEYDRAW CAD SYSTEM (1001, 1002, 1065) (3 disks) Popular. Also uses mouse. (Requires color graphics - CGA.)

SIDEWAYS (1007) Prints text sideways. Useful for spreadsheets.
SIMCGA/HGCIBM (1027, 1062) (2 disks) Use with Hercules graphics card/ compatibles to run programs requiring CGA on your monochrome PC.
IMAGE 3-D (1048) Create and edit 3-D objects. Move, scale, rotate and tip image.
FINGERPAINT (1050) Use keyboard or mouse to draw. Like MacPaint. (Requires CGA or EGA.)
DANCAD 3-D (1051, 1052) (2 disks) Create 3-D graphics. Rotate, magnify, etc. Runs on CGA, EGA, or Hercules.



FANTASY (1057) Create flowing graphic images with mouse or keyboard. (CGA).
FLOWCHARTING (1078-1079) Complete system for flowcharts, organizational, electrical, etc., with symbols.

RELIGION

THE BIBLE (3301-3306) (6 disks) Old Testament, King James version.
THE BIBLE (3307-3308) (2 disks) New Testament, King James version.
WORD WORKER (3309-3310) (2 disks) Bible search program. New Testament, King James version.
BIBLEMEN (3330) Excellent Bible quiz program.

BASIC

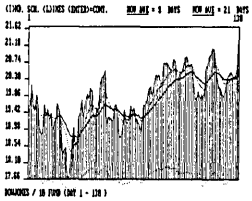
PC-PROFESSOR (1401) BASIC tutorial. Good.
BASIC PROGRAM GENERATOR (1402) The menu driven way to write programs.
B-WINDOW (1407) Give windowing capabilities to your Basic program.

UTILITIES

HOMEBASE (2608, 2612, 2613) Complete desktop organizer. Great!
PROFESSIONAL MASTERKEY (2805) Like Norton's. Retrieve deleted files. A lifesaver.
BAKER'S DOZEN (2821) 13 utilities from Buttonware.
AUTOMENU (3003) Make PC menu driven. Incl. passwords.
SCREEN (3006) Save your monitor from screen burn-in.
DOT MATRIX FONTS (3061-3062) (2 disks) Print your text in different fonts. Works with most printers.

ACCOUNTING/FINANCE

MARKET CGA (BUS17) Performs sophisticated analysis on stocks, funds, etc. (EGA version is BUS16).



BILLPOWER+ (BUS40) Bill clients for time and materials, advances, retainers, etc. Computes taxes, past due interest, etc. Has full G/L.
CPA LEDGER (706-708) (3 disks) Complete general ledger for corporations, partnerships or sole proprietors.
PERSONAL FINANCE MANAGER (715) Household budget manager. Keep track of checking, savings, investments.
PAYROLL USA (725-726) Up to 2,000 employees in any state. dBaseIII and Lotus compatible. Complete P/R system.
EXPRESS CHECK (786) Check acount with running balance, monthly reports, etc. Prints checks.
FINANCE MANAGER II (774-775) (2 disks) For personal or small business financial management.

DOS

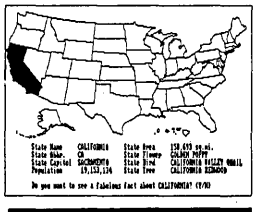
DOS TUTORIAL (1301) Teaches you to use DOS.
STILL RIVER SHELL (1304) Run DOS commands from a menu. Makes DOS easy.
BATCH FILE TUTORIAL (1305) Utilize batch file processing.
MORE DOS TIPS (1318, 1323) (2 disks) More about DOS.
HELP DOS (1326) On line DOS help with menus. Includes DOS dictionary of terms and a hints menu.

SPREADSHEETS

AS-EASY-AS (505) Great. Includes screen help menus. Utilizes function keys. A Lotus clone that reads Lotus files.
PC-CALC+ (512-514) (3 disks) Jim Button's famous Lotus clone.

