GETTING THE MOST OUT OF THE 8086/8088

For the PC Systems Integrator

August 1988 ● $3.95 USA (Canada $4.95)

Micro Systems JOURNAL

UNIX ON THE PC: All Systems Go!

386 UNIX Alternatives

How Does UNIX Stack Up To O/S2?

Running DOS As A UNIX Task

08

0 38351 19902 9

Make your programs millions of times smarter.

More and more, programmers and workstation builders are using DESQview 2.0 as a development tool. The reason is simple. They can create powerful, multitasking solutions today for the millions of DOS PCs in use today. Solutions comparable to those promised for tomorrow by OS/2.

Some of the applications under development right now using DESQview 2.0 API Tools: CAD, Medical systems, insurance, 3270 mainframe communications, network management, real estate, typesetting, point of sale, education, commodity trading, stock trading and online voting.

The API Advantage

Programmers who take advantage of DESQview's API (Application Program Interface) get access to the powerful capabilities built into DESQview–multitasking, windowing, intertask comunications, mailboxes, shared programs, memory management, mousing, data transfer, menu-building and context sensitive help.

Bells and Whistles

A program taking advantage of the DESQview 2.0 API can spawn subtasks for performing background operations or new processes for loading and running other programs concurrently. It can schedule processing after an interval or at a certain time. It can use DESQview's intertask communications to rapidly exchange data between programs, share common code and data; or interrupt at critical events. It can use DESQview's menuing and mousing capabilities to create menus. And there's lots more it can do.

80386 Power

80386 programmers can take advantage of the 80386's protected mode for large programs, yet run on DOS and multitask in DESQview–side by side with other 80386 and DOS programs. The breakthroughs that make this possible: DOS Extenders from PharLap Software and AI Architects and DESQview support of these DOS extenders.

DESQview Developer Conference

So if you are a developer, looking to create programs with mainframe capabilities, but wanting to sell into the existing base of millions of DOS PCs, come to Quarterdeck's first DESQview API Developers Conference, August 16-18, 1988 at the Marina Beach Hotel, in Marina del Rey, California. For more information call or write us.

Come learn about the DESQview 2.0 API and 80386 DOS Extenders. Meet 80386 experts as well as those smart people who are creating DESQview 2.0 API workstations solutions.

And if you want to get a leg up before the conference, ask us about the DESQview API Tools for assembler or C programmers.

Bringing New Power to DOS. DESQview 2.0 API Toolkit.

Quarterdeck Office Systems 150 Pico Blvd.,Santa Monica, CA 90405
(213) 392 9851

CIRCLE 86 ON READER SERVICE CARD.

Get your work done before 1991.

The future of personal computing is clear. More powerful PCs. Easier to use PCs. With graphics and character-based programs working side by side. Talking to each other. Multitasking. Windowing. Menuing. Mousing. Getting your work done easier and faster.

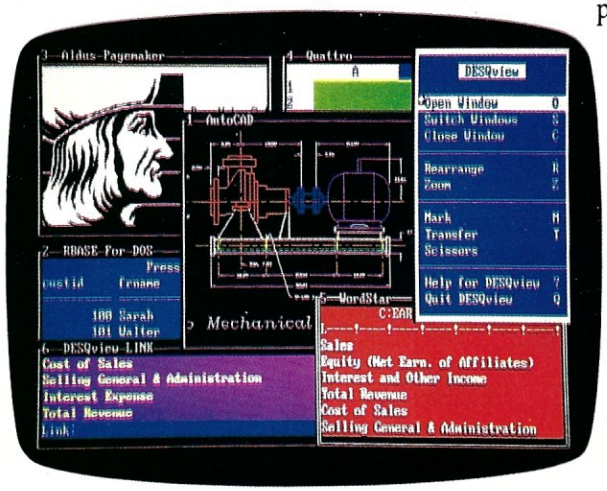
Have it all now.

DESQview™ is the operating environment that gives DOS the capabilities of OS/2.™ And it lets you, with your trusty 8088, 8086, 80286, or 80386 PC, leap to the productivity of the next generation. For not much money. And without throwing out your favorite software.

Add DESQview to your PC and it quickly finds your programs and lists them on menus. So you can just point to the program, using keyboard or mouse, to start it up. DESQview knows where that program lives. And what command loads it.

For those who have trouble remembering DOS commands, it adds menus to DOS. It even lets you sort your files and mark specific files to be copied, backed-up, or deleted—all without having to leave the program you're in.

Best of all, DESQview accomplishes all this with a substantial speed advantage over any alternative environment.



Multitask beyond 640K.

When you want to use several programs together, you don't have to leave your current program. Just open the next program. View your programs in windows or full screen. Open more programs than you have memory for. And multitask them. In 640K. Or if you own a special EMS 4.0 or EEMS memory board, or a 386 PC, DESQview lets you break through the DOS 640K barrier for multitasking. If you have other non-EMS memory expansion products like AST's Advantage or the IBM® Memory Expansion Option, we have a solution for you, too. The ALL CHARGECARD™ 'unifies' all your memory to provide up to 16 megabytes of continuous workspace. DESQview lets you use this memory to enhance your productivity. You can start 1-2-3 calculating and tell Paradox to print mailing labels while you're writing a report in Word Perfect, or laying out a newsletter in Ventura Publisher, or designing a building in AutoCAD.

DESQview even lets you transfer text, numbers, and fields of information between programs.

For programmers, DESQview's API, with its strengths in inter-task communications and multitasking, brings a quick and easy way to adapt to the future. With the API's mailboxes and shared programs, programmers are able to design programs running on DOS with capabilities like those of OS/2.

Fulfill the 386 promise.

For 80836 PC users, DESQview becomes a 386 control program when used in conjunction with Quarterdeck's Expanded Memory Manager (QEMM)-386—giving faster multitasking as well as virtual windowing support.

And when you use DESQview on an IBM PS/2™ Model 50 or 60 with QEMM-50/60 and the IBM Memory Expansion Option, DESQview gives you multitasking beyond 640K.

Experts are voting for DESQview. And over a million users, too.

If all of this sounds like promises you've been hearing for future systems, then you can understand why over a million users have chosen DESQview. And why *PC Magazine* gave DESQview its Editor's Choice Award for "The Best Alternative to OS/2," why readers of *InfoWorld* twice voted DESQview "Product of the Year" why, by popular vote at Comdex Fall for two years in a row, DESQview was voted "Best PC Environment" in *PC Tech Journal's* Systems Builder Contest.

PC MAGAZINE EDITOR'S CHOICE NOV. 24, 1987

INFO WORLD 1986 AND 1987 PRODUCT OF THE YEAR

DESQview lets you have it all now.

DESQVIEW SYSTEM REQUIREMENTS:
IBM Personal Computer and 100% compatibles (with 8086, 8088, 80286, or 80386 processors) with monochrome or color display; IBM Personal System/2• Memory: 640K recommended; for DESQview itself 0-145K • Expanded Memory (Optional): expanded memory boards compatible with the Intel AboveBoard; enhanced expanded memory boards compatible with the AST RAMpage; EMS 4.0 expanded memory boards•Disk: two diskette drives or one diskette drive and a hard disk•Graphics Card (Optional): Hercules, IBM Color/Graphics (CGA). IBM Enhanced Graphics (EGA), IBM Personal System/2 Advanced Graphics (VGA)• Mouse (Optional): Mouse Systems, Microsoft and compatibles• Modem for Auto-Dialer (Optional): Hayes or compatible• Operating System: PC-DOS 2.0-3.3; MS-DOS 2.0-3.2• Software: Most PC-DOS and MS-DOS application programs; programs specific to Microsoft Windows 1.03-2.03, GEM 1.1-3.0, IBM TopView 1.1• Media: DESQview 2.0 is available on either 5-1/4" or 3-1/2" floppy diskette.

YES! I need increased productivity now!

Qty	Product	Format	Price Each	Totals
	DESQview 2.0	❑5-1/4 ❑3-1/2	$129.95	
	QEMM-386	❑5-1/4 ❑3-1/2	$59.95	
	QEMM-50/60	❑5-1/4 ❑3-1/2	$59.95	
	ALL CHARGECARD (Special for DESQview owners)		$200.00*	
		Shipping & Handling	$5 in USA/ $10 outside USA	
		Calif Residents add 6.5%		
		Grand Total		

Name _____

Address _____

City _____ State _____ Zip _____

Payment Method ❑ Visa ❑ MasterCard Expiration ____/____

Account # ☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐

Quarterdeck

150 Pico Boulevard, Santa Monica, CA 90405
(213) 392-9851

*This ALL CHARGECARD is designed for the IBM PC AT and PS/2 50 and 60. If you have another type of 80286-based PC, there's a version for you, too. Please call 1-(800) 387-2744 for special ordering information. Offer expires August 31, 1988.
Trademarks are property of their respective holders: IBM, OS/2, PS/2, 1-2-3, Paradox, Word Perfect, Ventura Publisher, AutoCAD, Intel, Above Board, AST, RAMpage, Advantage, Hercules, Mouse Systems, Hayes, Microsoft, Windows, TopView.

CIRCLE 101 ON READER SERVICE CARD

We've spent years developing our file manager so you won't have to.

You could invest hundreds of programming hours writing a file management system for your next application.

Or you could simply invest in Btrieve.® And get all the file handling functionality you need, through subroutine calls from your favorite programming language.

Portable. Write your application once. Wherever Btrieve runs, your application will run, whether in a single user or multiuser environment. In fact, Btrieve is the standard access to NetWare.®

Fast. Written in assembly language, Btrieve uses b-tree indexing algorithms with caching and automatic balancing for fast, efficient file management.

Safe. Btrieve is the only fault tolerant file manager with built-in file recovery. In the event of a system or power failure, your database is protected.

Flexible. Develop applications with the capabilities you need most. Like 255 open files, unlimited records per file, 24 indexes per file, and a maximum file size of up to four gigabytes. You can access Btrieve from BASIC, C, Pascal, COBOL and others.

Invest in Btrieve. At just $245 for single user and $595 for multiuser, it's a small price to pay for all the file manager you'll ever need.* And you'll never pay royalties on the applications you develop. To find out more, see your authorized Novell reseller, or call (512) 346-8380.

For more information, call from your modem 1-800-444-4472 (8 bit, no parity, 1 stop bit) and enter the access code NVBT13.

NOVELL

For software solutions, you should be seeing red.

*Suggested retail price (US dollars) ©1987 Novell, Inc., World Headquarters, 122 East 1700 South, Provo, Utah 84601 (801) 379-5900
Requires PC-DOS or MS-DOS 2.X, 3.X or Xenix.

CIRCLE 65 ON READER SERVICE CARD

For the PC Systems Integrator

Micro Systems JOURNAL

August 1988
Vol. 4 . No. 8

FEATURE ARTICLES

PRODUCT REVIEWS

COLUMNS

About the cover: It is the dawn of a new age for the PC. The advent of 386 chip technology now makes it possible to harness the power of the mainframe or minicomputer on the desktop. In this issue we will explore ways to push your PC to the edge of the processing envelope with a look at UNIX alternatives for the 386.
Cover photograph by Michael Carr

DEPARTMENTS

Micro/Systems Journal (ISSN 8750-9482) is published monthly by M & T Publishing, Inc., 501 Galveston Drive, Redwood City, CA 94063; (415) 366-3600. Second-class postage paid at Redwood City and at additional entry points. **POSTMASTER:** send address changes (form 3547) to *Micro/Systems Journal*, Box 3713, Escondido, CA 92025. **Change of Address:** Please send old label and new address to *Micro/Systems Journal*, Box 3713, Escondido, CA 92025. **Customer Service:** For subscription orders and changes of address, call toll-free: (800) 321-3333; in California call (800) 331-4164; outside U.S. call (619) 485-9623. **Subscription Rates:** U.S., $29.97 for one year, $56.97 for two years. Foreign orders must be prepaid in U.S. funds with additional postage. Add $10 per year for surface mail to all countries or add $20 for airmail to Canada and Mexico; $28 for airlift/surface to other countries.

UNIX Challenges OS/2

Last year Intel shipped an estimated 600,000 80386 microprocessors. This year it is expected to ship close to 3 million 386 chips. Most of those chips are going into PC-compatibles, and a very significant percentage will not be running either DOS or OS/2. Rather, they will use UNIX as their operating system.

UNIX first became available on PCs way back in 1984, soon after the XT was introduced. IBM even introduced a version of UNIX for the XT that ran a three-user system. However, the XT's 10-Mbyte hard disk drive, slow speed, and small memory made UNIX more of a curiosity (some preferred to call it a joke) than a usable product.

When the AT was introduced, UNIX implementers thought that here, finally, was a desktop system that could support a small multiuser UNIX system. Several UNIX implementations were introduced. However, the improved horsepower of the AT was soon found to be too lacking for anything but the smallest multiuser systems.

The introduction of fast 80386 systems, megabyte-size memories, and hard disks capable of storing 100 megabytes and up appear to finally have provided the necessary performance and architecture to build low-cost desktop UNIX systems that compare favorably with mainframe UNIX implementations. Prices for multiuser UNIX systems based on this 386 hardware now compete favorably with local area networking systems. Three such UNIX systems are reviewed in this issue.

IBM and Microsoft, the two most powerful marketing entities in the computer business, are putting their combined forces behind OS/2. Their marketing wherewithal is most formidable. On the other hand, Sun Microsystems and AT&T have joined forces and are enlisting the aid of other companies, such as Unisys and Motorola, to develop a standard UNIX. This would overcome many of the incompatibility problems currently associated with UNIX.

OS/2 was written for 286-based machines and hence does not take advantage of the 386's architecture; yet OS/2 needs the 386's performance. On the other hand, many UNIX implementations exploit the 386's features. Add to that the ability to manage large memory, graphical user interfaces, and a powerful LAN interface, and UNIX presents a formidable competitor to OS/2. Also, there are many good UNIX applications already available while the availability of OS/2 applications has a long way to go to catch up.

It will be some time before we see a 386 version of OS/2. In the meantime, the UNIX vendors have a window of opportunity. And, they appear to be taking advantage of it.

Several UNIX vendors have also introduced a DOS facility for their UNIX implementations that allow many standard DOS applications to run in a true multiuser/multitasking environment. OS/2, on the other hand, can run standard DOS application software, but only in a single-tasking mode called the "compatibility box." OS/2 requires that applications which can be multitasked be written specifically for OS/2. On 386 systems, UNIX with DOS tasking facilities utilizes the 386's 8086 real mode, which means standard, unmodified DOS tasks can be run in a true multitasking environment. Many users are choosing this approach to multitasking standard DOS applications. Two such implementations are reviewed here.

We think you will find this issue of *Micro/Systems* provides a window into things to come, and one of the most interesting issues we have published. We found it rewarding to put it together for you and hope you will find it just as rewarding to read.

Sol Libes

Sol Libes
Founder and Editor

For the PC Systems Integrator
Micro\Systems JOURNAL

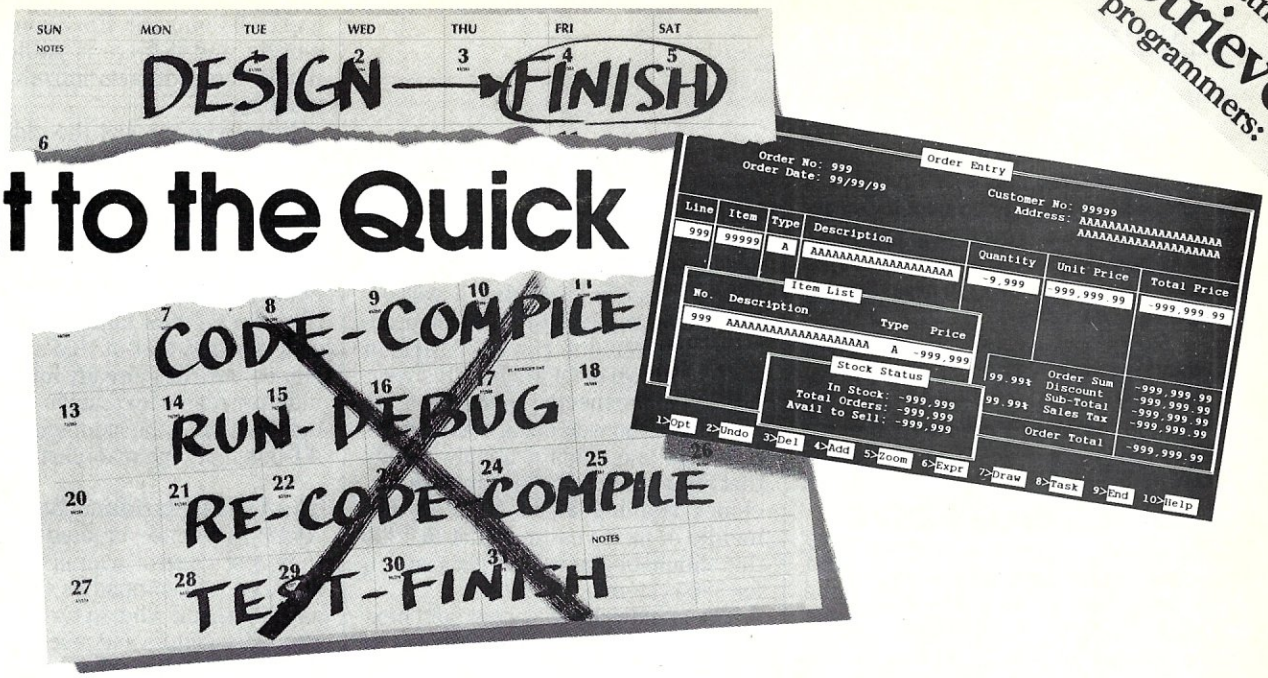
Micro/Systems Journal (ISSN 8750-9482) is published monthly by M&T Publishing, Inc., 501 Galveston Drive, Redwood City, CA 94063; (415) 366-3600. Second-class postage paid at Redwood City and at additional entry points.

Article Submission: If you have a specific area of expertise or interest and would like to contribute, please write Micro/Systems Journal, P.O. Box 1192, Mountainside, NJ 07092; (201) 522-9347, or contact M&T Publishing, Inc., 501 Galveston Drive, Redwood City, CA 94063; (415) 366-3600. Please do not submit articles without first contacting the editors.

Correspondence: Please send letters to the editor to Micro/Systems Journal, 501 Galveston Drive, Redwood City, CA 94063. Other editorial correspondence may also be directed to P.O. Box 1192, Mountainside, NJ 07092. The editors may also be reached via MCI Mail (SLIBES or MSJ).

Advertising Rates: Available upon request. Call (415) 366-3600 or write to: Advertising Department, Micro/Systems Journal, 501 Galveston Drive, Redwood City, CA 94063.

Attn Btrieve® programmers:



Cut to the Quick

MAGIC PC ELIMINATES CODING . . . CUTS MONTHS OF DATABASE DEVELOPMENT!

Time is money. And coding a DBMS application like Accounting or Order Entry takes a lot of both. Simply because hacking out mountains of code with your RDBMS or 4GL is too slow. Not to mention the time to rewrite if you make a mistake or change the design.

EXECUTION TABLES ELIMINATE CODE!

Magic PC cuts months of your application development time because it eliminates coding. You program with the state-of-the-art Execution Tables in place of conventional programming.

HOW DOES IT WORK?

Magic PC turns your database design scheme directly into executable applications without any coding. Use Execution Tables to describe only what your programs do with compact design spec's, free from lengthy how to programming details. Each table entry is a powerful non-procedural design instruction which is executed at compiled-like speed by a runtime engine. Yet the tables can be modified "on the fly" without any maintenance. Develop full-featured multi-user turnkey systems with custom screens, windows, menus, reports and much more in days — not months! No more low-level programming, no time wasted . . .

MAGIC PC™

The *Visual* Database Language

TECH JOURNAL

"Magic PC's database engine delivers powerful applications in a fraction of the time . . . there is no competitive product."

PC MAGAZINE

"Overall, Magic PC is one of the most powerful DBMS packages available."

- **Quick Application Generator**
- **BTRIEVE® — based multi-user RDBMS**
- **Visual design language eliminates coding**
- **Maintenance-free program modifications**
- **Easy-to-use Visual Query-By-Example**
- **Multi-file Zoom window look-ups**
- **Low-cost distribution Runtimes**
- **OEM versions available**

ATTENTION BTRIEVE® USERS

Now you can quickly enhance your BTRIEVE®-based applications beyond the capabilities of XTRIEVE® and RTRIEVE®. Use Magic PC as a turn-key BTRIEVE® Application Generator to customize your applications without even changing your existing code.

AKER

19782 MacArthur Boulevard, Suite 305
Irvine, California 92715
TLX: 493-1184 FAX: 714-955-0199

DATABASE PROGRAMMERS

Join the thousands of professional database programmers and vertical market developers who switched to Magic PC from dBase®, R:BASE®, Paradox®, Clipper®, Dataflex®, Revelation®, Basic, C, Pascal, etc.

TRY BEFORE YOU PAY

We're so sure you'll love Magic PC — we'll let you try the complete package first. Only a limited quantity is available, so call us today to reserve your copy. Pay for Magic PC only after 30 days of working with it.* To cancel . . . don't call . . . simply return in 30 days for a $19.95 restocking fee.

OR PAY NOW AT NO RISK

Pay when you order and we'll wave the $19.95 restocking fee so you have absolutely no risk.

SPECIAL OFFER ~~$695~~

$199

Magic LAN multi-user — $399
Magic RUN — call for price

Order Now Call:
800-345-MAGIC

In CA 714-250-1718

MS

Add $10 P&H, tax in CA. International orders add $30.
*Secured with credit card or open P.O. Valid in US.
Dealers welcomed

CALL ADVERTISER DIRECTLY

Requires IBM/100% comp., 512K, hard disk. DOS 3.0 or later. Includes BTRIEVE runtime. Not copy protected. 2 5.25-inch disks. All trademarks acknowledged.

News & Views

by Sol Libes

Random Rumors & Gossip

Hewlett-Packard is rumored to be readying a new laser printer with an expected list price of $1,500. This means that the street price should be close to $1,000.

Apple Computer Inc. has announced that it will support Microsoft's LAN Manager running as a client workstation. And there are rumors that Apple will soon introduce an SE+ that supports color and uses a 68020.

AT&T has disclosed that it will ship beta UNIX V.4 source code in the first quarter of next year, nine months ahead of schedule. AT&T will also publish its UNIX System V Interface Definition (SVID) in the fourth quarter of 1988, also ahead of schedule. These disclosures are no doubt a response to the creation of the Open Software Foundation created by its UNIX competitors.

Compaq Computer is expected to finally enter the laptop competition with high-performance 286- and 386-based systems. Prototypes that have been shown to select customers weigh about 12 pounds and include 40-Mbyte, low-profile hard disks, rechargeable batteries, 1.44-Mbyte, 3½-inch floppies, 1-Mbyte of RAM and double supertwist LCD displays.

IBM has disclosed that it "plans to deliver a PS/2 capable of parallel processing before the year is out." In all likelihood, this will be a smart disk drive controller using the MicroChannel's multi-master feature—IBM's first product to use this much-publicized feature.

286 & 386 Clock Rates Jump

Most of the new AT-compatible machines now feature 20-MHz, 286 processors. Some vendors are boasting that these machines outperform many 386-based machines when coupled with disk caching and full-track buffering.

Most vendors are using 16-MHz, 286 chips from Harris and AMD and are screening them for reliable performance at 20 MHz. Intel's fastest 286 chip is still only rated at 12.5 MHz. Both Harris and AMD are expected to soon start shipping 286 chips rated at 20 MHz. This probably means that system vendors will soon offer AT compatibles running at 25 MHz.

Intel is now shipping small quantities of 386 chips rated at 25 MHz to selected customers (mostly their own personal computer group). Shipments to OEMs are expected to begin by year-end so that next year 25 MHz should become the standard rate for 386-based systems.

Intel has disclosed that it will deliver initial samples of an 80386 processor chip rated to run at 33 MHz to a select group of OEMs late this year. Intel is also expected to ship sample 386 motherboards using the chip before year-end. This probably means that we will see initial production machines running at 33 MHz by the middle of next year and that in 1990 this should become the *de facto* speed for 386 systems.

The standard 386 system sold next year should come with 4 Mbytes of memory on the motherboard (upgradeable to 16 Mbytes), include a 40-Mbyte hard disk, and have a street price of around $2,500.

Prior to the Intel disclosure, Motorola had announced that it would provide 33-MHz, 68030 samples in August and production parts by year-end. These parts will most likely be used by the UNIX system vendors, such as Sun Microsystems, as early as the first quarter of '89, while Apple will probably wait until late in the year to announce high-speed versions of the Mac II.

PS/2 Clone Update

Compaq and Zenith Data Systems have both publicly stated that they do not intend to introduce PS/2 clones employing IBM's patented MicroChannel Architecture. Both cited the MCA's poor performance as the reason. Andrew Czernek, a Zenith vice president, went so far as to say, "The MCA is an inferior product with inferior performance; We are not going to be swayed by the hype."

And Mike Swavely, a Compaq vice president said, "The PS/2 has been unable to deliver any significant advantage over what the standard PCs deliver." He went on to say that "the market hype that IBM put out about multiprocessors is really inaccurate. Every interrupt from any I/O device of any type gets processed with every memory cycle. There is not enough [bus] bandwidth to support [multiple processors]."

Compaq has acknowledged that it has MCA technology under development and that it is negotiating PS/2 patent licensing with IBM. There is no doubt that they will be watching the market's reaction to the Tandy and Dell Computer PS/2 compatibles closely, and if there is sufficient demand, they will also introduce such machines.

It is apparent that the old AT bus architecture still has sufficient bandwidth to match that of the current PS/2 machines. The dual-bus expansion for 32-bit memory adopted by Compaq and others for 386-based machines has enabled manufacturers to improve AT-bus machines performance beyond that of IBM's PS/2 machines. AST has already introduced a multiprocessor extension to the AT bus that may be adopted by other AT-bus machine manufacturers.

The UNIX Standard Battle

AT&T may have created UNIX, but they lost control over it when they licensed it to companies such as IBM and Microsoft who, in turn, relicense dit to distributors and resellers. Added to that are the enhanced versions from university labs. The result is creating problems for software developers, UNIX system vendors, and end users who need software portability. Binary file portability has proven to be a major problem and source code portability has created a thriving business for VARs and consultants.

UNIX vendors have long argued for a tightly controlled UNIX standard such as exists for DOS, OS/2, and Apple's Macintosh operating systems. Recently Sun Microsystems and AT&T entered into an agreement to develop a new "standard" version of UNIX and a "standard" graphical user interface called "Open Look." More than 50 UNIX system vendors, including Xerox and Unisys, agreed to support the standard. And software developers such as Lotus and Ashton-Tate have agreed to port applications to Open Look.

However, several key UNIX vendors are quite upset with the AT&T/Sun effort and have launched their own "Open Software Foundation" to develop another "standard" UNIX and user interface. The foundation includes such key UNIX players as IBM, DEC, Hewlett-Packard, and Apollo. It is unusual for these heavyweights to cooperate on anything, which indicates that they are seriously threatened by the AT&T/Sun effort. IBM and DEC want to keep their customers locked into their own operating systems and hence certainly do not want a UNIX standard that would allow users to buy systems on price and performance. HP and Apollo are firmly committed to UNIX and are fearful that AT&T has given Sun an inside track.

One thing is for sure—UNIX growth will continue to be hampered by the problem of standards. □

For UNIX® V.3/386 Call Microport.
For DOS Merge 286/386 Call Microport.
For Multiport Serial I/O Call Microport.
For TCP/IP Call Microport.
For The Fastest Compilers Call Microport.
For Spreadsheets Call Microport.
For Development Tools Call Microport.
For VAR/VAD Programs Call Microport.
For Database Systems Call Microport.
For RFS Call Microport.
For DWB 2.0 Call Microport.
For Micom/Excelan Call Microport.
For Word Processors Call Microport.
For Overnight Delivery Call Microport.
For UNIX V/286 Call Microport.
For Technical Support Call Microport.
For Everything Else Call Microport.

If you use or integrate PC-based UNIX® systems, 800-722-8649 is your number. Because for multi-user, multi-tasking operating systems, languages, applications, networking, and the expertise to make everything work together, the answer is Microport. For every UNIX question. In hardware. Software. Systems integration. And end-user support.

How can one company deliver all this for UNIX? Simple. We did the original port of System V to the IBM AT. And delivered the first commercial System V Release 3 for 80386-based PCs. We've integrated on all machines. In even the most unusual configurations. And over the years, we've built all the right contacts. So, for UNIX 286 or 386, our business is putting it together *for* you. Which means you'll always get solutions that work. Based on 100% standard System V.

Microport. One-stop shopping, fast friendly service, the lowest prices, and a long-term commitment to helping real people deliver real UNIX System V solutions. For questions, quotes, catalogs, and even overnight delivery, call us. We'll help you put it all together.

Real UNIX® from $199.

MICROPORT

800-722-UNIX®

Informix ■ Greenhills ■ Smartware ■ Micom ■ Word Perfect ■ Excelan ■ Unify ■ Uniplex ■ DOS Merge

10 Victor Square, Scotts Valley, CA 95066. (408) 438-8649; Telex: 249554 MICR UR; FAX: (408) 438-2511.
CIRCLE 111 ON READER SERVICE CARD

by Don Libes

Speeding Up *strcpy*

We use *strcpy* all the time and take it for granted. We assume that it not only works, but that it is optimally fast. The C idiom for string copying is stated so simply, it is hard to see at first how it could possibly be improved:

```
while (*dest++ = *src++) ;
```

Wrapping this phrase with formal parameters, and arranging for the return value gives us a complete *strcpy*.

```
char *strcpy(dest,src)
char *dest, *src;
{
   char *s = dest;

   while (*dest++ = *src++) ;

   return(s);
}
```

Surprisingly, this is not necessarily the fastest implementation of *strcpy* for most machines and most compilers. For example, some machines have a *strcpy*-like machine instruction. This obviously makes things much easier. For now, however, I will discuss the situation most of us face.

A dramatic improvement can be made by having the compiler put the *while* loop variables in registers. This can be done by modifying the parameter declaration so that it reads:

```
strcpy(dest,src)
register char *dest, *src;
```

Many compilers pass arguments on the stack and cannot obey this request directly. However, some compilers simulate it as if the code was written as follows:

```
strcpy(dest2,src2)
char *dest2, *src2;
{
   register char *dest = dest2;
   register char *src = src2;
```

Other compilers are not so smart. It may seem obvious how to optimize when looking at a simple subroutine like *strcpy*, but subroutines with many variables cannot be optimally compiled. Thus, it is generally not worth using register declarations on formal parameters unless you are coding for a particular compiler.

Notice that *strcpy* returns its first argument. It has never been clear to me why this is, since returning the end of the string is much more useful. For example, if you are going to append existing strings to a string you have just created, you will need to know where to start from. The only way to get this information is with *strlen*. *strlen* has to walk across every character in the string, which is exactly what *strcpy* does. What is the point of returning something that we obviously had to know to call the function in the first place? It is easy to fix *strcpy*. Let's call it *strcpye*.

```
/* strcpy, but return end of dest */
char *strcpye(dest,src)
char *dest, *src;
{
   while (*dest++ = *src++) ;

   return(dest-1);
}
```

Having a pointer to the end of a string is very useful. For example, we can calculate the length of the string by subtracting the beginning from the end. We can also reverse this operation assuming we have the length and want to calculate the end of the string. Sometimes we may already know the length of a string without having to call *strlen* or another string routine. In that case, we can do string copying even faster.

Many machines have single machine instructions that can copy a given number of bytes. These are available through the C library function called *memcpy*. *bcopy* is another name used by some systems, but it does the same thing as *memcpy*. *memcpy* is the name given this function by the ANSI C standard.

memcpy is faster than *strcpy*, so if you already have the length or end of a string, you should call *memcpy* instead of *strcpy*. Unfortunately, both *memcpy* and *strcpy* handle overlapping strings in implementation-defined ways. This is noted by the ANSI C standard.

Using the *strcpy* we defined earlier, the result of the following code leaves *s* with the string "model."

```
char s[80] = "/model";

/* strip leading slash */
strcpy(s,s+1);
```

Attempting to add space into the string fails:

```
strcpy(s+1,s);
```

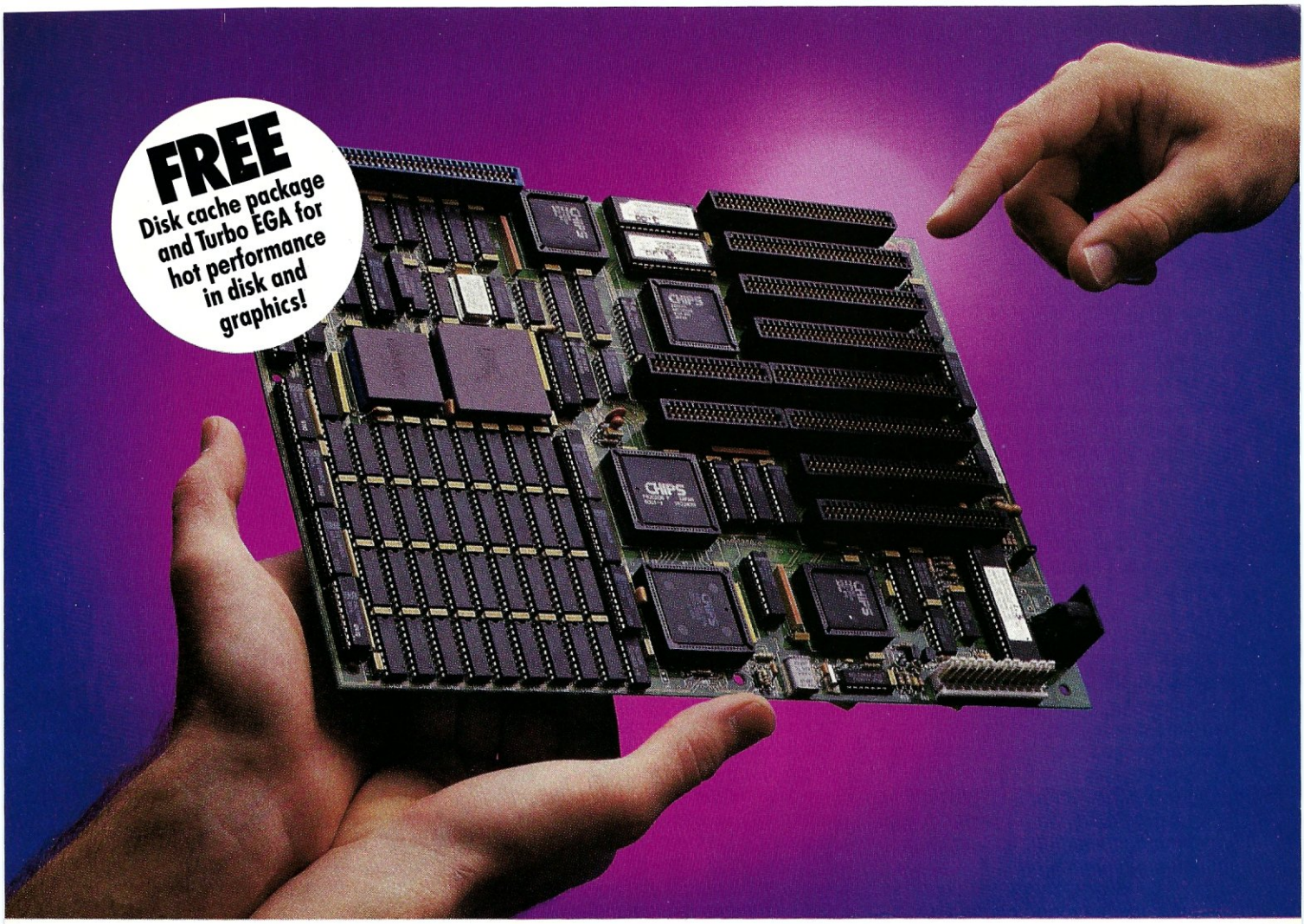
In particular, *s* will be filled with slashes, and *strcpy* will attempt to fill all remaining memory with slashes. The problem is that *strcpy* copies each character over the next one that it is about to use. It will never see the terminating null because it will have already written over it with a slash.

Reversing *strcpy* so that it copies rear to front fixes this problem but then breaks *strcpy(s,s+1)*. Reverse copying is also slower since we have to find the end of the string. One solution is to compare the pointers to each other. If the source is greater than the destination, *strcpy* runs from the front of the string to the rear, otherwise it runs from the rear to the front. Let's call this *strcpyo*.

```
/* strcpy that handles overlap */
char *strcpyo(dest,src)
char *dest, *src;
{
   return(src>dest?
     strcpy_forward(dest,src):
     strcpy_backward(dest,src));
}
```

A lot of string processing is done by *ad hoc* code because the string routines in the libraries don't quite match the programmer's requirements. My comment about returning string ends rather than string beginnings is one such example. Another is requiring *strcpy* to behave a particular way when dealing with overlapping strings. This is an unfortunate fact of life spelled out by the ANSI C standard.

Don Libes is a computer scientist based in the Washington, D.C., area working in artificial intelligence and robot control systems.



CONVERT YOUR PC, XT OR AT INTO A HIGHER FORM OF LIFE!

386 MOTHERBOARDS FOR 386 SPEED

Don't let your PC give up the ghost—Hauppauge has just arrived with a new spark of life: the 386 MotherBoard.™ Far more advanced than an accelerator card, our line of MotherBoards grace your PC, PC/XT, PC/AT or compatible with speeds equal to the IBM PS2 Model 80. And faster. Because we've built in 1 Megabyte of high speed RAM and a 387 math coprocessor socket for speeds that make users humble with awe.

OS/2 Compatible. To ensure a long, fruitful life, our 386 MotherBoard is compatible with the PC/AT (BIOS and I/O)—so you can run the new generation of DOS, OS/2. Our Board also runs Windows/386, UNIX V and PC-MOS/386. For more power, you'll find 16-bit expansion slots that accommodate the latst I/O expansion cards. No 386 accelerator card gives you so much versatility. **Only our 386 MotherBoard gives your PC a future with unlimited possibilities!**

The Critics Applaude! *PC Magazine* awarded our Board "The Editor's Choice" for 386 Replacement Boards. *PC World* called it "the Upgrade Product of the Year."

Technical Features ▪ 16 MHZ 80386 ▪ 1 Megabyte of 100 nsec 4-way interleaved RAM ▪ PC/AT compatible I/O and BIOS for support of OS/2 ▪ Six 8-bit expansion slots ▪ two 16-bit expansion slots (four on 386 MotherBoard/AT) ▪ One 32-bit expansion slot for up to 12 Megabytes of high speed memory ▪ Battery-powered clock calendar

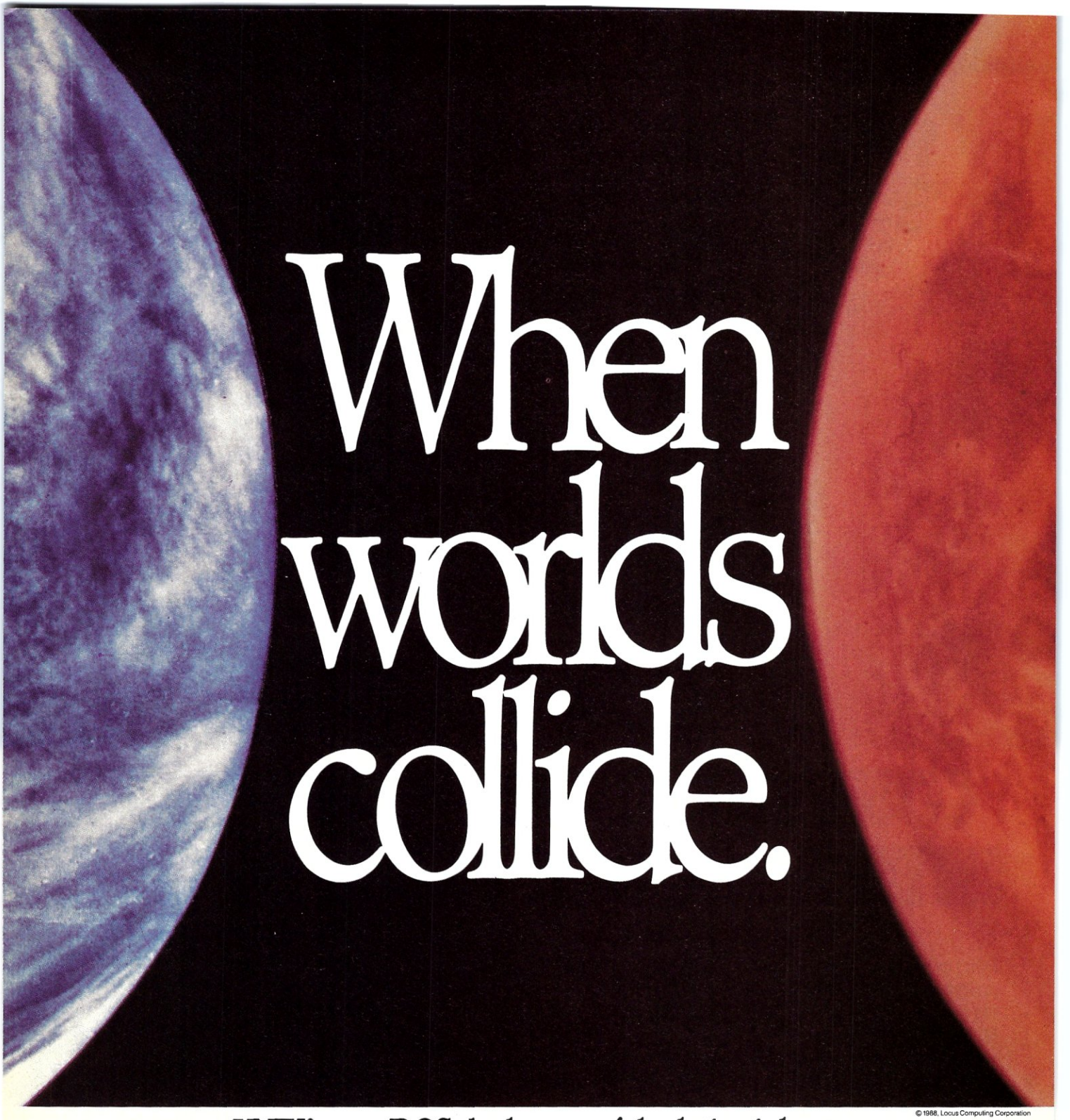
386 MotherBoard/PC or MotherBoard/XT	$1495
386 MotherBoard/AT	$1595
32-bit RAM Board (2 Mbytes installed; up to 4 Mbytes)	$795
16 MHz 80387 math coprocessor	$695
16-bit combination hard disk/ floppy disk controller	$245

For more information on our easy-to-install MotherBoard, call: **1 (800) 443-6284**. In New York, call **(516) 434-1600**.