EDUCATION

AMY'S FIRST PRIMER (248) Children's learning game that teaches letters, numbers and keyboard.
FUNNELS AND BUCKETS (201) A fun way to learn math.
MATHPAK (202) Tutorial with lessons in higher math.
PC-TOUCH (204) Learn typing.
BASIC TUTORIAL (208) Learn programming with BASIC.
BEGINNING SPANISH (211) Tutorial.
SPANISH II (232) Sequel.
BIBLEQ (214) Learn the Bible with this Q-A tutorial.
FACTS 50 (239) Geography lessons for U.S. Nice graphics.



APPLICATIONS

FORM LETTERS (1907) Commonly used form letters and business applications.
EZ-FORMS (1908) Make forms to meet different needs.
MANAGER'S PLANNER (1920) Daily planner. Prints out.
HOME INVENTORY (1966) Track all your possessions.
BIORHYTHM (1990) Display the 3 biological cycles: physical, emotional, intellectual.
FAMILY HISTORY (2203-2204) (2 disks) Create files and genealogical reports.
DR DATA LABEL (2327) Powerful mailing list program. Customize labels to size.
LOTTO PROPHET (2364) Best Lotto program we've seen.
CITY DESK (2513) Simple desktop publisher.

SPREADSHEET TEMPLATES

LOTUS MACROS (601) Save hours of work. (Req. Lotus)
LOTUS SPREADSHEET TEMPLATES (602) Ready-made. (Requires Lotus 1-2-3)
GOAL-SEEKER V3.5 (624) Achieve ojectives by changing spreadsheet and seeing result. (Requires Lotus.)
LOTUS TUTORIAL (630) Learn Lotus (requires Lotus).

ADULTS ONLY

ADULTS ONLY (2901) Animated. Req. CGA.
MAXINE (2902) Incredible. (CGA)
STRIP POKER (2903) Pick opponent (CGA)
BAD-BAD (2904) Adventure game.
ASTRO-(BLEEP) (2905) Arcade game (CGA)
X-RATED COLOR SHOW (2915) Beautiful girls. (CGA)
X-RATED PRINTSHOP (2909) Graphics for Printshop.

TELECOMMUNICATIONS

Q-MODEM 3.1 (1101, 1102, 1144) (3 disks) Powerful but easy to use. Fast.
RBBS V16.1A (1107-1109, 1150) (4 disks) Multi-user bulletin board system.
PROCOM 2.42 (1112-1113) (2 disks) Hacker's delight. Redial capability. Latest version.

SECURITY/HACKING

COPY PROTECTION I (1219) Instructions for unprotecting commercial software.
COPY PROTECTION II (1220) More software unprotect.
COPY PROTECTION III (1221) Additional software to unpro.
FLUSHOT (1225) Checks software for viruses.

DATABASE PROGRAMS

PC-FILE dB (801, 805, 837) (3 disks) Newest version! Rated better than dBase III+.
PC-GRAPH (802) Create graphics from PC FILE.
FILE EXPRESS 4.0 (803-804) Powerful system. Allows 32,000 records. Sorts up to 10 key fields.
DBASE III+ ROUTINES (851-852) (2 disks) Latest utilities to help you utilize dBase III+.

MICRO STAR

1105 SECOND ST. • ENCINITAS, CA 92024

HOURS: Monday - Saturday 7 AM - 5:00 PM, Pacific Time
TERMS: We accept MasterCard, VISA, Checks (allow 10 days to clear), Money Orders, and COD (add $4.00).
3½" DISKS: 3½" format add $1/disk.
SHIPPING & HANDLING: $3.50 (Total per order).
MAIL-IN ORDERS: Circle disk numbers. Include name & address.

CALL TODAY FOR FREE CATALOG

800-444-1343 Ext. 23

FOREIGN: 619-436-0130

Reader Service Number 120

Is There A Gap In Your Info?

Fill in your Back Issues of Micro C today!