Hauppauge Computer Works, Inc.
175 Commerce Drive,
Hauppauge, New York 11788

Hauppauge!

CIRCLE 105 ON READER SERVICE CARD



UNIX® meets DOS. And a powerful solution is born.

Until now, UNIX users coveted the wealth of application software available on DOS. While DOS users envied the power of their UNIX counterparts.

Now Locus Computing gives every user the best of both worlds. And more. In a family of software that puts DOS applications and UNIX power at their fingertips.

Merge 386™ uses a single 80386 processor to run multiple DOS environments concurrently, from within UNIX System V.3. Sharing all system resources easily, including record level access to the same files.

PC-Interface™ connects multiple PCs running DOS with a UNIX host computer. Users run DOS applications with centralized files stored on a UNIX host. Or work in a UNIX environment from their PC workstation. It's their choice.

Xsight™ windowing software gives PC workstations and host systems alike an easy-to-use graphic interface and lets you move smoothly between single system and networked configurations. So you preserve your investment in equipment and training while your systems evolve with your business.

For more information, write Locus Computing at 3330 Ocean Park Boulevard, Santa Monica, CA 90405. Or call (213) 452-2435. FAX: (213) 450-1432. And marvel at the delights of a new world of connectivity.

Locus
Great ideas connect

PC-Interface, Merge 386 and Xsight are trademarks of Locus Computing Corporation. UNIX is a registered trademark of AT&T. In Europe, contact Sphinx, Ltd., 43-53 Moorbridge Road, Maidenhead, Berkshire SL6 8PL, England. Telephone (44) 628 75343.

CIRCLE 108 ON READER SERVICE CARD

Making *strcpy* Fast on a RISC

Complex machine architectures demonstrate the problems of producing optimal code. For example, the VAX has a single instruction that can copy blocks of arbitrary length, and another that can locate the ends of null-terminated strings. Using these, it becomes very easy to write an efficient *strcpy* (as well as *strlen*, *strcat*, and others). The first instruction finds the end of the string, and the second uses it to do the copy. That's two machine instructions!

In the case of the VAX, this worked fine. However, when a new machine was introduced, the MicroVAX II, it turned out that the engineers couldn't fit all the old VAX microcode into the microcode space of the MicroVAX. The obvious solution was to emulate the instructions in software. This worked fine except that *strcpy* now executed much more slowly than the original test-and-branch loop! The reason was that the optimized (two-instruction) version traversed the source string twice, while the unoptimized (C while loop) version traversed it only once.

This demonstrates a general problem of complex architectures, often referred to as CISC (for Complex Instruction Set Computers); they cause problems for compiler writers trying to create optimal compilers and libraries. Implementors generally have to learn large sets of complex instructions and gain a much deeper level of understanding about how they execute. For example, instruction timings vary, depending upon context, because of caching and pipelines. While some of these factors affect RISC machines as well, all RISC instructions take one cycle (well, almost all do), which makes instruction choices much simpler. The result is that the time you spend optimizing a compiler for a RISC machine will be less than for a CISC machine.

When the next version of your RISC machine appears, you won't have to spend time rewriting the optimizer to account for new instruction timings. The VAX—clearly a CISC machine—presentsexactly this problem because there are so many models. The solution is to make the code straightforward, and avoid the specialized instructions that are slow or aren't implemented on some machines.

By analogy, you can use the same reasoning for per-compiler optimization. Since your C code may run on many different compilers, operating systems, and machines, it is often better to write straightforward code that maximizes portability. Super-optimizing code is difficult. Are you sure it is better to code it this way on an 8088/8086? 80286? 80386? 68000? VAX? SPARC? Cray? Let the compiler do the optimization.

Loop Unrolling

People familiar with compiler writing have doubtless seen the following trick. The C implementation was first attributed to Tom Duff of Bell Laboratories and is known as "Duff's Device":

```
switch (n&7) {
   do {
            *dest++ = *src++;
      case 7:  *dest++ = *src++;
      case 6:  *dest++ = *src++;
      case 5:  *dest++ = *src++;
      case 4:  *dest++ = *src++;
      case 3:  *dest++ = *src++;
      case 2:  *dest++ = *src++;
      case 1:  *dest++ = *src++;
      case 0:  ;
   } while (0 <= n-=8);
}
```

My compiler said, "warning: loop not entered through top," but compiled it correctly. Aside from demonstrating the bizarreness allowable with switches, this piece of code forces an optimization known as loop unrolling. The idea is that code you expect to loop many times can be sped up by not checking the termination condition each time through the loop. In this implementation, we check once every eight assignments. We handle the remainders separately.

For example, if *n* is 10, the switch branches to case 2. Case 2 falls into case 1, thereby executing two assignments (notice the missing breaks), and the *while* returns us to the top of the loop to execute the remaining eight assignments with no intervening tests.

This would make the heart of a very fast *memcpy*. Notice that it is not dependent upon the data types. You can copy by *char*, *ints*, *longs*, or even *structs*—whatever you decide. However, there are obvious trade offs you will have to consider. For example, while larger types can store as quickly as smaller types, larger types often

require alignment or you pay a speed penalty. Compilers can do loop unrolling automatically, but only if they are assured of a payoff. This is difficult in C programs, because so many miscellaneous functions call *memcpy*. You might like to create a second version of *memcpy* that has the optimization in it. Depending on the situation, you will want to call the copy or the original.

A Real Fast *strcpy*

Why did I bring up loop unrolling? Partly to demonstrate it, but also to improve *strcpy*. *strcpy* can use loop unrolling, although it is a little trickier than *memcpy*. The actual assignment is easy; the hard part is the termination condition.

Suppose we are copying four bytes at a time. We want to stop if any of the four bytes are null. The obvious test is:

```
(*x == 0) || (*(x+1) == 0) ||
(*(x+2) == 0) || (*(x+3) == 0)
```

Unfortunately, implementation of this expression is more expensive than the simple test we were hoping to avoid. Fortunately, there is a way to detect null bytes in a multibyte constant with only a few machine instructions. The following macro does the job:

```
#define has_null_byte(x) \
   ((x - 0x01010101) & ~(x) &\
     0x80808080)
```

This particular version is written for four-byte unsigneds on a 2's complement machine. The basic idea is that the subtraction propagates bits up to the most significant bit in each byte by successive carries. If you want to really understand this, try running an example by hand and watching how the bits move.

Here is a straightforward rewrite of *strcpy* using this macro:

```
#define has_null_byte(x) \
   ((x - magic1) & ~(x) & magic2)

char *real_fast_strcpy(dest,src)
char *dest, *src;
{
   register long *d = (long *)dest;
   register long *s = (long *)src;
   register long magic1 =
     0x01010101;
   register long magic2 =
     0x80808080;
   register char *dc, *sc;

   /* copy four bytes at a time */
   while (!has_null_byte(*s)) {
     *d++ = *s++;
   }

   /* prepare to copy remaining
    chars */
   dc = (char *)d;
   sc = (char *)s;
   while (*dc++ = *sc++) ;

   return(dest);
}
```

real_fast_strcpy copies four bytes at a time, once each block is determined to be null free. If there is a null, it just reverts back to the old behavior. Since this only happens close to the end of a string, there isn't much penalty for this.

By declaring *d* and *s* as *longs* (4 bytes on my machine) I am able to have the compiler provide the fastest way of copying. This will turn out to be a single instruction on most machines.

Notice all the register declarations, and how they are ordered. *s* and *d* are going to be accessed the most frequently so they come first. I modified the macro so the magic numbers are stored in registers. Only if the compiler has any registers left over, will the old-style *strcpy* get the benefit of using them.

As I remarked earlier, heavily optimized code like this starts to sacrifice portability. In this case, I assume I know how *longs* are stored, assigned and mathematically manipulated. I also hope that I'm going to get several registers for this to pay off. The worst aspect of all is that I have to guarantee that none of my strings are butted up against the end of my address space, and if one is that it is aligned on a *longword* boundary. While such an occurrence is extremely unlikely, the algorithm will cause an address fault when *has_null_byte* attempts to look at several nonexistent bytes at the end of the string.

sprintf

One final note. While *sprintf* may seem like overkill for a lot of simple string manipulations, it is defined to return the total number of bytes written (according to ANSI C, others may vary). Adding this to the beginning of the string gives you the end of the string. This avoids exactly the problem I was complaining about earlier with *strcpy* and friends. If you are writing code that doesn't need to be as fast as possible, don't bother with the low-level functions—use *sprintf* for everything! □

Did you find this article particularly useful? Circle number 1 on the reader service card.

C CODE FOR THE PC

source code, of course

NEW!	MS-DOS File 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
	PforC or PforCe++ (COM, database, windows, file, user interface, DOS & CRT)	$345
	CQL Query System (SQL retrievals plus windows)	$325
	GraphiC 4.1 (high-resolution, DISSPLA-style scientific plots in color & hardcopy)	$325
	Barcode Generator (specify Code 39 (alphanumeric), Interleaved 2 of 5 (numeric), or UPC)	$300
NEW!	Vmem/C (virtual memory manager; least-recently used pager; dynamic expansion of swap file)	$250
	PC Curses (Aspen, Software, System V compatible, extensive documentation)	$250
	Greenleaf Data Windows (windows, menus, data entry, interactive form design)	$250
	Vitamin C (MacWindows)	$200
	TurboTEX (TRIP certified; HP, PS, dot drivers; CM fonts; LaTEX)	$170
	Essential resident C (TSRify C programs, DOS shared libraries)	$165
	Essential C Utility Library (400 useful C functions)	$160
	Essential Communications Library (C functions for RS-232-based communication systems)	$160
	Greenleaf Communications Library (interrupt mode, modem control, XON-XOFF)	$150
	Greenleaf Functions (296 useful C functions, all DOS services)	$150
	OS/88 (U**x-like operating system, many tools, cross-development from MS-DOS)	$150
	ME Version 2.0 (programmer's editor with C-like macro language by Magma Software; Version 1.31 still $75)	$140
	Turbo G Graphics Library (all popular adapters, hidden line removal)	$135
	PC Curses Package (full Berkeley 4.3, menu and data entry examples)	$120
	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
	The Profiler (program execution profile tool)	$100
	Entelekon C Function Library (screen, graphics, keyboard, string, printer, etc.)	$100
	Entelekon Power Windows (menus, overlays, messages, alarms, file handling, etc.)	$100
NEW!	TurboGeometry (library of routines for computational geometry)	$90
NEW!	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
	Professional C Windows (windows and keyboard functions)	$80
	JATE Async Terminal Emulator (includes file transfer and menu subsystem)	$80
	MultiDOS Plus (DOS-based multitasking, intertask messaging, semaphores)	$80
	WKS Library (C program interface to Lotus 1-2-3 program & files)	$80
	Professional C Windows (lean & mean window and keyboard handler)	$70
NEW!	lp (flexible printer driver; most popular printers supported)	$65
	Quincy (interactive C interpreter)	$60
	EZ_ASM (assembly language macros bridging C and MASM)	$60
	PTree (parse tree management)	$60
	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
	Vector-to-Raster Conversion (stroke letters & Tektronix 4010 codes to bitmaps)	$50
	Coder's Prolog (inference engine for use with C programs)	$45
NEW!	Virtual Memory System (least recently used swapping)	$40
	C-Notes (pop-up help for C programmers ... add your own notes)	$40
	Biggerstaff's System Tools (multi-tasking window manager kit)	$40
	PC-XINU (Comer's XINU operating system for PC)	$35
	CLIPS (rule-based expert system generator, Version 4.1)	$35
	Tiny Curses (Berkeley curses package)	$35
	TELE Kernel or TELE Windows (Ken Berry's multi-tasking kernel & window package)	$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 (a dozen file compression & expansion programs)	$30
	ICON (string and list processing language, Version 7)	$25
NEW!	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
	C Compiler Torture Test (checks a C compiler against K & R)	$20
	Benchmark Package (C compiler, PC hardware, and Unix system)	$20
	TN3270 (remote login to IBM VM/CMS as a 3270 terminal on a 3274 controller)	$20
	A68 (68000 cross-assembler)	$20
	List-Pac (C functions for lists, stacks, and queues)	$20
	XLT Macro Processor (general purpose text translator)	$20
NEW!	C/reativity (Eliza-based notetaker)	$15
	Data	
	WordCruncher (text retrieval & document analysis program)	$275
	DNA Sequences (GenBank 52.0 including fast similarity search program)	$150
	Protein Sequences (5,415 sequences, 1,302,966 residuals, with similarity search program)	$60
	Webster's Second Dictionary (234,932 words)	$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

acw!info@uunet.uu.net

Voice: (512) 258-0785
BBS: (512) 258-8831
FidoNet: 1:382/12

Free shipping on prepaid orders **For delivery in Texas add 7%** **MasterCard/VISA**

by Phil Hughes

Connecting UNIX Tools to Create A Program Solution

Since this is my first contribution to *The UNIX File*, a quick introduction is in order. I am one of the founders of Specialized Systems Consultants (SSC), a computer consulting and training company. I started working with UNIX in 1980 when I wrote a requirements statement for a computer system to be used for software cross-development. This resulted in the acquisition of a UNIX-based system. I installed the system and then trained the company employees in the use of that system and the C programming language. Since that time I have been involved in similar activities for other companies.

In 1983, SSC decided to specialize in UNIX systems exclusively and we acquired a second UNIX system—an IBM XT-class machine running Venix, a version of UNIX developed by VenturCom. Since that time we have acquired two AT-class machines, one running SCO Xenix 286 and the other running Microport V/AT, as well as a 386-based machine running SCO Xenix 386. Today, all computing needs at SSC are handled by UNIX-based systems, with the exception of graphics generation which is done on an Atari ST.

At SSC we address four product areas: consulting, training, publishing, and software. Software development and distribution are all performed on UNIX systems using the standard UNIX tools. We even have a dial-in demonstration system that was developed using UNIX shell scripts. Training classes are hands-on using a 386-based system.

In-house, the most interesting use of the UNIX system is in our production of publications. We produce all of our own catalogs, training notebooks, and a series of Pocket Reference booklets on C and UNIX. All of these products are created using the UNIX text processing system. We have relied heavily on the UNIX tools to make this an easy and efficient process.

Most of our computer needs have been satisfied with programs developed in-house and the use of shell scripts and UNIX utilities. The exceptions are a database system (Progress) to handle our customer lists and a filter that converts *troff* output to PostScript (*devps*) so we can proof our publications on a laser printer and then set type on a Linotronic typesetter.

Now that you know about my UNIX background and experience, let me address what you will be seeing in this column in the future. There will be a mixture of tips and techniques, things I have learned from my work with UNIX systems over the past eight years, and a discussion of new UNIX-related products. Emphasis will be on products that are useful for computers based on the PC and AT bus. I also encourage you to write with questions or suggestions for future columns.

Solving Problems With UNIX Tools

This month I want to talk about why UNIX should be considered a tool kit for solving problems. Most of us with a technical background think of a programming language such as C as the right way to solve many programming problems. Those with a less technical background are more likely to think in terms of using a BASIC program or possibly a database management system. With most operating systems, DOS being a good example, this is probably the reasonable way to approach a problem.

Listing 1. The script file

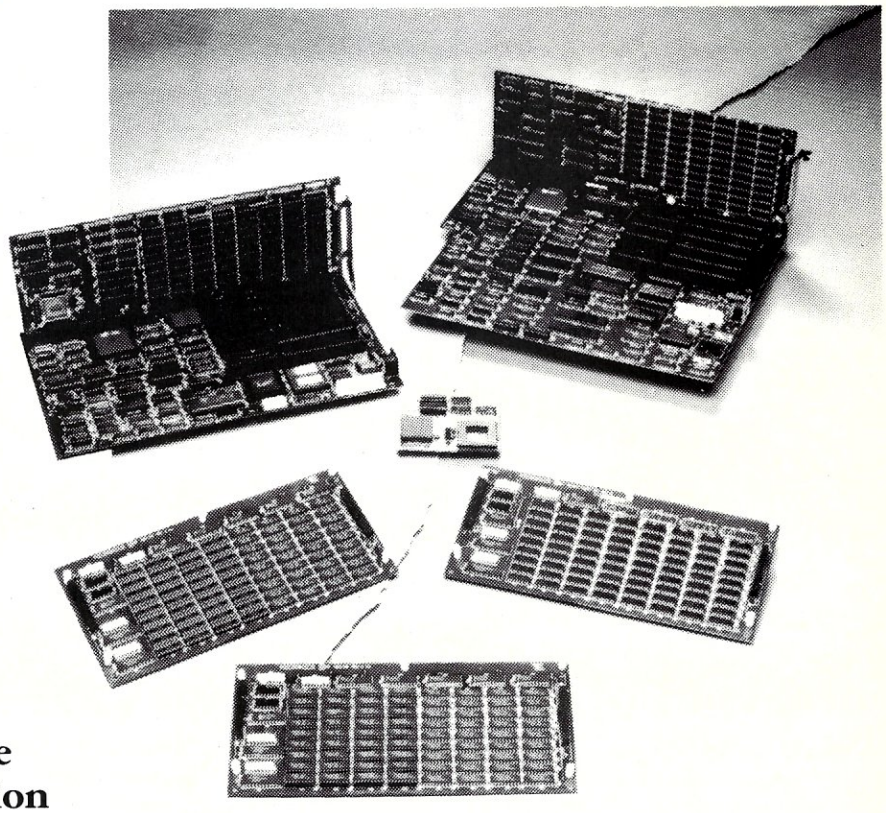
```
$ cat smry
: smry -- produce phone bill summary
# sort phone bill in phone number order
sort -o phone.bill -t'|' +0 -1 phone.bill
# sort phone list in phone number order
sort -o phone.list -t'|' +1 -2 phone.list
# output all lines from phone.bill with matching records from phone.list
# then produce summary and totals by using awk
join -a2 -t'|' -j1 2 phone.list phone.bill | awk -F'|' '
BEGIN{
print "phone bill summary\n"
print "Name or Company                number          amount"
number = "dummy"
}
{
if ($1 != number)
{
if (number != "dummy")
printf "%-30.30s %-15.15s %6.2f\n", \
 name, number, total
grand_total += total
total = 0
number = $1
if (NF == 3)
name = $2
else
name = "    ????"
}
total += $NF
}

END{
if (total > 0)
{
grand_total += total
printf "%-30.30s %-15.15s %6.2f\n", \
 name, number, total
}
print "\ntotal bill is $" grand_total
}

$
```

Phil Hughes is vice president of Specialized Systems Consultants Inc. (SSC). Located in Seattle, Washington. SSC offers pocket reference booklets for various UNIX systems and utilities, and software, and offers training and consulting to UNIX users.

Insist On The Best 386 Micronics Motherboards



Quality, Performance and **Innovation** best describe our 80386 based board level product line. Now with both **AT** and **Baby size** and high speed **CACHE** memory. Micronics is the leading supplier to **OEMs, VARs** and Systems Integrators that require the **best in 80386 technology.**

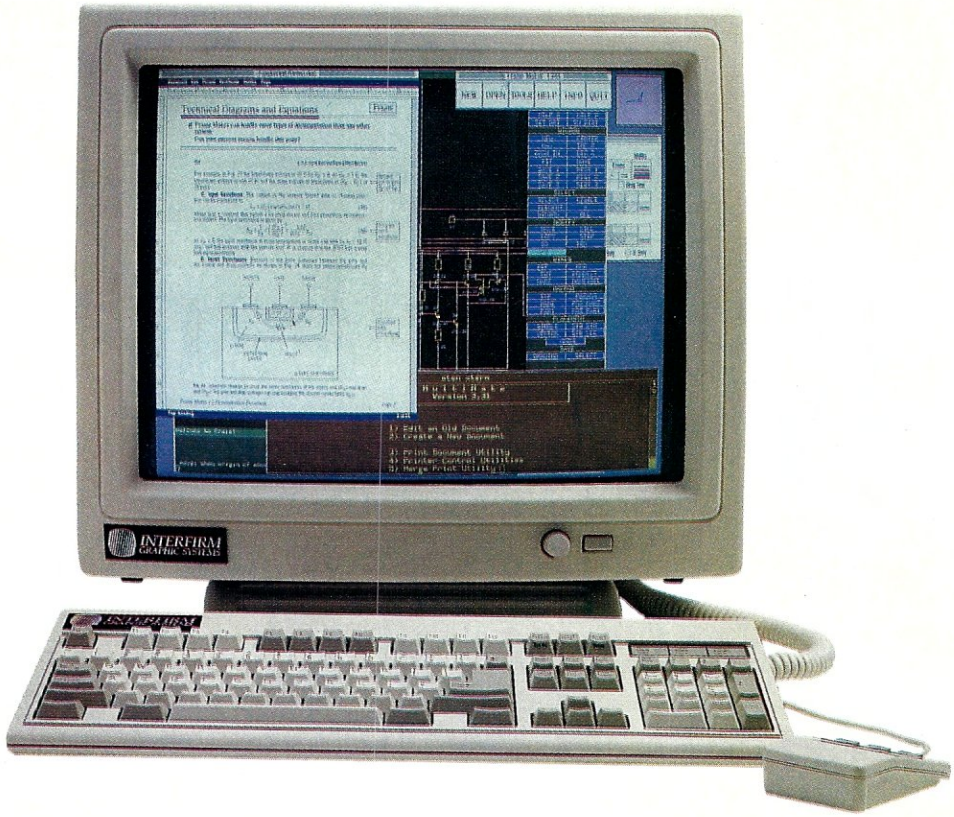
For a distributor near you, call **800 / 234-4386.**

Innovation
and
Performance

MICRONICS
COMPUTERS INC.

CIRCLE 113 ON READER SERVICE CARD.

Get a window on the future . . .



without giving up the present

At last, you can have the benefits of UNIX™ V.3 and high resolution graphics through X-Windows—without sacrificing the investment you've made in your DOS based 80386™ system.

The Easy Way to X-Windows

With the XSUBSYSTEM™ from Interfirm Graphic Systems, you'll have everything you need to convert your 80386-based machine into a powerful UNIX-based graphics workstation. Suddenly, you'll have all of the advantages of a multi-user, multi-tasking system—plus the incredible convenience of DOS programs runnng side-by-side with UNIX.

The XSUBSYSTEM Includes:

- 19" High resolution color or monochrome display
- Graphics controller
- UNIX System V.3 from Interactive Systems Corp.
- The X Window System software
- Optional hard disk with all software pre-installed
- Optional TCP/IP Ethernet controller and software

The Future of 386 Based Graphics

The XSUBSYSTEM gives you the best of both worlds - you get to keep both your hardware and your DOS software while you upgrade to a UNIX V.3/X-Windows System.

For example, with Interfirm's XSUBSYSTEM, you can take data created in a DOS application and paste it into a document that you are creating with the Frame Maker™ desktop publishing system.

It's the most sensible way we know to give your 386 based system the power of a true graphics workstation - at a sliver of the price.

At Interfirm Graphic Systems, we are pioneering UNIX-based high-performance graphics, utilizing the X Window System and other emerging non-proprietary standards. When you look to the future of desktop computing, look to Interfirm.

INTERFIRM
GRAPHIC SYSTEMS
3940 Freedom Circle
Santa Clara, CA 95054
(408) 970-7070
800-338-4513

With the UNIX system there is such a rich supply of tools. Many times you can connect a few of these tools together to produce the desired result. Even if this approach isn't efficient in terms of computer resources, it may offer a quick way to prototype a solution. Once it is running, programs can be written to handle the time-consuming parts in a more efficient manner.

To help you look at possible uses for these tools, I offer the following problem. Every month you get a telephone bill and then stare at it for a while attempting to figure out the long distance charges. You look up a few numbers, remember a few, and finally accept the total as being close enough and pay the bill. Many of us have some sort of phone number list on a computer system. If we just had a way to extract the name or company information by phone number and match it up with the bill, it would be easy to enter numbers and amounts and produce a report showing who was called and what it cost.

The production of this kind of report is a mundane programming task. Print out a summary line for each phone number with the amount and print a grand total. This could be easily written in any programming language. The hard part of the project is how to match up the phone numbers in the bill with those in the name/phone number data file, and how to handle the numbers that don't match—the numbers that are on the bill but not in the name/number data file.

Join, a UNIX program that joins two tables, to the rescue. *Join* will match two files and print-selected fields from each file. It will also optionally print out the lines in one file that have no match in the other file. The only input requirements are that the files be sorted by the fields to be used for the match and that there be some sort of field separator character.

Sorting can be handled by the *sort* program, another standard UNIX utility. Just to keep things in the UNIX utility ballpark, I chose to use *awk* as a report generation program to produce the output. Listing 1 shows the script that invokes *sort*, *join*, and *awk*. Example 1 shows the two data files, PHONE.BILL and PHONE.LIST. Finally, Example 2 shows the output from running the script.

In this discussion, I want to concentrate on the *sort* and *join* commands. *Awk* was discussed by Ian Darwin in "The UNIX File" column (*Micro/Systems Journal*, July/August 1987). I will discuss *awk* further in a future column.

First, the two *sort* lines. The *–o* option tells *sort* that the next argument is the name of the output file. By using this option it is possible to sort a file "in place." In other words, the sorted data ends up back in the original file. The *–t* option is used to specify the field separator character. If you look in Example 1, you will see that I used a | as the field separator. It was necessary to quote the character in the *sort* command line as it has special meaning to the shell. Finally the *+0* and *–1* options in the first *sort* command tell *sort* to skip no fields to get to the *sort* key and skip one field to get to the end of the *sort* key; in other words, sort on the first field. Looking at Example 1 again, you can see that the telephone number is the first field in the PHONE.BILL file.

In the *sort* example, the *+1* and *–2* options are used. This directs *sort* to skip one field to get to the beginning of the *sort* key and two fields to get to the end of the *sort* key. In other words, sort on the second field. Looking at Example 1, you can see that the telephone number is the second field in the PHONE.LIST file.

Once the two files are sorted into telephone number order the match operation can be performed. This is done by *join*. The field separator character is specified with the *–t* option, the same as with *sort*. The *–j1 2* option instructs *join* to perform the match on the second field in the first file. By default, the first field is used so the match specified in this command is the first field of the second file with the second field of the first file. The *–a2* option tells *join* to print any lines from file 2 that have no match in file 1. File 1 is the PHONE.LIST file and file 2 is the PHONE.BILL file, so *join* will print any lines that appear in the phone bill that are not in the name/phone list.

Example 3 shows the output of *join*. In the script this is used as input to the *awk* program that prints a formatted report, but this processing could have been done with a program in any language. Note that the output is three fields for the lines that appear in both files, but two fields for those phone numbers on the bill that are not in the phone list. This is how the report program can determine if the number is known. In my report program I print ???? for the name if it is unknown.

The data from the phone bill can be entered using a text editor. In fact, you may already have the phone list in this form. If not, *awk* can probably be used to reformat it. The end result is a workable system with no software purchases and about an hour of work. □

Did you find this article particularly useful? Circle number 2 on the reader service card.

Example1. The two data files

```
$ cat phone.list
Micro/Systems Journal|201-522-9347
UNIX User|213-372-9917
Office of the Americas|213-451-9761
tecNICA|415-848-0292
Mike Lowry|442-7170
Metro|447-4800
Phil Hughes|527-3385
Freedom Fund|547-7644

$ cat phone.bill
201-522-9347|1.68
201-522-9347|4.15
213-372-9917|3.71
213-555-1212|.75
415-848-0292|2.17

$
```

Example 2. Output produced by running the script file

```
$ smry
phone bill summary

Name or Company              number          amount
Micro/Systems Journal        201-522-9347     5.83
UNIX User                    213-372-9917     3.71
    ????                     213-555-1212     0.75
tecNICA                      415-848-0292     2.17

total bill is $12.46

$
```

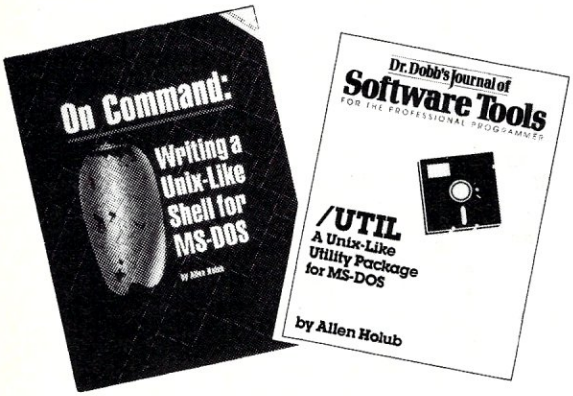
Example 3. The output produced by *join*

```
$ join -a2 -t'|' -j1 2 phone.list phone.bill
201-522-9347|Micro/Systems Journal|1.68
201-522-9347|Micro/Systems Journal|4.15
213-372-9917|UNIX User|3.71
213-555-1212|.75
415-848-0292|tecNICA|2.17

$
```

Bring the Conveniences of UNIX To YOUR MS-DOS Machine

Full Source Code on Disk!



On Command: Writing a UNIX-Like Shell for MS-DOS

by Allen Holub

This book and ready-to-use program demonstrate how to write a UNIX-like shell for MS-DOS, with techniques applicable to most other programming languages as well. The book and disk include a detailed description and working version of the Shell, complete C source code, a thorough discussion of low-level MS-DOS interfacing, and significant examples of C programming at the system level.

Supported features include: read, aliases, history, redirection and pipes, UNIX-like command syntax, MS-DOS compatible prompt support, C-Shell based shell scripts, and a Shell variable that expands to the contents of a file so a program can produce text that is used by Shell scripts.

The ready-to-use program and all C source code are included on disk. For IBM PC and direct compatibles.

Book & Disk (MS-DOS) **Item #29-1** **$39.95**

/Util

by Allen Holub

When used with the Shell, this collection of utility programs and subroutines provides you with a fully functional subset of the UNIX environment. Many of the utilities may also be used independently. You'll find executable version of the cat; cp; date; du; echo; grep; ls; mkdir; mv; p; pause; printevn; rm; rmdir; sub; and chmod.

The /Util package includes complete source code on disk. All programs and most of the utility subroutines are full documented in a UNIX-style manual. For IBM PC and direct compatibles.

Manual & Disk (MS-DOS) **Item #12-7** **$29.95**

NR: An Implementation of the UNIX NROFF Word Processor

by Allen Holub

NR is a text formatter that is written in C and compatible with UNIX's NROFF. Complete source code is included in the **NR** package so that it can be easily customized to fit your needs. **NR** also includes an implementation of how -ms works. **NR** does hyphenation and simple proporational spacing. It supports automatioc table of contents and index generation, automatic footnotes and endnotes, italics, boldface, overstriking, underlining, and left and right margin adjustment. **NR** also contains:

- Extensive macro and sting capability
- Number registers in vaious formats
- Diversions and diversion traps
- Input and output line traps

NR is easily conmfigurable for most printers. Both the ready-to-use program and full source code are included. For PC compatibles.

Manual & Disk (MS-DOS) **Item #33-X** **$29.95**

SAVE 15%

Receive **On Command, Util** and **NR** together for only $85.95! You get all the convenience of UNIX-like features and save 15%.

UNIX-like Features Package **Item #167** **$85.95**

To Order: Return this order form with your payment to:
M&T Books, 501 Galveston Drive, Redwood City, CA 94063
Or, **CALL TOLL-FREE 800-533-4372** Mon-Fri 8AM-5PM (In CA call 800-356-2002)

Name _____

Address _____

City _____ State _____ Zip _____

☐ ***YES!*** Please send me the following:

UNIX-like Features Package	$85.95
On Command book & disk	$39.95
/Util manual & disk	$29.95
NR manual & disk	$29.95
Subtotal	
CA residents add sales tax ____ %	
Add $2.95 per item for shipping	
Total	

☐ Check enclosed. Make payable to M&T Publishing.

Charge my: ☐ Visa ☐ M/C ☐ Am.Ex.

Card No. _____ Exp.Date _____

Signature _____

3078F

OS/2 or UNIX?

How are OS/2 and UNIX the same and where do they differ? And what do you have to be concerned with to port DOS applications to these multitasking platforms?

by William G. Wong

OS/2 is taking developers by storm, although it may rain for awhile before we see applications that will make users want to buy OS/2. And even though UNIX has been available on PCs for a much longer time, users are not flocking to that platform in droves either. Both OS/2 and UNIX have great potential, both provide advanced features for the user and programmer, and both have yet to really fulfill their potential.

The main thing holding back OS/2 and UNIX right now is DOS (PC-DOS and MS-DOS). DOS's popularity has been brought about by the abundance of applications and low-cost PCs. Both OS/2 and UNIX require a great deal of resources compared to DOS. The normal low-end platform consists of an 80286 PC with 2 megabytes of memory and a 30-MB hard disk. A more suitable platform is an 80386 with 4 MB of memory, a 60-MB hard disk, a tape backup unit, and an intelligent communications controller. OS/2 is primarily designed to run a system as a node within a local area network or attached to a mainframe or mini. UNIX can be used as a single-user system, but multiuser configurations are more common.

Version 1.0 of OS/2 provides a DOS-like command-line interface. UNIX provides a *shell* that is also a command-line interface. Actually, the two are quite similar in function and really only differ in syntax. However, OS/2's similarity to DOS gives it an advantage with DOS users.

The program interfaces for OS/2 and UNIX are also similar in conventional system and file operations, as well as multitasking facilities. These similarities become a major issue when considering design of new applications and porting of existing DOS applications because both OS/2 and UNIX provide potential platforms for new applications.

This article addresses the similarities and differences between UNIX, OS/2, and DOS starting with the user interface. The general programming interface for OS/2 and UNIX will be compared, as well as the implications of graphics front ends (Presentation Manager for OS/2 and X Windows for UNIX). Finally, the issue of porting DOS applications to OS/2 and UNIX will be addressed.

User Interface

The infamous *A>* prompt is the hallmark of DOS, even though it did not originate there. (COMMAND.COM, the program that provides the DOS command-line interface, actually allows the prompt to be changed.) COMMAND waits for a user to enter a command that it then executes. Commands are either built into COMMAND programs (.COM and .EXE files) or batch files. COMMAND.COM can actually be replaced and there are a number of products that provide either an extension to COMMAND.COM or provide a menu interface. However, none of these programs has even made a dent in the number of systems using COMMAND.COM. The only one that comes close is Microsoft Windows, which does so for different reasons.

(Continued on page 60)

William Wong is president of Logic Fusion, Inc., a systems software development firm located in Morrisville, Pennsylvania.

UNIX on the PC

A comparative look at SCO Xenix, Microport System V/386, and Interactive Systems 386/ix — three UNIX alternatives for the PC.

by Bob Morein

"As with most sophisticated computer systems, panics will occasionally occur; they should not cause much concern,"

The UNIX System Administrator's Guide

With those words of reassurance, let me add that the readers of this review will probably be divided into two camps: those who credit me with divine infallibility and those who don't believe a word I say. To the first group I say, "No one is infallible," and to the second group I say, "I tried." Documentation for a UNIX system is variously 5000 to 10,000 pages, while typical binary distributions range in size between 15 and 30 megabytes, although those considered here are under 20 MB. For the next thousand years, scholars will argue about what constitutes a UNIX system and what is SVID (System V Interface Definition) compatibility. Consider the following myth: "The UNIX porting base provided by AT&T is truth, beauty, and perfection, and the porting vendors muck it up and make it buggy."

In fact, the porting base is full of bugs, which are only gradually being discovered by AT&T (they are human beings, after all!). It is the job of the vendors to fix the bugs of their respective ports, something that consumes a major portion of their time. For this reason we have not considered bargain-basement ports, which are supplied as fixed releases, but only those that are supported and improved by a vendor. These include the Santa Cruz Operation (SCO), Microport, and Interactive. The hardware used was a pair of essentially identical AT-compatible 10-MHz machines, the Intel Inboard 386 with a math coprocessor, and a total of 4 MB of memory (1 MB of 16-bit, 1 AT bus wait-state and 3 MB of 32-bit, 1 wait-state). A 386 UNIX running on this platform provides performance equivalent to the Sun 3/60, which is a 20-MHz 68020 machine that runs the vaunted FFS (Berkeley Fast File System).

Until recently there has been no universal benchmark for file system performance, but Arvin Park of Princeton University has now provided that in the form of the "iostone," which will join the whetstone and dhrystone in the pantheon of benchmarks. It's interesting to note that, with a standard Western Digital ST506-type controller (WD1002-WA2) and Micropolis 1325A 70-MB disk drives, two of the ports were able to exceed the file system performance of the Sun and come very close to that of the VAX 8600. With a more modern controller, of which there are several, performance might move beyond the VAX and the deluxe Sun/260 into the area previously reserved for mainframes. There is no limit to the amount of possible disk storage since National Memory Systems (not to be confused with National of Japan) has an SMD (storage module device) controller with drivers for all of these systems that has a data transfer rate of 3 MB per second. It can handle up to 4 SMD drives, with a practical maximum of 1.44 gigabytes per drive. The only limit on the number of controllers is power and slot space.

I'm appalled at the popularity of bad benchmarks. But what makes a good benchmark? Ideally the number should strongly correlate with the performance of a class of applications. Inventors tend to neglect this fact, preferring to proliferate tables that compare the speed of some isolated function in the kernel. Readers are left with a pile of figures, and must conduct their own "benchmark of benchmarks" to determine which ones, or what combination with what weighting, corresponds to their applications.

I believe that the dhrystone adequately represents CPU-bound per-

Bob Morein is founder of Automata Design Associates, specializing in Prolog language systems for DOS and UNIX. He holds a BS in engineering physics from Lehigh University, an MS in physics from Temple University, and is working for a doctorate in electrical engineering at Drexel University. In addition to UNIX, he is interested in neural networks and digital design.


DOS
386
UNIX
DOS
DOS

formance for non–floating-point applications; the whetstone gives math speed; and the iostone satisfactorily imitates the disk activity of the average installation. The iostone was compiled by statistical analysis of disk activity at several sites, just as the dhrystone is the result of statistical analysis of program structure. We don't want language-independent benchmarks, since virtually all UNIX applications are written in C, and the compiler is as important as the processor in predicting system speed.

If you will examine the charts, the surprising conclusion of this article is that the AT hardware is good for a lot more than critics say, and when a 386 chip is used, you have a VAX-killer capable of replacing the VAX 8600 series in departmental computing. Further, with the UNIX System V, Release 3, innovation of RFS (remote file system) machines in a network can transparently mount and use file systems in any other machine also running RFS. So a distributed mainframe could be constructed that would replace the heaviest IBM hardware.