ISSUE #1 (8/81)
Power Supply
RAM Protection
Video Wiggle
1/2 PFM.PRN
16 pages

ISSUE #2 (10/81)
Parallel Print Driver
Drive Motor Control
Shugart Jumpers
Program Storage Above PFM
1/2 PFM.PRN
16 pages

ISSUE #3 (12/81)
4 MHz Mods
Configuring Modem 7
Safer Formatter
Reverse Video Cursor
FORTHwords Begins
16 pages

ISSUE #4 (2/82)
Keyboard Translation
More 4 MHz Mods
Modems, Lync, and S10s
Undoing CP/M ERASE
Keyboard Encoder
20 pages

ISSUE #5 (4/82)
Word Processing
Two Great Spells
Two Text Editors
Double Density Review
Scribble, A Formatter
20 pages

ISSUE #6 (6/82)
BBI EPROM Programmer
Customize Your Chars
Double Density Update
Terminal In FORTH
24 pages

ISSUE #7 (8/82)
6 Reviews Of C
Adding 6K Of RAM
Viewing 50 Hz
On Your Own Begins
24 pages

ISSUE #8 (10/82)
SOLD OUT

ISSUE #9 (12/82)
BBII EPROM Program
Relocating Your CP/M
Serial Print Driver
Big Board I Fixes
Bringing Up WordStar
Cheap RAM Disk
32 pages

ISSUE #10 (2/83)
SOLD OUT

ISSUE #11 (4/83)
SOLD OUT

ISSUE #12 (6/83)
256K for BBI
Bringing Up BBII
dBase II
Look at WordStar
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
Nevada COBOL Review
BBI Printer Interface
Kaypro Reverse Video Mod
44 pages

ISSUE #14 (10/83)
BBII Installation
The Perfect Terminal
Interface To Electronic Typewriter
BBI Video Size
Video Jitter Fix
Slicer Column Begins
Kaypro Color Graphics Review
48 pages

ISSUE #15 (12/83)
Screen Dump Listing
Fixing Serial Ports
Playing Adventure
SBASIC Column Begins
Upgrading Kaypro II To 4
Upgrading Kaypro 4 To 8
48 pages

ISSUE #16 (2/84)
Xerox 820 Column Restarts
BBI Double Density
BBII 5"/8" Interface Fix
Kaypro ZCPR Patch
Adding Joystick To Color Graphics
Recovering Text From Memory
52 pages

ISSUE #17 (4/84)
Voice Synthesizer
820 RAM Disk
Kaypro Morse Code Interface
68000-Based System Review
Inside CP/M 86
56 pages

ISSUE #18 (6/84)
Kaypro EPROM Programmer
I/O Byte: A Primer
Kaypro Joystick
Serial To Parallel Interface
Business COBOL
60 pages

ISSUE #19 (8/84)
Adding Winchester To BBII
6 MHz On The BBI
Bulletin Boards
Track Buffering On Slicer
4 MHz For The 820-I
64 pages

ISSUE #20 (10/84)
HSC 68000 Co-Processor
DynaDisk For The BBII
Serial Printer On BBI Sans S10
Cheap & Dirty Talker For Kaypro
Extended 8" Single Density
72 pages

ISSUE #21 (12/84)
Analog To Digital Interface
Installing Turbo Pascal
Low Intensity BBI Video
Turbo Pascal, The Early Days
80 pages

ISSUE #22 (2/85)
Xerox 820-II To A Kaypro-8
Sound Generator For the STD Bus
Reviews Of 256K RAM Expansion
In the Public Domain Begins
88 pages

ISSUE #23 (4/85)
Automatic Disk Relogging
Interrupt Drive Serial Printer
Low Cost EPROM Eraser
Smart Video Controller
Review: MicroSphere RAM Disk
Future Tense Begins
86 pages

ISSUE #24 (6/85)
C'ing Into Turbo Pascal
8" Drives On The Kaypro
48 Lines On A BBI
68000 Versus 80x86
Soldering: The First Steps
88 pages