All of these systems are roughly equivalent in function. It is practically impossible to compare every feature since there are hundreds of utilities, a dozen or more languages, and thousands of functions, and any one of them taken in isolation is not terribly important. But every release of UNIX contains major architectural features. Here readers may have to make hard choices. For example, the STREAMS facility is a significant improvement in the use of character I/O for interprocess communication. Applications such as X-Windows exist that require STREAMS or Berkeley sockets to run, and there is no satisfactory substitute. In practice, the reader may conclude that the reliability and performance of these systems has greater significance than the superiority of any particular feature.

SCO Xenix

There are about 400,000 UNIX systems in the world to date. Of these, 200,000 use Xenix, and the rest use BSD, ULTRIX, System V, and other things. SCO has sold 50,000 Xenix licenses, and is currently shipping 2500 systems a month. Xenix-286 has been one of the mainstays of small-office systems, and the 386 version can do more. All SCO Xenix licenses are currently of the unlimited user type; other UNIX systems typically provide a two-user login limit with unlimited privileges available at extra cost. The system is robust and insensitive to hardware variations. This is important because we have only a vague notion of what "386-AT" means, since there is no corresponding IBM model. Each machine has subtly different hardware characteristics which become known in an unsubtle way —you try it, for example, and it doesn't work. But SCO rises above this for the most part, except for disk controller problems. (They tell me the Western Digital WAH doesn't work, but the WA2 does. Why?) RLL and ESDI are also supported, and a custom disk formatter handles drives not in the AT ROM table. The DTC RLL controller is recommended, since that's what SCO uses in-house; it gives an inexpensive boost if you cheat with high-quality, non-RLL drives.

The documentation is a marvel. They are the best books written on UNIX anywhere —totally unpretentious (no new-speak!) and extremely complete, with two chapters on writing device drivers. Eight binders contain the Imagen laser printout, which you could do yourself with the text-processing system. Read it, try it, and read some more. Then refer to the manual pages, which are supplied online (a very important feature for a school or remote use). There is complete conformance to the software, since the documentation is done by the same people. And they never stop! In the space of six months I've seen significant expansion and revision.

There is complete downward compatibility, to the extent that much of the software has been left as 286 code, but it's fast. The Microsoft C compiler has a 2:1 speed advantage over the Portable C Compiler, and the code is much smaller. The object file format is not compatible with AT&T 5.3, but the next release will have COFF (common object file format) as a first step toward the 386-merged UNIX that will embrace and supersede the current products. Operations are extremely well organized, and the degree of technical support provided may not be equalled by any other company in the software industry.

Chief among concerns is to what extent Xenix *is* UNIX. It would be foolish to argue that it is not, since the kernel passes SVID. The utilities do not pass, however, which means there are operational differences—all to the better. More importantly, SVID conformance does not require that all components of 5.3 be implemented, and they are not. Networking is possible, but it is heterogeneous, with special utilities for file transfer and logging into remote systems. The STREAMS feature is not available as of this writing, but a form is scheduled for Release 2.3. It will permit local applications to run, but RFS will not appear until the 3.2 edition of the merge product.

Multiview, available at extra cost, is a very interesting character-based windowing system, shell, and extension to the operating system. It can project onto every compatible terminal in the system up to six windows of adjustable size.

Microport System V/386

The Microport System V/386 is the most complete UNIX distribution. As with Interactive's 386/IX, it is a direct product of the UNIX System 5.3 porting base, and is almost completely documented by the AT&T UNIX manuals. What distinguishes one port from another is the quality of the basic drivers, which are roughly equivalent to the system BIOS, the completeness of the distribution, and features or utilities beyond those supplied with the porting base. Every component of V.3 is available from Microport, including the Network Extensions which comprise RFS and STREAMS. The price of the complete 5.3 documentation from AT&T is $780, but you receive most of that, and a two-user "limited" version of 5.3 as well, for a typical retail price of just $700. An unlimited user version is available at considerably greater cost. The supplied documentation includes:

- *Product Overview*
- *User's Guide*
- *User's Reference Manual*
- *System Administrator's Reference Manual*
- *Programmer's Guide*
- *Programmer's Reference Manual*
- *Documenter's Workbench Software User's Guide*
- *Documenter's Workbench Software Technical Discussion and Reference Manual*
- *Documenter's Workbench Software Handbook*
- *Documenter's Workbench Software Handbook for New Users*

It's a pity they left out the Administrator's Guide, but you can obtain it, as well as the rest of the manuals, from AT&T, or better yet, you can get the excellent Prentice-Hall edition from bookstores. And if you wish to complete your education, consider adding:

- *Network Programmer's Guide*
- *STREAMS Primer*
- *STREAMS Programmer's Guide*
- *Utilities Release Notes* (they charge you for the bug reports)

The binders are handsome and will complement your library. It's a shame

that divider tabs are not supplied.

For UNIX aficionados there are irresistible attractions:

- The Korn Shell, a much-improved shell resembling a structured programming language, replete with array data types.
- The Green Hills "C" Compiler, known as a benchmark champion. If you must produce highly optimized COFF output, there is no other way at present. It would have done even better in the dhrystone, but the "-O2" switch was apparently not implemented.

It is well known that Microport is working hard on implementations of X-Windows and GNU. There is only very limited downward compatibility with System V/286, since loading of Intel .OMF (object module format) files is not supported.

Alas, Microport suffers from the travails of youth. Operations are iffy, so while shipment of new orders is fairly prompt, purchasers of support contracts frequently face delays of several months in receiving updates. Release dates tend to be so unrealistic as to be deliberately exaggerated. Support lines are frequently tied up, with promised callbacks often never being made or scheduled. A strong plus is the company's new bulletin-board system policy which allows any registered Microport user to download bug fixes without charge, even in the absense of a service contract.

The system is quite touchy with respect to hardware, but it is likely that by the time you read this article there will be significant improvements in fault recovery and hardware flexibility.

Interactive Systems 386/ix

The 386/ix operating system was tossed softly into my lap by UPS. Usually you don't toss UNIX documentation around. What was it, a credit memo? I eagerly ripped open the carton. Within were three small and shiny printed boxes obviously designed for retail display. The box of disks could be inverted and propped up for display, like the packaging for a Timex watch. Three small D-ring binders emerged. A faint smile appeared on my lips. "At last," I thought, "somebody's read the documentation and isolated the important stuff." As I flipped through the binders and saw only end-user documentation, another pleasant thought flitted past. "It's on disk! It's on disk?" The truth emerged when I read "Roadmap to the Documentation," which tells you where to buy the complete documentation package.

I had not expected Interactive, which did the original 386 port, to embark on a brave experiment in product packaging, but that was precisely what the company did. Simple mathematics revealed that, at some point, there would be more UNIX users than developers. And who would sell them licenses? Interactive would, by repackaging UNIX in a new menu-driven shell (with new terminology), a new editor, and a strictly hierarchial, *a la carte* approach to documentation that would reduce delivery costs and user confusion. The standard distribution has the two-login limitation, with an unlimited user version available at extra cost.

In a sense, the Interactive product defines System 5.3 on the 386, and the AT&T documentation is directly and completely applicable. The importance of this should not be exaggerated; the result of the porting contract was quite bare and undebugged, and has been subject to elaboration by AT&T, Microport, and Interactive as well.

So if you're a developer, you separately purchase the excellent Prentice-Hall edition (ask for the 386 edition, not 3B2) of the System V documentation, and consider purchasing additional port-specific documentation from Interactive as well. Since the software and the hardware have been through many different hands, the

Table 1. A comparative look at UNIX performance on the 386

Dhrystone Benchmark Results

SCO/ Microsoft	Microport/ PCC	Microport/ GCC	Interactive
6020	2704	4147	2649

Whetstone Benchmark Results

single	double	single	double	single	double	single	double
549	555	729	819	1234	1388	724	826

Note: Double precision really was faster.

Comparison Figures

Manufacturer/Model	Dhrystones	Whetstones
Sun 3/60 (20 MHz 68020)	4273	
Sun 3/280 (25 MHz 68020, cache)	6329	
DEC VAX 11/785	2090	
DEC VAX 8600	3866	
IBM PC/AT, large model	696	approx 200

Iostone Benchmark

Vendor	Interleave	Disk Buffers	Iostones
SCO	2:1	537	2072
SCO	2:1	1000	2259
Microport	2:1	1100	1049
Microport	3:1	1100	1156
Interactive	2:1	600	1990

Comparison Figures

Sun 3/60	local disk, 8 MB of RAM	1834
Sun 3/60	Ethernet to Sun 3/280 server	1176
Sun 3/280	local SMD disk–Fuji Eagle	3669
Sun 3/280	Ethernet to Sun 3/60	1133

Espresso Compilation

Vendor	Interleave	Disk buffers	C&L	L	cat
SCO	2:1	537	4:04	5.1	27.5
Microport	3:1	1100	5:02	–	1:37.4
Interactive	2:1	600	4:34	4.3	1:36.4

Comparison Figures

Sun 3/60	8 MB of RAM, 800K buffers	4:04	–	–
Sun 3/280		3:54	–	–
Apollo			–	–
DN3000		8:27	–	–

C = Compile
L = Link
cat = cat source files to screen

Load Degradation—Espresso Compilation

Number of compilations running concurrently

	1	2	3	4
Vendor	*Real time per compilation*			
SCO	4:10	7:46	11:22	15:31
Microport	5:14	11:13	15:58	22:44
Interactive	4:35	8:56	12:42	17:24

tight integration found in the SCO documentation is absent. Although the AT&T documentation is markedly better now than in the past, the coverage is still uneven, with an impractical bent and a tendency toward supernatural jargon: "Insert the *medium* in the disk drive." The Interactive documentation is expensive but well written, and in fact, programming with the company's window-like user interface cannot be done without it.

Interactive has promised that purchasers of the software development system will receive the *Programmer's Guide* and the *Programmer's Reference Manual*, and a very nice small book they have authored about writing device drivers. It's probably the last word, since they ported the kernel, and in conjunction with *Writing a UNIX Device Driver* (Janet I. Elan and Thomas J. Teixeira, Wiley, 1988), it puts the lie to the notion that you have to have the kernel source to know what you're doing.

The port acquitted itself quite well in performance, although there were some embarrassing bugs and some hardware sensitivity. Also, perhaps it did not like having an enlarged process table. It seems that nobody knows how to format a disk these days. The surface analysis program repeatedly died when confronted with bad sectors, and the *adddisk* program terminated without warning or success when a second hard drive was installed. Sometimes a low-level format of the entire disk would fix this, but toward the end of my repeated trials, it failed entirely. Interestingly, Microport admits that its surface analysis is buggy, but in fact it did reasonably well. Only SCO can edit the bad track table, handle nonstandard disks, find bad sectors with any reasonable certainty, and recover data from bad sectors in an existing file system. And both Microport and Interactive insisted on destroying the entire contents of the disk, rather than remaining within their own partitions. Microport and Interactive map defects on the sector level, but allow only 62 bad sectors per drive. Since there can be as many as 17 bad sectors on a single track, this scheme is only marginally capable of handling large hard disks.

Retail is new to Interactive. It was clear when I called that the switchboard operator did not know what technical support was because I was repeatedly connected to someone in janitorial, who didn't know either. It's possible I was the first caller; eventually the call got through to some hapless individual sitting in a dark room with his feet in a bucket of ice water.

Simply the BEST C and Pascal on AT, 386, Sun, Apollo, RT, VAX, 370

"The most rock-solid C compiler in the industry. Superb technical support and portability. Superior code generated."
Gordon Eubanks, Symantec — Q&A (386).

"It simply works, with no trouble, no chasing strange bugs, and excellent warning and error messages ... a professional product."
Robert Lerche, Bay Partners.

"For large-scale software development, the highest quality C compiler available on the market today. Pragmas are great. Quality of support is exceptional." **Randy Neilsen, Ansa—Paradox** (DOS,OS/2).

"15% smaller and 15% faster than Lattice C."
Robert Wenig, Autodesk.

"Our software is running anywhere from 30 to 50% faster than when compiled under Lattice." **David Marcus, Micronetics.**

"We switched from Lattice due to a 10% reduction in code size. The compiler is very stable." **Lee Lorenzen, Ventura Software — Ventura Publisher, marketted by Xerox Corp.**

"Best quality emitted code by any compiler I've encountered. Often amazing." **Bill Ferguson, Fox Software — FoxBase (386).**

"Messages sometimes pointed out type mismatches, incorrect-length argument lists, and uninitialized variables that had been undetected for years [in 4.x bsd]." **Larry Breed, IBM ACIS [RT PC].**

"Diagnostics turned up bugs missed by other compilers. Rapid bug fixes by technical support, someone who knew what he was talking about. 80386 code is well optimized."
Tim Addison, Logistics Data Systems.

"386 protected mode support is fantastic, especially the access to large amounts of memory. It's mainframe compute power on a PC." **Dan Eggleston, Viewlogic.**

"The preprocessor supplied with Professional Pascal is quite useful. The code quality and control over segmentation and memory models are superior to MS Pascal." **Bob Wallace, QuickSoft.**

Check Out These Reviews

- **High C ™:**

Computer Language	February 1986, '87	
Dr. Dobb's Journal	August 1986	
PC Magazine	Jan. 27, 1987	(80386 version)
Dr. Dobb's Journal	July 1987	(80386 version)
BYTE Magazine	November 1987	(80386 version)

- **Professional Pascal ™:**

PC Magazine	Dec. 29, 1985	
Computer Language	May 1986	
PC Tech Journal	July 1986	
Journal of Pascal, Ada, & Modula-2	Nov.-Dec. 1986	
BYTE Magazine	Dec. '86, June '87	(80386 version)

Why MetaWare compilers

- They are specifically designed for serious software developers.
- They are reliable and robust: they don't break at every turn.
- Their generated code is the best, or near best, on each architecture.
- Their superior diagnostic messages help you produce better products more quickly.
- Your source can be ported with ease to the most popular systems.
- You can link mixed-language modules from our compilers, others
- You can benefit from high-level, personal technical support.
- You can take advantage of the latest ANSI C extensions, and/or extensive Pascal extensions. **High C** has been tracking the ANSI Standard for two years; **Professional Pascal** will soon have a VS Pascal compatibility switch and several Apollo Pascal ext'ns.
- You can take advantage of the latest 387 and Weitek 1167 support — we have the only compilers with Weitek real mode support.

Power Tools for Power Users

Ashton-Tate: dBase III Plus, MultiMate; Autodesk: AUTOCAD, AUTOSKETCH (8087, '387, Weitek); Boeing Computer Services (Sun); CASE Technology (Sun); CAD/CAM giant Daisy ('86, '386, VAX); Deloitte Haskins & Sells; Digital Research: FlexOS; GE; IBM: 4.3/RT, 4680 OS; Lifetree Software (Pascal): Volkswriter Deluxe, GEM-Write; Lugaru: Epsilon; NYU: Ada-Ed cmplr; Semantec: Q&A; Sky Computers; ... (Product names are trademarks of the companies indicated.)

Industrial-Strength

MetaWare C and Pascal compilers are designed for professional software developers. These tools are loaded with options to control them for special purposes. You can adjust the space-time trade-off in code quality. You can adjust external naming conventions to agree with linkers and operating systems. You can specify segment names for segmented architectures, and to help place code or data in particular places for embedded applications. You can select from five memory models for the 8086 family. And on and on.

A Partial List of Optimizations

Common subexpression and dead-code elimination, retention and reuse of register contents, jump-instruction size minimization, tail merging (cross jumping), constant folding, short-circuit evaluation of Boolean expressions, strength reductions, fast procedure calls, automatic mapping of variables to registers (where advantageous), ...

"Platform" — Code Quality

Sun,Apollo,SGI — 18%, 3%, 26% > resident compiler (Dhrystone).
PC: DOS, OS/2 — 3-10% > Microsoft C; 30% > MS Pascal, LatticeC.
386 32-bit DOS— no competitors, since November, 1986.
286, 386 UNIX — 66% better than pcc (Dhrystone, 386).
VAX VMS — ≈ DEC's excellent C and Pascal; better features.
VAX Ultrix — 19% > pcc (Dhrystone); much > Berkeley Pascal.
RT PC/4.3bsd — 89% > the original port of pcc (Dhrystone).
370 CMS,UNIX— much better than any C, and VS Pascal.
AMD 29000 — >40,000 Dhrystones! Available in Q2, cross.

(408) 429-6382, telex 493-0879. ***Since 1979.***

MetaWare™
INCORPORATED

903 Pacific Avenue, Suite 201 • Santa Cruz, CA 95060-4429

The Clear Choice for Large Programming Projects — PC Tech J.

CIRCLE 55 ON READER SERVICE CARD

(I did receive a nice callback later.) The people at Microport are more convivial, since apparently the phone system there causes every curious person to pick up the call, see if it pertains to him, and if not, transfer it randomly to some other person—until the switchboard operator pulls the plug.

The Benchmarks Revealed

The dhrystone (Table 1) was originally coded in Ada by Rheinhold Weicker and converted to C by Rick Richardson. In each loop, 99 statements are executed: 52 assignments, 32 control, and 15 procedure calls. Version 1.1 was used and compiled with all available compiler optimizations. The result is given in number of dhrystones, or loops, per second, so the higher the better. Floating-point arithmetic is not used, so it is rather amusing to see reviewers quoting results with and without math coprocessors. When the same hardware is used to compare different compilers it becomes principally a test of code quality, since the kernel overhead is small, even in the worst of ports. Running multiple simultaneous dhrystones gave rise to an interesting observation: The degradation per process was almost exactly the inverse of the number of processes; the whole *was* approximately equal to the sum of the parts.

The iostone is described as "a synthetic file system performance benchmark." The authors discovered apparently universal laws, such as the approximate 2:1 ratio of reads to writes over many installations, and analyzed the size and frequency of blocks transferred. The iostone performs neither strictly sequential nor random access but models a continuum, from very short blocks to 64K in a single transfer, all according to the empirical data. All operations occur within a single, temporary file. Although I have not tabulated performance of concurrent iostones (owing to the difficulty of getting sufficient disk space), degradation was severe.

The whetstone measures the quality of the particular code for the 80387. Some systems, notably SCO Xenix, suffer because they generate 80287-compatible code. The 80386 coprocessor bug, not to be confused with the 32-bit multiply bug, tends to cause the sudden death of the operating system; this is a fix the OS people would prefer to leave to hardware, although there is, in theory, a way around it using software. In fact, SCO Xenix Version 2.3 has addressed this coprocessor bug with a workaround option at bootup. This solution, however, adversely affects coprocessor performance and does not work with all hardware.

The *espresso* benchmark is the most interesting of all, at least to the programmer. It measures the time required to compile and execute 200K of C source code for a public domain program developed at the University of California at Berkeley, which designs two-level programmable logic arrays from truth table specifications. This activity provides an ideal mix sequential read (file loading), compiling (compute-bound activity), linking (random access), and sequential write (writing the object file). Examination of the multiprocess results indicate that the overall system efficiency actually increased with the number of processes running, indicating that the UNIX ports for the 386 manage resources efficiently compared to their predecessors.

The AT hardware is good for a lot more than critics say, and when a 386 chip is used, you have a VAX-killer, capable of replacing the VAX 8600 series.

Another test is the "cat to the screen" of all the source files, which gives some notion of screen bandwidth. The composite results explode the notion that a Sun is the requisite tool for heavy programming. If it weren't for the small, windowless screen, the AT-386 type would be the workstation equal of the Sun, and this disadvantage is likely to be removed before the end of the year.

Reliability

Because the 386-AT is the invention of many different engineering minds, the problem of hardware compatibility rears its ugly head. The three operating systems split extremely fine hairs in all departments. You will discover this in all it's glorious absurdity when you read that Microport DOS-merge does not work with MAXTOR drives, or that EGA boards, which function perfectly under DOS cannot be made to work with SCO CGI (discussed later). To this we must add a CPU chip for which the specifications change frequently! Intel publishes periodic "revision of specification" documents and has actually deleted some instructions. The coprocessor bug caused by a design flaw in the 386 chip is universally present and generates a small probability of a system crash per coprocessor instruction executed. A new mask, not yet in production, is one of the possible solutions to this position.

Reliability problems in hardware that contained the 80387 coprocessor were encountered with two of the operating systems, while identical hardware minus the coprocessor worked flawlessly. The system crashes occurred during the compilation of C, which does not use floating-point, and would not be expected to invoke the 80387, although it's possible that the long integer arithmetic involved in process accounting does. But the syndrome exactly matched a typical coprocessor crash: A job failed to terminate, and an attempt to start another job or kill the hung process brought down the system. This is not proof that the coprocessor was the culprit, but the system with the coprocessor was put under a heavy load running Xenix for 40 hours without mishap.

Users who have the "right" hardware report that all of these operating systems operate reliably, at least for their activities, while others complain endlessly via USENET. Had I reported perfect operation of all the systems, I would have been reporting a misleading, fortuitous result. A disk containing the benchmarks used in this article is available so that the reader can attempt to validate a particular software/hardware choice. But be forewarned that it does not address the ability of a system to handle serial input/output. USENET reports indicate that there is a substantial difference in the ability of these systems to handle high-speed serial input.

Innovations and Directions

In times gone by, a device-independent, character-based interface for cursor-positioning terminals was one of the great achievements of UNIX. But in a bit-mapped world, UNIX lovers must grin and bear it when they are told that simple DOS provides a better CAD platform. The screen in graphics mode is mapped to one application unless, of course, there is a windowing system. None of the systems considered offer a windowing system at

present, but this may change shortly. What is alarming, however, is that the range of graphics devices supported is quite narrow compared to that supported by DOS. SCO offers a precursor to windowing called CGI, which is an implementation of the ANSI computer graphics interface. It offers the customary graphics primitives and font generation facilities—in one window—and the fanciest adaptor supported is standard EGA. The next step would be X-Windows, a worthy public-domain windowing system that would have become the UNIX standard were it not for the Sun/AT&T romance, which culminated in an exchange of vows and 20 percent of Sun's stock. This means that we will eventually receive, as part of System 5.4, the extremely complicated system NEWS, or Network Extensible Windowing System, which purports to communicate via Postscript over networks. Early users at Sun reported that it consumed all of your RAM and most of your patience.

The last gasp of centralized processing has just appeared in the form of the Sun River fiber-optic workstations. Priced at little more than the cost of a complete 386 machine, each station is a bit-mapped EGA or Hercules terminal connected by a 30-megabit (Mb) fiber-optic link to a single 386 machine, which must refresh up to four bit-mapped workstations simultaneously. This must stand as the most significant development since the discovery of n–p complete algorithms. More logical would be the marketing of diskless 386 machines, as is standard practice in the workstation market.

We may conclude that the future of UNIX on the 386-AT class machine is bright. Unfortunately, I cannot determine whether the brightness is one of cheer or that of a display stuck in reverse video. □

Did you find this article particularly useful? Circle number 4 on the reader service card.

Product Information

AT&T
P.O. Box 19901
Indianapolis, IN 46219
(800) 432-6600
Circle reader service #252.

Interactive Systems
2401 Colorado Ave., Third Floor
Santa Monica, CA 90404
(213) 453-8649
Circle reader service #253.

Microport Systems
10 Victor Square
Scotts Valley, CA 95066-3518
(800) 438-8649
Circle reader service #254.

National Memory Systems
355 Earhart Way
Livermore, CA 94550
(415) 443-1669
Circle reader service #255.

Prentice-Hall
Sylvan Ave.
Englewood Cliffs, NJ 079632
(201) 592-2498
Circle reader service #256.

Santa Cruz Operations
400 Encinal St.
Santa Cruz, CA 95061
(800) 626-4381
(408) 425-7222
Circle reader service #257.

The benchmark diskette may be purchased for $20 postpaid from:

Automata Design Associates
1570 Arran Way
Dresher, PA. 19025
(215) 646-4894

Running DOS Under UNIX

You no longer have to abandon your favorite DOS applications to tap the power of UNIX.

by Phil Hughes

Over the past five years, the number of UNIX applications has been growing, so that today you can find almost any program you want. At the same time, however, people have become increasingly involved in the world of MS-DOS. Today, a person is more likely to be able to use Lotus 1-2-3 or Wordstar than drive a stick-shift car. So, telling people to scrap their 1-2-3 templates and dBASE programs is not the way to get them to accept UNIX.

The ultimate solution is to offer all of UNIX's advantages (a large base of utility programs, automatic communications packages, support of more powerful machines, and a multitasking, multiuser environment) to the DOS user; That is, allow users to keep running their DOS programs, but give them access to UNIX when they need it. And, that is exactly what this article is about: products that allow a UNIX system to support DOS users.

The good news is there are such products and they do even more than I expected. And the bad news isn't that bad. In fact, the only real bad news is that, as I write this article these products have just hit the street and they have kinks that have to be worked out. By the time you read this, I expect that most of the problems will be a thing of the past.

Phil Hughes is Vice-President of Specialized Systems Consultants Inc. (SSC) located in Seattle, Washington. SSC markets pocket reference booklets for various UNIX systems and utilities, and software, and offers training and consulting to UNIX users.

The Concept

The concept of running DOS as a UNIX task is not particularly difficult to grasp or achieve. If you look at it from the UNIX perspective, DOS is just another program to run within the operating system. You schedule it when it needs processor time, handle I/O for it, and wait for it to terminate. The problem is that DOS expects to have complete control of the hardware—it expects to handle interrupts, access the disk drives, and hang in a loop waiting for a keystroke from the keyboard. It also assumes that your keyboard generates *scancodes* rather than standard ASCII characters. Scancodes are sent to the CPU for all keys, including depressing a Shift or Control key. DOS also expects to be loaded into the first 640K of RAM.

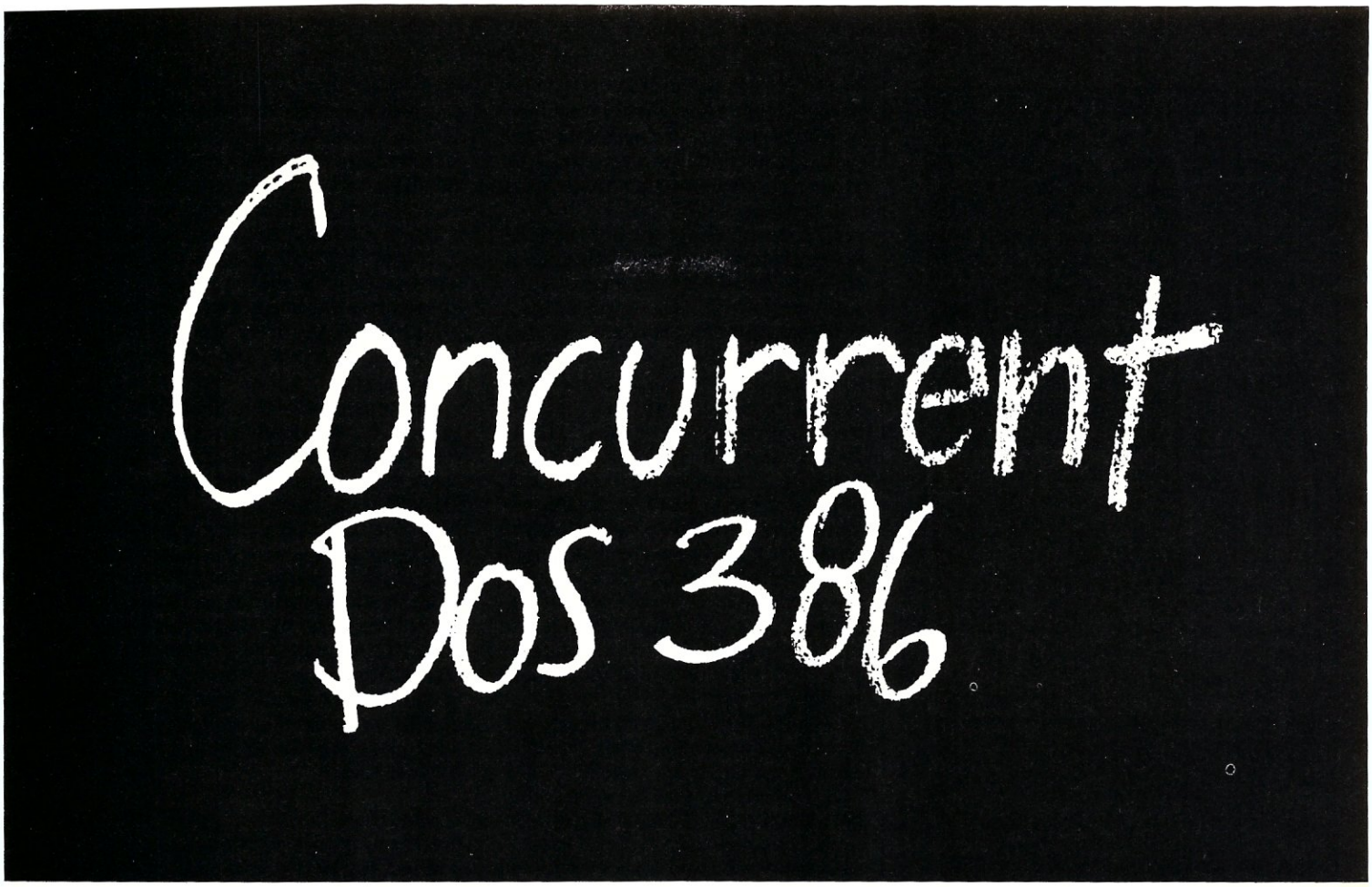
Most of these problems can be solved by making changes to the device drivers for DOS; make them talk to UNIX, which in turn will talk to the actual hardware. This adds software overhead but it works. Use of resources such as the floppy disks remains a problem because only one person at a time can use them. Software locks can be used to prevent multiple users from accessing them. Finally, loading DOS into the first 640K of RAM is not a problem on 80386-based machines. They have the capability to support virtual machines, where each user only sees their part of the machine. On 286-based machines the only practical solution is to allow a DOS user to occupy the beginning of RAM. This means that only one DOS user can be supported (along with multiple UNIX users) on 286-based machines.

The Products

Two distinct products can be combined with different hardware and UNIX versions: Merge 386 (or 286) and VP/ix. (Except where a distinction is necessary, I will refer to both Merge 386 and Merge 286 as just Merge.) Merge was developed by Locus Computing Corporation. VP/ix was developed by Interactive Systems Corporation and Phoenix Technologies Ltd. The other half of each combination consists of whatever flavor of UNIX is attached. We have four different UNIX flavors with two significantly different parentages to deal with. The hardest one to explain is Xenix. Xenix was developed by Microsoft Corporation from AT&T's UNIX Version 7. It contains enhancements to make it a more commercially viable product. Over the years, Xenix has been updated to comply with AT&T System III and now AT&T System V, Release 2, specifications. The Santa Cruz Operation (SCO) has developed generic ports of Xenix for XT, AT, and 386 clones, and markets this package to end users. The result is that Xenix looks like AT&T System V, but when you start attempting to integrate support of DOS into UNIX it is clearly different.

The remaining three flavors are Interactive Systems 386/ix, Microport V/AT, and Microport V/386. All three of these are directly descended from the AT&T System V release of UNIX. Both 386/ix and V/386 are based on UNIX System V, Release 3, and run on 386-based systems. V/AT is based on UNIX System V, Release 2, and runs on 286-based machines.

Now, let's get the DOS-related products paired with the UNIX flavors.



Think small in a big way

When you think multiuser/multitasking, think Concurrent™ DOS 386, the big name in small systems from Digital Research,® architects of the first standard operating system for personal computers. Now, Concurrent DOS 386 allows multiple users to share peripherals, files and applications, using serial terminal workstations linked by RS-232 cables to the system. It's fast, reliable and economical.

The big news today is small systems

Concurrent DOS 386 meets the increasing demands placed on small systems by supporting multiple DOS programs on both the system console and attached terminals. You can run popular programs such as Lotus® 1-2-3,® dBase® III, WordPerfect® and many more, with full math coprocessor support. The system runs up to 255 tasks simultaneously, with full intertask communications and byte-level record, file and device locking.

For people who hate waiting in line

Concurrent DOS 386 brings you all the remarkable speed and power of the Intel® 80386 processor. A prioritized preemptive scheduler allows task execution and intertask communication by several users at near full processor speed while letting some tasks "interrupt" others according to the needs of each user.

A small system with a big memory

Concurrent DOS 386 gives you access to four gigabytes of linear physical memory. Its powerful memory paging capability fully supports the Expanded Memory Specification with no additional hardware or software.

Menus at a touch

Now you can create and customize menus, while programmable function keys let you condense complex commands to a single keystroke. The file manager runs standard operating system functions, plus you have an on-line help facility, text editor and support for DOS-based device drivers.

Multiuser color graphics

Now with the introduction of the newest member of the Concurrent DOS family, Concurrent DOS 386/Multiuser Graphics Edition, your demands for high-resolution EGA bit-mapped graphics in the workstation environment can be met. Take advantage of advanced technology allowing you to run popular DOS-based graphics programs on individual workstations as well as on the system console without sacrificing system performance. Ask us about this exciting new version of Concurrent DOS 386.

All you have to remember is Concurrent DOS 386

Concurrent DOS 386 from Digital Research is the name to remember when it comes to 386 technology. The power and versatility of Concurrent DOS 386 are giving a new meaning to the word multiuser.

CALL DIGITAL RESEARCH AT 1-800-443-4200 AND ASK FOR OUR CONCURRENT DOS PROGRAMMER INFORMATION KIT.

CONCURRENT DOS 386: SHARING THE SYSTEM AFFORDABLY

DIGITAL RESEARCH®

CIRCLE 98 ON READER SERVICE CARD

Merge runs with the Microport UNIX kernel. Thus, Merge 386 is available with Microport V/386 and Merge 286 is available with Microport V/AT. These products are available from either Microport or Locus. VP/ix is available with either the 386/ix UNIX kernel from Interactive Systems or with the Xenix kernel from SCO.

System Requirements

Each product has virtually the same system requirements. The 386-based products require an Intel 80386-based computer, at least 2 MB of RAM, at least a 20-MB hard disk, and some kind of video board: EGA, CGA, or monochrome. (For Merge 286, substitute an Intel 80286 processor in the requirements.) These are clearly minimums. Most of the 20-MB disk space is used up by UNIX, so add the amount of disk space you actually plan to use for your application. For serious use, 4 MB or more of RAM seems more realistic.

I ran the 386 tests on a MicroSmart, 16-MHz, 386 system with 1 MB of 32-bit RAM and 2 MB of 16-bit RAM. The system has a 64K cache buffer between the 16-bit RAM and the processor, so actual access speed is close to what you could expect with all 32-bit RAM. The system also has a monochrome video card, parallel port, two dumb serial ports, an 8-line Arnet SmartPort board, and AMI BIOS. Disks used were either an Atasi 3046 (36 MB, 33-ms access) or Seagate 4096 (80 MB, 28-ms access).

The 286 tests were run on a FiveStar 8-MHz FS-286 system with 2 MB of 16-bit RAM, a monochrome video card, a parallel port and two dumb serial ports, and TriGem BIOS. The disk was an Atasi 3046.

A word of warning on BIOS compatibility. Merge 286 is "fully supported" on an IBM PC/AT with IBM BIOS and TeleVideo Telecat 286 with TeleVideo BIOS. Although I had no problem with the TriGem BIOS, I suggest you check with Microport before you buy Merge 286 if you have unconventional BIOS. Compatability is less of a problem on the 386-based system.

Installation

Installation of each package was pretty much the same. Basically there are three steps:

1. Install UNIX.
2. Install the DOS product.
3. Configure DOS users.

All of the 386 packages come with DOS. Merge 286 does not, so you need to make a copy of your existing DOS system disk (Version 3.0 or later) and install a modified image of DOS. This means the 286 installation takes a little more work than the 386, but, again, there's not a significant difference. My biggest problem was the confusion I faced because I was installing all different packages, and each one had to be installed in a slightly different manner. (It reminded me of the time I was learning C, writing programs in PL/M, and teaching a Pascal class.)

One final note on installation. Although I may make it sound easy, remember that I work for a UNIX consulting company and have done lots of UNIX installations. If you can install UNIX, you can easily install any of these DOS support products. Just remember, installing UNIX is much more complicated than installing a typical DOS system, which simply involves copying the floppy to the hard disk and rebooting. If you have never installed UNIX before, plan to spend at least a couple of hours for reading and a couple more hours to perform the installation.

Initial Use

Once you have everything installed, you can log on to UNIX and enter a command to get into the DOS environment. The command is DOS for the Merge products and *vpix* for VP/ix. Assuming you do this from the console, it will look as though you are now on a DOS system. Such commands as *dir A:* will do exactly as expected. To leave the DOS world from Merge, you just enter *quit*. A UNIX prompt returns. On VP/ix, depressing the Alt key and hitting SysReq gets you a menu. One choice is *quit*. Selecting it gets you back to UNIX.

Using the multiple screens feature of UNIX allows you to run more than one UNIX session from the console. This also works for DOS. Control sequences (although slightly different on the different products) allow you to switch between DOS and UNIX screens. The VP/ix menu also allows you to create a new UNIX shell without leaving DOS.

So far, so good. I would like to point out that, at this point, I discovered a few minor bugs in each of the systems. An example is that in SCO Xenix, while running GWBASIC under VP/ix, the interrupt (control-C) and stop output (control-S) characters are ignored. I am working with beta releases of these products and am sure that all the bugs I encountered at this time will be corrected (or at least replaced with new, fresh bugs) by the time you read this article.

One important deficiency I noted

at the time of this writing is the lack of support of intelligent communications boards. If you plan to use DOS at multiple terminals, verify that your hardware is supported before you purchase a particular UNIX/DOS product.

Terminal support is another consideration. The system keyboard generates scancodes for each key change. This includes a scancode when a key is depressed and another when a key is released. Also, screen capabilities of the console video board may be unique. The screen is 25 lines by 80 columns. Most terminals are 24 by 80. The video board may also support a graphics mode. Both TeleVideo and Wyse make scancode terminals (TeleVideo PCS1 and Wyse 60), but this may not solve the support problem if you plan on running an application that expects an enhanced graphics adapter.

File Access

Both products allow you to access floppy disks, a DOS partition on the hard disk, and the UNIX file system from DOS. Floppy disks and the DOS partition on the hard disk are accessed the same way as under DOS. Under VP/ix, the DOS partition can be shared by multiple DOS users only if it is read-only. Merge allows multiple users to have read/write access.

For most applications you will want to use the UNIX file system. Both products allow access to this file system from DOS in similar manners. Access is allowed by drive designator letters being mapped to the UNIX file system. All products allow multiple designators for the same file system; these allow you to move around more easily and satisfy the requirements of some applications programs. UNIX file access permissions determine the access privileges of these files.

DOS uses the key sequence of carriage-return then line-feed to designate the end of a line in a text file. UNIX uses only a line-feed. To allow for this, utility programs are provided to perform the conversions. On each system, you will run the appropriate utility to either add or remove carriage-return characters. Additionally, under Interactive Systems' VP/ix, there is a UNIX utility, *lef*, which looks at the file and decides which conversion to perform.

One nicety that was only in Interactive Systems VP/ix (actually, I expect it is part of 386/ix) was the way a missing floppy disk is handled. Most systems display an error message and abort the program or produce the DOS-like "Abort, Retry, or Ignore" message. VP/ix displays the message "diskette not present—please insert," then automatically keeps polling the floppy drive for the disk.

Installing DOS Applications

Once VP/ix or Merge is installed, you can permit UNIX users to access the DOS capability. You can also install DOS applications so they can be shared by all users. These configuration decisions are performed by equivalent programs in each environment: *DOSadmin* for Merge and *vpixadmin* under VP/ix.

In this area, Merge seems to have an advantage over VP/ix. Merge offers a menu program that allows you to install DOS applications and configure them. This includes specifying which CONFIG.SYS and AUTOEXEC.BAT files to use when an application is invoked.

Running DOS Applications

In order to see if DOS applications can really be run I tried the following:

- Lotus 1-2-3
- PC-Write
- Microsoft Word
- Alice–The Personal Pascal–demo
- Spybase
- Motorola Data Disk
- IBM AT Advanced Diagnostics
- OrCAD Demonstration Disk

I started the tests with PC-Write and Spybase on each system. They are both very screen-oriented and I am familiar with their operation. There

were no problems on any system from the console. They ran properly and at an apparently reasonable speed. Switching to a UNIX screen and back to their screens seemed to work fine.

The Motorola Data Disk is a database program and information on Motorola semiconductor products. It displays information in about six different languages and offers various search methods. I ran this in two ways: directly off the distribution floppy and by loading onto the UNIX partition of the hard disk. The only problem was that the screen print feature didn't work under VP/ix. This could have been caused by a misunderstanding on my part.