ISSUE #25 (8/85)
Why I Wrote A Debugger
The 32-Bit Super Chips
Programming The 32032
Modula II
RS-232C: The Interface
104 pages

ISSUE #26 (10/85)
Inside ZCPR3
Two Megabytes On DSI-32
SOG IV
The Future Of Computing
Graphics In Turbo Pascal
104 pages

ISSUE #27 (12/85)
SOLD OUT

ISSUE #28 (2/86)
Pascal Runoff Winners
Rescuing Lost Text From Memory
Introduction To Modula-2
Inside The PC
104 pages

ISSUE #29 (4/86)
Speeding Up Your XT
Importing Systems From Taiwan
Prototyping In C
C Interpreters Reviewed
Benchmarking The PCs
104 pages

ISSUE #30 (6/86)
PROLOG On The PC
Expert Systems
Logic Programming
Building Your Own Logic Analyzer
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)
SOLD OUT

ISSUE #34 (2/87)
SOLD OUT

ISSUE #35 (4/87)
SOLD OUT

ISSUE #36 (6/87)
Mouse Control
Build A Midi Interface For Your PC
Designing A Database, Part 2
Interrupts On The PC
Digital To Analog Conversion, A Designer's View
96 pages

ISSUE #37 (9/87)
Desktop Publishing On A PC
Build Your Own Hi-Res Graphics Scanner For $6, Part 1
Designing A Database, Part 3
Controlling AC Power From Your PC
Expanded Memory On The PC/XT/AT
Uninterruptable Power Supply For RAM Disks
96 pages

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

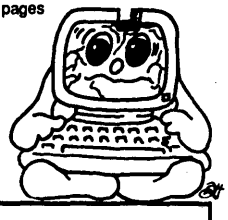
ISSUE #41 (5/88)
Artificial Intelligence
3-D Graphics
Neural Networks
Logic Of Programming Languages
Applying Information Theory
96 pages

ISSUE # 42 (6/88)
Maintaining PCs
Keeping Your Hard Drives Running
Troubleshooting PCs
XT Theory of Operation
Simulating A Bus
Ray Tracing
96 pages

ISSUE #43 (9/87)
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



To Order:

Phone: 1-800-888-8087
Mail: PO Box 223
Bend, Oregon 97709

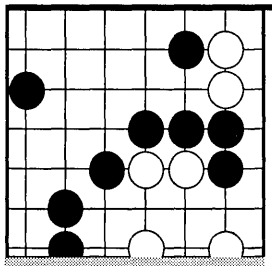
United States,	
Issues #1-34	$3.00 each ppd.
Issues #35-current	$3.95 each ppd.
Canada, & Mexico	
All issues	$5.00 each ppd.
Foreign (air mail)	
All Issues	$7.00 each ppd.

ADVERTISERS INDEX

Issue 46

Reader Service	Page Number
72	Acquired Intelligence 14
107	American Cosmotron 31
4	Austin Codeworks 45
5	Blaise Computing 13
**	Capital Software 20
15	Cascade Electronics 39
**	CC Software 35
**	C Gazette 59
105	Computerized Processing Unltd. 65
7	CompuView . . Inside Back Cover
90	Dair Computer Systems 79
10	Emerald Microware 87
93	Erac Company 15
112	Garrison, Peter 85
130	GEMS 67
138	Gibson Research 22
146	Greenleaf Software 7
11	Halted Specialties 23
22	Integrand 81
17	Manx Software Inside Front Cover
42	McTek Systems 75
**	Micro Cornucopia 94
37	Microprocessors Unltd 79
2	Microsphere 1
120	Micro Star 93
110	NuMega Technologies 2
145	Pathfinder Associates 74
119	Peripheral Tech 35
3	PC Tech . . . . . Back Cover
140	PMI 71
139	Quantum Software 18
129	Research Group 5
142	RJSwantek 70
127	SemWare 27
108	SofSolutions 6
40	Star-K 30
141	Sterling Castle 18
101	United Products 63

** Contact Advertiser Directly.



THE LAST PAGE

By Gary Entsminger
1912 Haussler Dr.
Davis, CA 95616

Computer Go —

Playing With Rules And Patterns

I remember watching the Go games in the Tektronix lunchroom. Many of the cafeteria "masters" had novices pick up their food so they could play an extra five minutes.