Microsoft Word was a different case. It ran under DOS directly and under Merge with no problems. Under Interactive Systems' VP/ix it worked fine, but if you changed to a UNIX screen and back to Word, the display became messed up. The only way out that I found was to kill the *vpix* process. On SCO VP/ix, as soon as you load Word the screen becomes a disaster. It is clearly a problem of initializing the display adapter, since the UNIX screens also end up sick. A reboot is the only way out. Neither I nor the person who brought over Word to give it a try are experts on its use, however.

Alice and OrCAD results were the opposite of each other. The Alice demo seemed to work perfectly with all systems, OrCAD worked with none of them, even though it worked correctly under DOS. The problem had to do with the video adapter.

. . . you can execute DOS commands from the UNIX shell and UNIX commands from the DOS prompt.

These results don't surprise me. The DOS programs were written to run under DOS on specific hardware. Each of them seems to have made some assumptions. At first, the fact that PC-Write worked surprised me. It is very screen intensive. Then I remembered that Bob Wallace, the author of PC-Write, used to work for Microsoft. I think his inside knowledge of MS-DOS has allowed him to create a portable product.

DOS/UNIX Interaction

There are three reasons you might want to run DOS under UNIX. The first is if you want to continue to use DOS but also want the advantages of a multiuser system. The second is if you primarily want to run UNIX but have an occasional need to run a DOS program. Third is if you plan to use both DOS and UNIX and want to communicate between the two—in other words, you want to get the best of each, all in one package. All of the products supported communication between DOS and UNIX much better than I had expected.

With some limitations, you can execute DOS commands from the UNIX shell and UNIX commands from the DOS prompt. This means that you can write UNIX shell scripts that include DOS commands, embed DOS commands in *make* files, connect com-

mands (UNIX and DOS) with UNIX pipes, and even use the UNIX *at* command to run DOS batch programs at a particular time. This powerful capability offers a great chance to merge the capabilities of DOS and UNIX, and facilitates an easy transition path from DOS to UNIX.

Documentation

I found the Merge documentation to be the best. This is probably because the product has been in the field for the longest time. It consists of about 20 pages of Release Notes, a 100-page Administrator's Manual, and a 200-page User's Manual. It is well done and accurate. All of this documentation is in addition to the UNIX documentation. Note that it doesn't matter whether you purchase the product from Locus or Microport. The documentation is the same, only the color of the binders is different.

The Interactive Systems VP/ix documentation seems to be well done and complete but I found it harder to use. This could be due to the fact that I worked with the Merge documentation first. There is less documentation on VP/ix than on Merge but what is there is well written. My major complaint is the lack of sufficient user documentation on UNIX itself. I expect this indicates that Interactive sees its market as "canned" multiuser DOS systems rather than UNIX running DOS.

SCO VP/ix documentation comes in third, mainly because it lacks an extensive user's guide. What is provided is well written, and the fact that I have a "Controlled Release" is the reason. I am sure the manual will be up to snuff with the Merge documentation when the full release is completed.

Technical Support

I used only the Microport and SCO technical support. I have had a lot of experience with SCO tech support and know many people who have worked with Microport tech support. The conclusions seem consistent.

SCO is the bigger company, and you will spend a lot of time talking to people who get your name and a one-line explanation of your problem, and then put you on hold or have you wait for a call back. Of course, once you do get assistance, they will be helpful and walk you through a problem. Just be sure to have a book to read or something else to do before you call. I give them a grade of B.

Microport support is spotty. When you call you may get a busy signal, someone who doesn't know what you are talking about, or an expert. You

will spend less time trying to find the answer but the answer you get could be excellent or relatively worthless. Recently this support seems to be getting better. I give them a B as well.

Okay, Which Should I Buy?

Which system do you choose? That's a tough one. If you already are running UNIX or Xenix on a 286- or 386-based machine, the answer is easy: Buy the one that works with what you have. If you are shopping for a complete package, I would let the selection of the operating system determine the decision. Picking the right operating system, in turn, depends on the intended use of the system. This decision could easily be debated in another article.

Both Merge and VP/ix work under Xenix and 386/ix. Performance-wise, Merge seems faster with such things as keyboard input, and the user interface seems smoother. The other advantage today is that Merge has been in the field for a longer time so more is known about it. On the other hand, SCO has a larger installed base and can therefore afford to put more effort into the product. Six months from now there could be a clear difference, but I expect exactly the opposite. I expect the products to become more alike as the market expands and matures.

A final note concerning Merge 286. If you have a DOS system at home and want to get your feet wet with UNIX, seriously consider this package. It will allow you to continue to run your DOS programs and give you access to real UNIX system with only a very small investment in hardware for some additional RAM and possibly a larger hard disk.

On the other hand, if you are computer shopping, buy a 386 system. The difference in cost between 286 and 386 systems is small, the peripherals are generally interchangeable, and the performance improvement is significant. Then purchase a version of UNIX and a DOS support product that fits your specific requirements. □

Did you find this article particularly useful? Circle number 5 on the reader service card.

Product Information

DOS Merge 386
Unlimited $499
Two-user $399
DOS Merge 286
Two-user Version $249

Microport Systems Inc.
10 Victor Square
Scotts Valley, CA 95066
(408) 438-8649
Circle reader service #258.

DOS Merge 386
Unlimited $1,195
Two-user $695
Unlimited (with UNIX) $1,995
Two-user (with UNIX) $795

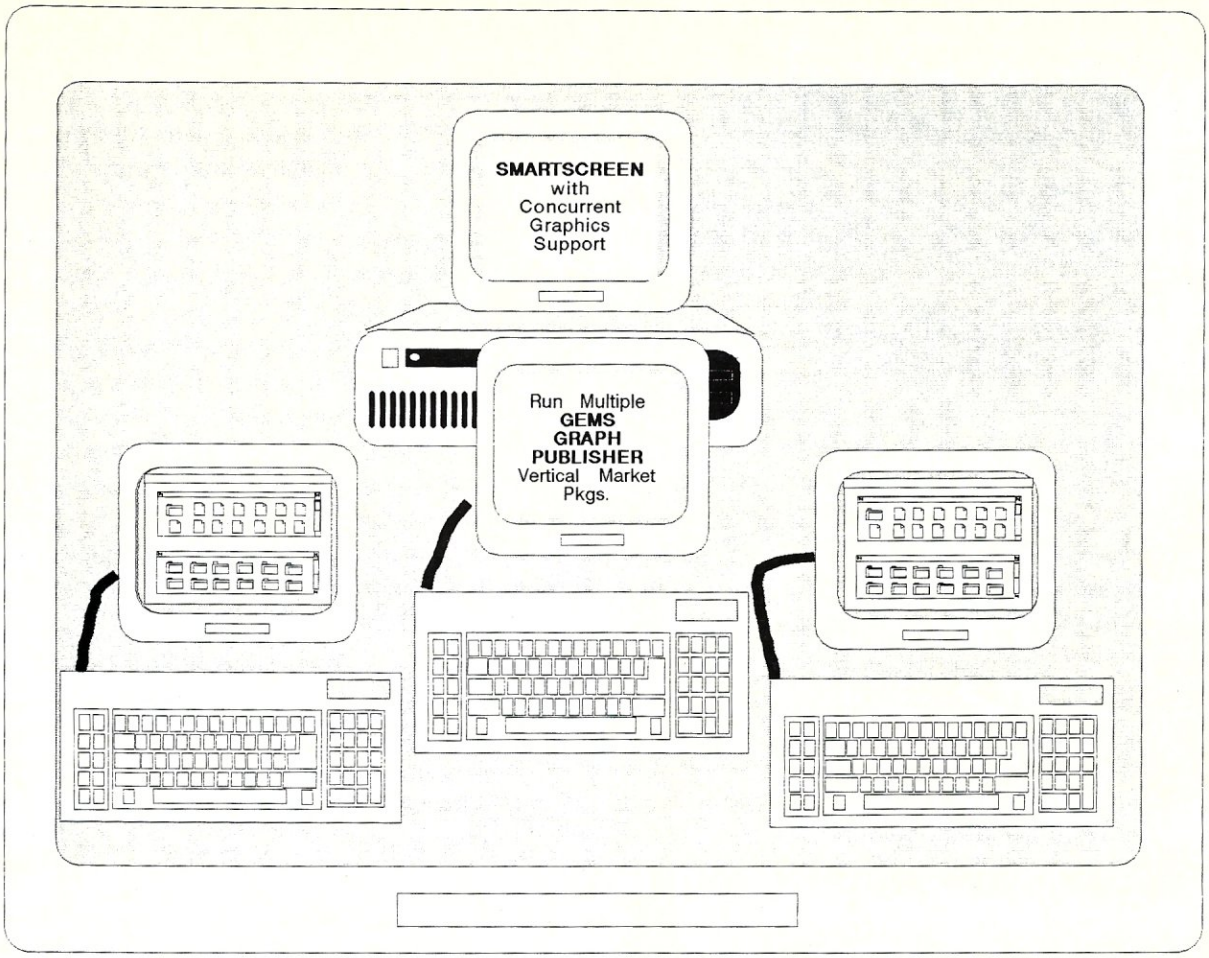
Locus Computing Corporation
3330 Ocean Park Boulevard
Santa Monica, CA 90405
(213) 452-2435
Circle reader service #259.

VP/ix
Unlimited $795
Two-user $395

Interactive Systems Corporation
2401 Colorado Avenue
Santa Monica, CA 90404
(213) 453-8649
Circle reader service #260

VP/ix
Unlimited $995
Two-user $495

Santa Cruz Operation
400 Encinal Street
Santa Cruz, CA 95060
(800) 626-8649
(408) 425-7222
Circle reader service #261



SMARTSCREEN

CONCURRENT SUPPORT FOR MULTIUSER GRAPHICS WORKSTATIONS

A multi-user DOS operating system with full graphics support!

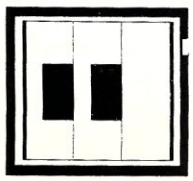
Smartscreen uses **cost-efficient** monitors and keyboards for 1- 8 users on 80386-based computers with monitor multiplexer boards to create a **high-performance** multiuser system with **graphics** capabilities **for every user**.

Smartscreen, an enhanced version of Concurrent Dos 386 Release 2.0, supports **full Hercules graphics** or **Color/EGA capabilities** and **multitasking** at every station, and runs PC-DOS and Concurrent software for maximum flexibility. You can have **two tasks** at every monitor for increased productivity. **Smartscreen** also supports local devices - mouse, serial printer and optional parallel printer - at each station for added flexibility.

Smartscreen allows application software to write directly to the video RAM at all stations for high performance and full PC graphics capabilities, eliminating screen emulation and serial slowdown.

Smartscreen has special versions available for optimized performance of selected vertical market packages, including MC Software's **INCOME & INMASS, Medical Manager, Digital Dining, Dataflex** and other **DBMS**-based applications. Also supported are most **desktop publishing** packages and selected **design and CAD** packages.

SMARTSCREEN includes **CCI support** - prompt, professional support when you need it, tailored to dealers, distributors, and developers. **For more details on Smartscreen and supported multiplexer boards,** call us today.



CONCURRENT CONTROLS, INC

A Digital Research Inc. Master VAR

925 Linden Avenue, South San Francisco, CA 94080

(415) 873-6240 FAX (415) 873-6091

Optimizing UNIX's Serial Character I/O

by Terry Keene

Serial character input/output is the life's blood of any multiuser system, so enhancing character processing is the key to improving multiuser performance.

Since the mid-1960s, time-shared, multiuser computer systems have been used in major corporations, government, and institutions equipped with mainframe and large minicomputers. The need to share information and computing resources has existed since the first computers were developed. The promise of efficient, affordable multiuser computing systems has been touted by major industry players since the announcement of the first 16-bit microprocessor. This promise has remained largely unfulfilled, however, due to the lack of an industry-accepted multiuser operating system, although many have existed since the mid-1970s.

The UNIX operating system, released in 1973, opened new vistas in portable operating environments. It has been adopted by many large manufacturers for use in scientific and engineering applications, as well as narrowly targeted, vertical markets supported by independent value-added-resellers (VARs). These manufacturers include, most notably, Hewlett-Packard, Digital Equipment, AT&T, Honeywell, and Unisys. Even IBM has had UNIX running on most of its systems, from the 370 architectures to the PC, for several years without much fanfair, and without much support.

Several years ago, Microsoft purchased a UNIX license from AT&T and created the first commercial UNIX implementation for general office/business use on small systems. Microsoft named it Xenix and began distributing it to OEMs. The Santa Cruz Operation (SCO) joined forces with Microsoft to enhance Xenix. SCO adapted Xenix to the AT&T SVID (System V Interface Definition) standard and announced SCO Xenix System V in 1985. Several other developers have adapted UNIX for the Intel-based systems, but SCO has garnered the lion's share of the microcomputer UNIX market.

The promise of affordable multiuser computer systems can now be fulfilled using UNIX as the operating system. With the right combination of hardware peripherals, UNIX installed on a 286/386-based system will efficiently support up to 32 users. The important phrase is "right combination of hardware peripherals." An 80386, 20-MHz-based computer with three or four terminals employing standard asynchronous communications ports will grind to a

---

Terry Keene is President of Technology Research Group Inc., an Atlanta-based Xenix systems integrator/distributor. He has a master's degree in electrical engineering from Georgia Institute of Technology and taught at the school for four years.

halt with just a few user processes executing. That same computer with an intelligent serial communications subsystem can handle 32 terminals running simultaneous processes with little or no degradation. There are other considerations and system parameters that must be discussed to qualify this assertion. These include hard-disk access, memory size and speed, caching for both system memory and disk I/O, as well as the nature of the applications being executed. But these are topics for discussion in later articles.

I elected to discuss the number of terminals and processes as a factor in CPU performance because timesharing multiuser systems were developed to support interactive terminal (*tty*) devices. UNIX is no exception. UNIX was originally developed by programmers for programmers. The intense keyboard activity of programming is a critical factor in system performance. In fact, much of the UNIX design philosophy hinges around character processing. It has the highest priority relative to interrupts and commands significant system resources. For large character input/output functions, like word processing and text editing, this is significant.

This article will deal with UNIX character processing and how it can be enhanced to improve multiuser performance.

How UNIX Handles Character I/O

Character I/O is initiated by a user process requesting either that the UNIX kernel read or write one or more characters from a particular character device. For this discussion, this device will be a standard ASCII terminal. Most other character devices provide similar character type interfaces. Two types of requests are recognized for character I/O: write characters from the user process to the device, or read characters from the device to the user process. When either type of request is made to the kernel, a set of programs are invoked by the kernel to perform these tasks. These processes, known as "device drivers," perform all of the functions necessary for the requested task.

The UNIX kernel provides a set of internal buffers in support of character routines known as character lists or *clists*. Each *clist* is a linked list of blocks of characters known as *cblocks*. Device drivers for interactive terminals use *clists* extensively. A structure is declared for each terminal device available on the system. This structure contains headers for three *clists* to be used by the device driver for that device. The first clist is the "raw" queue, which holds the characters exactly as they are passed from the terminal to the computer in 8-bit format. The second clist is the "cooked" or "canonical" queue, which holds characters from the raw queue that have been processed by the "line-discipline" routines. The line-discipline routines process the characters requiring special handling for UNIX, such as rubouts, delete line, expanding tabs to spaces, inserting carriage returns with line feeds, and other processing as defined in the *tty* structure. (The line-discipline routines will be discussed in more detail later.) The third *clist* is the "output" queue that holds characters to be sent to the terminal.

Upon receiving a request for character processing from a user process, the kernel invokes the appropriate set of drivers for the specific device requested. The device driver routines perform five basic functions; open, close, read, write, and *ioctl* (I/O control). On the call to the device driver, the device is opened by the driver. If the request is for a write to the device, the kernel invokes the write routine which copies the requested characters from the user process space into the output *clists* for the device. The driver then begins the output process, which is continued by the interrupt handler within the driver. The

interrupt handler processes characters to the device one character at a time until the output queue is empty. If the number of characters to be output to the device is larger than the output *clist* can accommodate, the write driver manages the copying process to the output *clist* using a high-water, low-water scheme—the write routine fills the *clist* to the high-water mark and then goes to sleep while the interrupt handler empties the queue. When the low-water mark is reached, the write process is awakened to refill the queue. This continues until all of the requested characters are transferred to the queue.

If the request is for a read from the device, the kernel invokes the read routine initiating the read process. Characters are input from the device by the interrupt handler into the raw queue and immediately transferred to the output queue as they are received so that there appears to be immediate echo to the screen. As characters are received into the raw queue, the read process transfers the characters to the cooked queue, processing all of the special characters as it makes the transfer. Upon receipt of a new-line character (typically a carriage return), the read process passes the characters to the user process for processing. This reduces the user process overhead. This mode of processing is known as "cooked" mode. The user process can request "raw" data, that is, receipt of each character as it is received without special processing. In this "raw" mode, the read process does not wait for the new-line character to awaken the user process with data, but rather passes characters to the user based upon either timed input or minimum packet sizing.

Line-disciplines are processes that are typically standard within the kernel. These routines process the special characters during the input/output process. In fact, the read and write routines in the device driver actually call line-discipline routines to pass characters from the user space to the output queue, and from the raw queue to the cooked and output queue.

The line-discipline routines provide both input and output processing. Input processing provides for erase and line-kill character processing. When the erase character is received, the line-discipline removes both that character and the previous character. Echo can be disabled for raw input/output through the line-discipline, and output echo can be selected so that an erase character will echo backspace/space/backspace to the terminal. New-line/carriage-return mapping, and lower-to-upper-case translation are handled. Eight-bit characters can be stripped to 7-bit for standard ASCII character processing, and tab characters can be expanded to spaces.

Two very important functions are performed by the device driver and the line-discipline. One function manages software flow control, XON/XOFF processing, and the other function manages hardware control of modem lines or printers. Software flow control provides the terminal user with XON/XOFF and functions to stop output and restart it as needed. XON/XOFF control is also used by terminal firmware for intelligent terminals to control the flow of characters to the terminal from the CPU. This prevents loss of characters. The modem control protects the system security by either logging the user off when he hangs up, or hanging up the user when he logs off. Either case could prevent potential system security and/or access problems.

All of the device driver and line-discipline capabilities are maintained in a set of parameters in the *tty* structure created for each terminal device. These capabilities are available to the user through *stty* system calls. The *stty* system calls, in turn, cause the kernel to invoke the device

driver's *ioctl* process. *Ioctl* processes provide the program interface to the device to modify such device parameters as baud rate, data bits, start and stop bits, and other processing done by the device.

In raw-mode processing, the line-disciplines are still used but the *tty* structure is changed to prevent any special processing during the transfer. There are some obvious opportunities to optimize character processing in the raw mode by bypassing the line-disciplines during transfers (which is discussed later).

The major drawback affecting system performance is that each time the device has a character available to be read by the device driver interrupt routine, or each time that the write or read routine places a character into the output queue, a high-priority hardware interrupt is generated. With each interrupt, the CPU stops processing the current task, saves the system context, invokes the appropriate interrupt routine, processes the interrupt, reinstates the previous machine state, and continues processing. If terminal I/O is in cooked mode, the interrupt handler also provides character "cooking," tying up the CPU even longer. If the I/O device has multiple serial inputs that activate a single interrupt, then the interrupt routine also must poll the devices to find the interrupting device that takes more time. If the programmer who writes the device driver does not understand all of this (or doesn't care about other system processes), then he could well decide to optimize his driver by reading multiple characters at a time from the device by putting his interrupt handler to sleep, or in a loop-waiting for a relatively slow device, making the whole system I/O-bound by his device. In these cases, even the system clock suffers since it has a lower interrupt priority than the terminal I/O and it begins to lose time. Since the device drivers provided by the UNIX vendors and OEMs were written by UNIX programmers, these programming mistakes tend not to happen. But the interrupt overhead on the CPU cannot be avoided if the I/O device is a typical "dumb" serial I/O device using a standard UART under interrupt control.

Standard Serial I/O Ports Limit Performance

Since all other device I/O —i.e., hard disk, printers, keyboards, consoles —are at a lower interrupt level than serial interrupts, the rest of the system suffers degradation as the number of terminals increase. With heavy terminal usage for keystroke input and screen refresh output, system degradation will become quite significant. In the case of a 286-based, 6-MHz machine, a single terminal on the standard serial I/O port processing output to the terminal at 9600 baud demands 87 percent of the CPU time during output. On a 386-based system, the same output consumes 30 percent of the CPU time. It is not hard to see that dumb serial boards on these systems will quickly consume all of the CPU processing time and degrade the system to the point where it becomes unresponsive and unusable.