Often the discussion would turn to computerizing the game, but the masters always discouraged the idea. "Too complex," they'd say. "Do something easy, like Chess." So we'd go back to watching the boards and fetching food.

Every game has a set of rules which separates it from its surrounding reality. Most games have a simple set of rules and simple behavior or potential board configuration. But some games (the most intriguing ones) have a simple set of rules which often lead to extremely complex boards and patterns.

One of these, the Asian game "Go," has survived 4,000 years of human, and 20 years of computer attempts to master it.

In Go, two players alternate in placing stones on a 19x19 grid. Once placed, a stone cannot be moved, but it can be removed (via capture).

You win the game by enclosing more territory than your opponent. In the process of enclosing territory, you may capture an opponent's stones by surrounding them. Generally, the fewer stones either of you has on the board, the less territory you hold.

The rules of Go are much simpler than the rules of chess, but a Go board can be much more difficult to decipher because of the incredible number of possible board configurations (about 10^{761}).

In chess, all pieces are placed on an 8 x 8 board before the game starts. Each play moves a piece to a new location. The number of legal options can increase or decrease as the game progresses.

The complexity of the "look ahead," or "what if I do this, and you do that, and I do this..." scenario is complex in chess (configuration possibilities about 10^{120}), but far less complex than in Go.

In Go, there's only one "play" — you place a stone on an unoccupied intersection. At most (at the beginning of the game), there are 381 possible plays. Each move reduces the number of possible plays by one, while increasing the complexity of the board by a large nonlinear factor.

Although the rules of Go are simple, the staggering number of possible board configurations (or patterns) make programming a "smart" Go game very, very difficult.

Nevertheless, computer Go games are improving thanks to the joint efforts of AI researchers and "smart" programmers.

According to Walter Reitman and Bruce Wilcox, the designers and programmers of two "smart" Go games —

"Skilled human Go play presumes the ability to recognize and make inferences from many different kinds of complex patterns — our Go programs use a small set of basic scanning and recognition mechanisms (rules) to deal with these patterns."

Nemesis, Go Master

Recently, I had the pleasure of playing Bruce Wilcox's latest version of Go — Nemesis, the Go Master (from Toyogo, Inc., in Lexington, Massachusetts).

Nemesis is a clone of POGO, created by Reitman and Wilcox on an IBM mainframe between 1972 and 1977. That Go took seven person-years and 8,000 lines of LISP to create. It required 3 megabytes of memory to run.

Bruce writes (in *Reflections on Building Two Go Programs*) that a single game of mainframe Go costs close to $2,000. So eventually, even on an academic budget, he had to give it up. Fortunately, a game with Nemesis on the PC isn't so expensive.

It took Bruce one year and 13,500 lines of C to create Nemesis. Over 3,000 lines of that (almost 1/4) went into the user-interface. It takes only 146 kilobytes and a PC to play it. At $79, Nemesis is affordable (even for non-academics).

Nemesis is as good (or better) a player than its mainframe ancestor.

I've had a blast playing (and learning to play) this incredible game. The beauty of it is that Nemesis helps —

- Each time Nemesis places a stone, it explains its motives, associating them to a rule (or rules).
- I can ask Nemesis to help me place a stone and to explain its motives.
- I can edit the board (to a new position) and ask Nemesis to suggest moves.
- I can take back moves (and so can Nemesis).
- I can practice (or learn) by playing Nemesis on smaller boards (9 x 9 and 13 x 13).
- I can save and resume games later.

For more information —

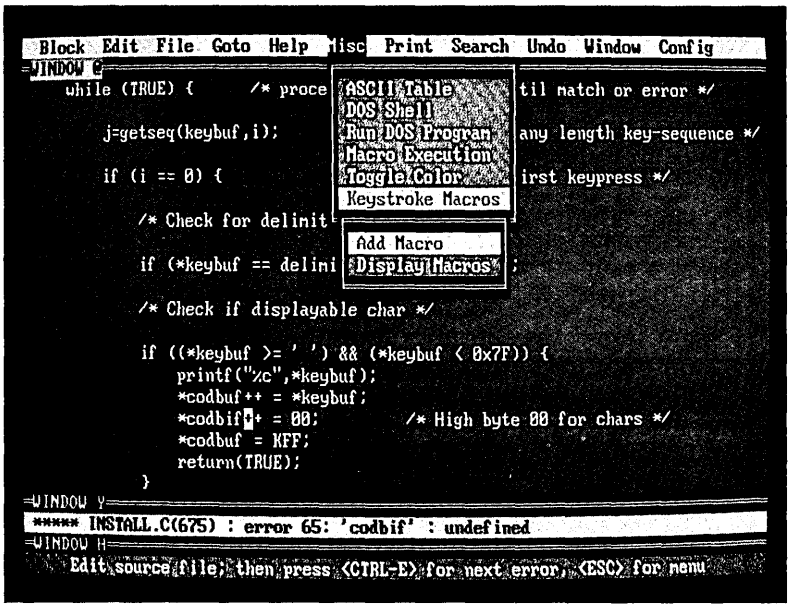
Toyogo, Inc.
76 Bedford St., Suite #34
Lexington, MA 02173
(617) 861-0488

References

Reitman, Walter and B. Wilcox. *Pattern Recognition and Pattern-Directed Inference In a Program for Playing Go.* Academic Press. 1978.

Wilcox, Bruce. *Reflections On Building Two Go Programs.* SIGART Newsletter. October 1985. Number 94.

♦ ♦ ♦



VEDIT PLUS

#1 PROGRAMMABLE EDITOR

NEW VERSION 3.0

- **Best Multi-Level Undo**
- **Regular Expressions**
- **Pop-Up ASCII Table**
- **Pull-Down Menus**
- **Compiler Support**
- **Column Blocks**

Until now, if you wanted the best Undo, the best compiler support, regular expressions and column blocks you chose BRIEF™. If you wanted unlimited keystroke macros, the best configurability, "off the cuff" command language macros and blazing speed, you chose VEDIT PLUS.®

Now the Choice is Easy

The all new VEDIT PLUS 3.0 gives you the best Undo of any editor, the best compiler support, unequaled windows, true regular expressions and extensive new features. We're leading the way with easy to use pull down menus, context sensitive help, a pop-up ASCII table, new printing options and much more. Incredibly, VEDIT PLUS 3.0 is now twice as fast as before and, at only 60K in size, it loads fast!

Completely Configurable

Change a few keys or redefine the entire keyboard, VEDIT PLUS adjusts to your editing style in minutes. You can even create new editing functions using simple keystroke macros or fine tune existing ones. VEDIT PLUS is so configurable that it easily emulates other editors and word processors (WordStar and Word Perfect emulation included). Quickly access editing functions with a single key or through the pull-down menus.

Try before You Buy

We challenge you to experience the dazzling performance and exceptional features that make VEDIT PLUS the best choice. Our evaluation disk includes the complete editor.* Learn VEDIT PLUS using our extensive "training" macro that gives instructions in one window while you experiment in another. See for yourself why no other macro language comes close.

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

Supports the IBM PC, XT, AT and PS/2 including DESQview, Microsoft Windows, PC-MOS/386, Concurrent DOS and most networks. Also available for MS-DOS, CP/M-86 and FlexOS. (Yes! We support windows on CRT terminals.) $185.

*Also available for TI Professional, Tandy 2000, DEC Rainbow, WYSE 700, Amdek 1280 and Others.
*Free evaluation disk is fully functional and can even edit small files.

FREE EVALUATION COPY* Call 1-800-45-VEDIT

- **Fully Network Compatible**
- **Call for XENIX and OS/2 versions**
- **30 Day Money-back guarantee**

Features of VEDIT PLUS 3.0

- Simultaneously edit up to 37 files of unlimited size.
- Variable sized windows; multiple windows per file.
- Execute DOS commands and other programs.
- Flexible "cut and paste" with 36 "scratch-pad" buffers.
- Block operations by line, character or column.
- Search with pattern matching or regular expressions.
- Configuration—determine your own keyboard layout, create your own editing functions, support any screen size.
- Select window colors, support 43 line EGA, 50 line VGA.

EASY TO USE

- Modern pull-down menu system. Pop-up ASCII table.
- Context sensitive on-line help is user changeable.
- Multi-level Undo (100 to 1000 levels). Undo keystroke by keystroke or line by line.
- On-line integer calculator (also algebraic expressions).
- Keystroke macros speed editing, menu function "hot keys."

FOR PROGRAMMERS

- Automatic Indent/Undent for "C," PL/I, PASCAL.
- Match/check nested parentheses, e.g. "{" and "}" for "C."
- Flexible macro runs popular compilers and automatically moves cursor to each error in your program. Easily changed to support new compilers and assemblers.

FOR WRITERS

- Word wrap, paragraph formatting and justification.
- Convert to/from Wordstar and mainframe files.
- Flexible printing; fully adjustable margins and Tab stops.

MACRO PROGRAMMING LANGUAGE

- If-then-else, looping, testing, string compare, branching, user prompts, keyboard input, 24 bit algebraic expressions.
- Flexible windowing—forms entry, select size, color, etc.
- Simplifies complex text processing, formatting, conversions.
- Extensive 400 page manual with hundreds of examples.

Reader Service Number 7

1955 Pauline Blvd., Ann Arbor, MI 48103
(313) 996-1299 • Telex 701821 • Fax (313) 996-1308

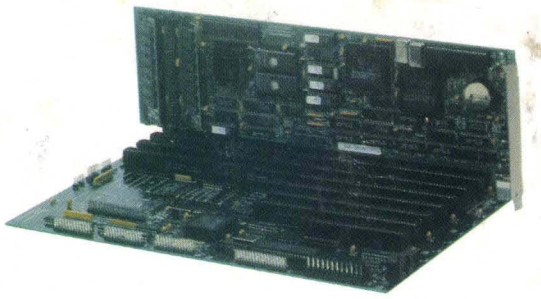
CompuView

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.



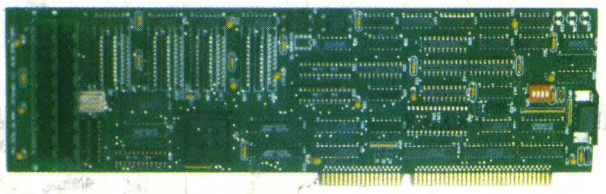
PC Tech X24 and ASMB

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.

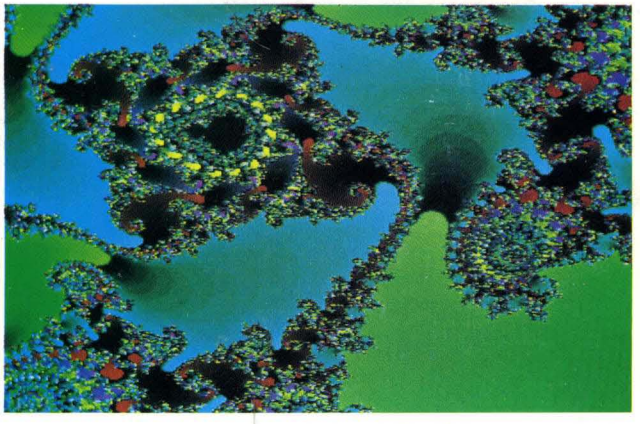
About PC Tech

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