Multiport "dumb" serial boards also do little to improve throughput. These boards typically multiplex multiple ports onto a single memory-mapped I/O port and share the interrupt of that port. The CPU character handling overhead is not reduced. In some cases, the device drivers for these boards poll each separate port on the board to see which port caused the interrupt. In the process of polling, the device drivers block out lower priority interrupts to the CPU for an extended period of time. If the clock interrupt is at a lower priority than the serial I/O, the clock will not be updated and will appear to run slow.

Using An Intelligent Serial I/O Controller

To avoid severe system degradation with increased numbers of users, the first area to try to improve system performance and throughput is with intelligent serial input/output boards. These boards offload processing from the CPU and can make dramatic improvements in terminal I/O processing.

Intelligent multiport I/O boards contain buffers, typically of the ring-buffer type, which continually fill and empty at the same time. As characters are entered at the terminal keyboard, they are buffered in the board's memory and do not necessarily interrupt the CPU. When characters begin filling the ring-buffers, the on-board processor collects the characters and interrupts the main CPU. The characters are typically collected in another area of the board's memory for transmission to the main CPU. The board's processor transfers chunks of data, sometimes complete *cblocks* at a time, to the internal *clist* structure of the appropriate *tty*. These transfers take place under control of the device driver interrupt handler for the intelligent device. The read process then invokes the line-discipline to process and echo the characters, as in the case of the dumb board. As the intelligent board processor is transferring characters to the main CPU, characters continue to fill the ring-buffer for processing. Because keyboard input is typically sent in bursts, different amounts of data will be transferred in a single interrupt. In each case, the ring-buffer will continue to fill, independent of the transfer block size or frequency.

The relative efficiencies of the intelligent board depend upon several characteristics of its implementation. It is possible to improve the efficiency of the data transfer between the kernel and the device by implementing dual-ported memory that both the kernel and the device can access. In this manner, the CPU can read characters directly from the board memory as the input *clist* and save one level of overhead. The same is also true in the case of the output *clist*. For purists, this is really the character control block, not the *clist* buffers.

Flow control to the terminal will be more efficient if the size of the ring-buffer is larger. Further, efficiencies can be obtained through optimization of the software executing on the intelligent board. The efficiency of the ring-buffer algorithm, the handling of the double buffering and interrupts by the board's processor, dynamic allocation of available ring-buffer memory to each port, and automatic character echoing (with the necessary processing) can differentiate one board from another in overall throughput and CPU overhead. For example, it is reasonable to assume that one device, such as a modem at 1200 baud, would not need as much memory for ring buffering as would a terminal operating as 38,400 bits per second (bps). Additional functions, such as flow control (both XON/XOFF and RTS/CTS), can be allocated to processing on the intelligent board to improve efficiencies even further.

Intelligent boards can suffer some of the same pitfalls that dumb boards experience if the ports on the board are polled for input while interrupts are disabled. Since most of the intelligent boards support more ports than most dumb multiport boards, this problem could be exacerbated.

The obvious advantage of the intelligent board handling of buffered characters for interrupt processing is to reduce CPU overhead in handling interrupts. Such buffered multiport boards can offer a 30-percent to 50-percent reduction in CPU overhead and a comparable improvement in overall throughput. However, as the number of active terminals increases, the CPU overhead continues to climb until the same symptoms appear as in the dumb boards. This is a result of the character-at-a-time processing taking place in the device driver and the line-discipline

processing special characters and transferring characters between queues as required. This internal processing of characters is the dominant factor in the overhead of terminal I/O processing. An intelligent board that could offload this character processing could provide dramatic improvements in reducing CPU overhead and increase overall terminal throughput.

A complete, intelligent I/O coprocessor subsystem that provides the foregoing character buffering mechanism, as well as character processing of the line-discipline, would make great strides toward the realization of the 32-user 80386-based UNIX system. This type of subsystem would provide tab expansion to spaces, end-of-line mapping, character delays as necessary, and case mapping provided on output by the UNIX line-discipline without any overhead from the main CPU. It could also perform the input conversions and character handling for erase, line kill, signals, and timeout/packet size processing required for raw mode. The main CPU would only process characters as raw characters. It would only have to initialize the user process I/O request and handle the kernel portion of the device driver for data transfer between user memory and the I/O board's memory.

The need to use the kernel's internal buffering *clist* mechanism could be eliminated if the subsystem and driver software were implemented efficiently. Rather than transferring data into a *clist* for processing, the device driver would transfer data directly into dual-ported memory on the I/O subsystem board for processing. This would be raw data, and all required processing would take place on the subsystem board. Input data would likewise be processed on the subsystem board and the interrupt handler would simply copy the cooked data from the board's dual-ported memory to user memory space so it is ready to be used by the user process. Processing by the main CPU would be memory-to-memory, similar to a caching mechanism, and the overhead would be dramatically reduced.

Since this processing is relatively short in duration, the addition of active terminals would mean only a relatively small incremental increase in CPU overhead per terminal. This would have little effect on executing processes and overall system degradation would be minimal.

Offloading the line-discipline and buffer processing to an intelligent subsystem is not without its problems. As in the case of the CPU's character processing for dumb and intelligent boards, overhead on the coprocessor servicing the I/O subsystem will increase dramatically as the number of active terminals increase. This could cause overall throughput to suffer at the I/O subsystem. Although system degradation would be minimized, character throughput could choke the system and the symptoms would be very much like the cases mentioned with dumb and intelligent boards. To truly optimize system performance, the intelligent subsystem must support the classic line-discipline functions differently than UNIX. The broad objective of developing the UNIX operating system was portability, which often meant trading off efficiency and eloquence.

Some of the line-discipline functions could be optimized by modularizing these functions into a set of standard processes. These processes would represent normal input and output configurations and would not process little-used functions like upper-to-lower-case mapping and character delays for slow terminals. A typical output function would be tab expansions to spaces and new-line/carriage-return mapping. This would optimize output processing, which accounts for 90 percent of the I/O function, and would drastically reduce the coprocessor overhead. Similar processing modules could be developed for input

processing and for character echo.

Raw character processing also could be optimized by simply bypassing the line-discipline routines for special character processing and streamlining the flow of characters from the asynchronous port to the buffered memory. Character timeout and minimum packet size processing could be handled by the subsystem and raw mode. I/O would be efficient and very low in coprocessor overhead.

In addition to optimizing system performance by offloading and optimizing line-discipline functions, the intelligent I/O subsystem could further improve performance using dual-ported memory through which the device driver transfers large amounts of data, either to or from user memory, for a single-user I/O request. With sufficient memory on the intelligent board, a single screen refresh could be handled through a single request and a transfer of the screen from user memory to the subsystem memory for immediate processing, all on a single interrupt with no character processing overhead.

It is easy to see why manufacturers of complete intelligent I/O subsystems claim dramatic improvements in overall character throughput and reductions in CPU overhead. In some cases, intelligent subsystem manufacturers have achieved overall throughput in raw mode of approximately 80,000 baud with 8 active 9600 bps terminals compared to approximately 40,000 baud for intelligent boards and 20,000 baud for dumb boards. These same subsystem manufacturers claim cooked-mode CPU overhead of less than 20 percent for 8 terminals compared to 90 percent to 100 percent for intelligent and dumb boards.

One of the obvious disadvantages to the intelligent I/O subsystem is the need to develop independent firmware (the software installed on the board) with each implementation and modification to the operating system. This would not be nearly so prevalent with intelligent boards that do nominal processing in addition to character buffering. This problem effects board manufacturers, but could also have an impact on system users since upgrading the operating system to the latest release could mean upgrading the EPROMs and drivers for the subsystem board as well. It could mean taking down the system and in extreme cases could mean shipping the board to the manufacturer for an upgrade.

Some manufacturers have overcome this potential problem by providing their firmware with the installable drivers, and having the initialization portion of the drivers download the firmware at boot time. The memory on the subsystem board serves as normal system memory as well as buffers for character I/O. This reduces the software upgrade problem for the user. More manufacturers should follow this lead, even for intelligent boards.

The last advantage of intelligent I/O boards, both intelligent and subsystem boards, is their ability to add features to the I/O processing not available in dumb boards. The main feature available in intelligent coprocessor-based I/O boards is transparent print—the ability to connect a serial or parallel printer to the back of an ASCII terminal without installing two cables and using two ports on the I/O board. This feature allows two data streams, one for the terminal and one for the printer, to be multiplexed over a single set of cables to the terminal. The printer is connected to a separate or auxiliary port on the back of the terminal. A special character code sequence (escape sequence or control code) is sent to the terminal to shut off the screen output and redirect output to the auxiliary port. A second sequence returns output to the terminal screen.

The newer ASCII terminals are microprocessor-based

with many added features. These terminals typically have large character buffers. Printers, on the other hand, have relatively small buffers because they process those characters more slowly through the print head mechanism. If properly tuned, this provides the opportunity to multiplex the two data steams so that the printer can be employed at maximum capacity without any apparent interruption in terminal processing. Small bursts of data can be sent to fill the printer buffer and be processed by the printhead mechanism while large bursts of data would be sent to the terminal buffer. By manipulating the size of these two data steams, the printer and terminal will appear to operate simultaneously without hesitation.

Additional features are available through intelligent coprocessor serial boards. Multisession support can be implemented whereby the terminal user can log into multiple sessions on separate virtual screens on the terminal. Some boards implement this function using the multipage memory inherent in newer terminals, and some boards implement this feature on the intelligent board through use of terminal memory mapping on the board and virtual terminal algorithms. Function key mapping and function key programmable macros also are features that have been implemented on some intelligent boards as a result of the power of the coprocessor. No doubt we have just begun to see the expansion of features that will appear when the power of the intelligent peripheral becomes the norm for multiuser systems.

The UNIX Login Process

The line-discipline process handles character processing or "cooking" for the device driver. This cooking of each individual character is a function of the requirements of the UNIX command-level syntax and the requirements of the applications program. Character processing is performed according to values set in a software structure for each terminal line, called the *termio* structure. This structure establishes: first, whether characters will be processed in a cooked mode or raw mode, and then what type of additional processing will occur. Options for input character processing include signal handling for interrupt and quit signals, erase character and erase-line characters, flow control characters XON(ASCII dc3) and XOFF(ASCII dc1), new-line/carriage-return processing, and upper-to-lower-case mapping. Output processing options are also selected in the *termio tty* structure and include new-line/carriage-return mapping, lower-to-upper-case mapping, expansion of tabs to spaces, and necessary delays for new lines, carriage returns, and form feeds for slower hard-copy-type terminals. Line characteristics for each terminal also are contained in the *tty* structure, including baud rate, data bits, stop bits, parity and special hardware signal processing for data terminal ready, carrier detect, and request-to-send signals. The last section of the *tty* structure determines how the line discipline will interact with the terminal driver. From this section the line discipline gets information describing whether to process signals, whether input will be cooked or raw, and how to handle character echoing to the terminal.

The *tty* structure is read by a program called "getty." Getty initializes the terminal line and opens it for processing. Initial settings for the *tty* structure can be established through the use of a system file called GETTYDEFS. GETTYDEFS contains a series of mnemonics that describe settings in the *tty* structure. When getty is called to open a terminal line, it is passed the line name, or device name, and an index into the GETTYDEFS file. Getty reads the GETTYDEFS file for that index and sets the *tty* structure

accordingly. Getty then opens the line, which causes the device driver to invoke the line discipline using the *tty* structure set up by getty. The *tty* structure can be altered as necessary through either an I/O control command to the driver, *ioctl*, or through the *stty* system command. These commands allow the interactive user or the applications program to alter the character processing as necessary after the line is open.

Cooked character processing by the line-discipline means that characters are passed to the requesting process on a line-by-line basis. That is, regardless of the number of characters requested, the device driver passes characters to the requesting process after a new-line or end-of-file character has been received. End-of-line in the UNIX file system is indicated by the new-line (line feed) character and end-of-file is indicated by an EOT (control D) character. If raw input processing is selected then characters are passed to the requesting process as received by the device driver. In raw mode, it is also possible to accept bursts of characters that should be processed together, such as escape sequences and programmed keys like special function keys, by setting the time flag in the *tty* structure. This flag causes the driver to wait for a specified period

In UNIX remote terminal connections, the critical signals are between the DCE and the DTE.

of time before passing characters to the requester. When the time has expired, the driver passes all characters received during that time to the requesting process. This allows small groups of characters to be processed together.

Getty opens the line with initial parameters, typically parameters capable of accepting characters from most generic terminals, and issues a login prompt on the line. Upon acceptance of the login entry, getty establishes the final character processing parameters from GETTYDEFS and invokes the login process. As long as a line is to be used as an active terminal, all character processing can be established through the GETTYDEFS file and custom login parameters can be established after login. If the line is to be used for other than an active terminal, such as for a printer, getty should not be invoked for that line to avoid a login message. GETTYDEFS will not affect the character processing. In this case, a default set of *tty* parameters will be established for the line by the system developers supplying the UNIX system. If these parameters are not acceptable for a given device, for example a slower 2400-baud printer or a printer that can only provide XON/XOFF flow control and has a small character buffer, the parameters will need to be changed through the *stty* system command. This can be accomplished through the printer spooler interface to a specific device, or through an interface script or program, or through the system initialization file that executes at boot time.

Remote access to a UNIX system via modems presents additional requirements for serial I/O processing. Modems for dial-up lines typically operate at speeds slower than normal terminals. Parameters for the *tty* structure must be changed to process these characters. In addition, security becomes more important since there is no physical control available as with local terminals. It is necessary to ensure that, when a user dials in to the system, the user has permission to process. When an authorized user exits the system, the line must be cleared for the next user access, and if the line drops off or hangs up before the user logs off, that user must be automatically logged off and the line cleared. All of this activity can be controlled through the same *tty* structure that controls character cooking.

Handling DTE/DCE Serial I/O

There are two types of devices that interface through the RS-232C protocol. The equipment that provides the interface to the source of information is known as the Data Terminal Processing Equipment or Data Terminal Equipment (DTE), and the equipment that provides the signal conversion to the communications link is the Data Communications Equipment (DCE). Each type of equipment provides a different part of the RS-232C interface. Data is transmitted from the transmit line of the DTE to the transmit line of the DCE through the communications line. It is then received by the receive line of the distant DCE and retransmitted to the receive line of the distant DTE. This takes place under one of two types of data flow control. The first is software control provided in the data stream and recognized by the character processing at both ends of the link, such as XON/XOFF or an ACK/NACK type of control. The second is hardware flow control as provided for in the RS-232C standard. This flow control is typically handled by two lines called request-to-send (RTS) and clear-to-send (CTS). The DTE typically activates RTS when it is ready to send data, and the DCE activates CTS when it is ready to receive data. Hardware flow control is usually not necessary unless the rate of data flow is sufficiently fast and the buffer size of the connected DTE and/or DCE is so small that it causes data to be lost during transmission. Typically software flow control such as XON/XOFF will suffice, but the delay in transmission, processing, and turnaround of the communications line could make software control too slow to be effective. For most modem connections, software flow control will be more than sufficient.

The critical signals in UNIX remote terminal connections are not the flow control lines but rather the initial connection signals between the DCE (modem) and the DTE (computer serial I/O port). These signals are called data terminal ready (DTR) on the serial port and carrier detect (CD) on the modem. The DTR is activated when the terminal is ready to receive data. In UNIX, the DTR line is activated by the device driver under a set of possible conditions. The first is that the *tty* structure indicates that the device connected to the line is a remote device. This flag in the *tty* structure is the *LOCAL* flag. If CLOCAL appears in the GETTYDEFS file, or if *clocal* is entered through the *stty* system call, the driver assumes that the device is local and DTR, CD, RTS, and CTS processing are ignored. If CLOCAL does not appear in GETTYDEFS, and the supplier of the operating system did not assume that the default line was a local line, or if the *-clocal* flag is used with the *stty* system call, then DTR, CD, RTS, and CTS signal processing are enabled. With processing enabled, upon opening of the line the DTR signal is activated, telling the DCE that the DTE is ready for processing. Typically, modem control will prevent the modem from answering a call if the DTR line is not active. (This is usually a setting on the modem and should be selected during setup.) If

the device driver for the serial port supports modem control, then the driver will await a CD signal from the modem before getty issues a login to the remote device through the modem. This prevents unauthorized access to the system if the port is not enabled since DTR is not active until the port is opened. The DTR signal can also be used to cause the modem to hang up the line, or go "on-hook" if the DTR line is dropped.

The other critical line from the modem to the serial port is the carrier detect line. When the modem answers an incoming call (goes "off-hook"), it sends the distant modem a carrier tone. When the distant modem receives carrier, it returns carrier to the local modem. Upon receipt of the carrier, the local modem handshakes with the distant modem, then activates the CD line to the serial port. There is another *tty* structure flag that plays an important role in remote terminal processing. The HUPCL or hang-up-on-close flag causes the DTR line to be deactivated upon close of the last process associated with the line. The combination of the not CLOCAL and the HUPCL flags provides security for remote terminals, plus it can provide ease of terminal support for local terminals.

Use of the CD and DTR lines for remote terminals provides two critical functions for system security. The first is control of access to the modem. If the line with the modem is not open or enabled, the modem will not answer the phone line. If the distant terminal logs off or closes all processes on the terminal, HUPCL causes the DTR line to go inactive to the modem and the modem will hang up, which prevents line charges in the case that the distant terminal failed to hang up the modem. It also makes the line available to other remote users. In addition, if the distant terminal hangs up the modem without logging off or if line quality or interruption causes the modem to hang up prematurely, the loss of CD causes a hang-up signal to be sent to all processes attached to the terminal line logging off the remote terminal. This causes getty to recycle for a new login, and resets the modem line for the next remote access.

These two signals can also be used for local terminal control. If the same *tty* structure parameters are set to a local terminal, and control signals are implemented by running cable connections to the Tx, Rx, DTR, CD, etc., pins in the normal implementation of RS-232C on a DB-25 connector, then turning off the power on the terminal has a similar effect to modem control. If the terminal hangs or is non-responsive to the CPU, then powering off the terminal drops the CD line to the serial port which drops the line, the hang-up signal is sent to the terminal processes, and a new getty is spawned. This will usually clear non-responsive terminals and close all files and processes connected to that terminal without corrupting files or databases. Typically, the RTS and CTS lines of the DB-25, RS-232C connection should also be run or at least looped back upon themselves to maintain continuity and guarantee system performance.

Summing Up

Serial character input and output is the heart and soul of a multiuser system. If implemented correctly and with appropriate intelligent controller support, 286- and 386-based UNIX systems can support up to 32 terminals and beyond for specific applications. The market for cost-effective computing is growing rapidly, and technology is available to take advantage of that growing base. Understanding serial I/O is an essential first step to taking full advantage of this market growth. □

Did you find this article particularly useful?
Circle number 6 on the reader service card.

Programming the 8086/8088 To Perform

by Kevin Parker

Creating tight, fast programs that operate close to the CPU is the objective of all programmers. But there are times when you want to trade program size for speed.

When confronted with both simple and complex problems, most programmers will go to a high-level language such as C, FORTRAN, Pascal, or BASIC. These languages offer portability, short development time, easily modifiable program modules, and can provide satisfactory program performance. High-level languages, however, do not provide the most compact machine code or the fastest execution time. When such a program has unacceptable performance, you can usually determine where the program spends most of its time and re-code these portions in assembly language.

Many applications demand better performance than can be achieved with a mixture of low-level and high-level language programming. When maximum speed and/or minimum code size are essential, the only programming language that fits the bill is one that brings the programmer as close to the CPU architecture as possible—assembly language.

What types of applications require assembly-level source programming? Many powerful operating systems are programmed in assembly language to minimize size and enhance speed. The highest performance software packages available on the market today were written partially or entirely in assembly language. In many "real-time" processing systems (signal processing, process control, data collection and recording, etc.), assembly language is the only choice because the system must be synchronized to some external clock via the microprocessor's interrupt mechanism while heavily taxing the machine's capabilities. For example, a real-time communications signal processing system design I am currently involved with uses five VME-bus, 16-MHz, 68010 processors running in parallel at nearly 100-percent capacity with assem-

Kevin Parker is a communications engineer at the Johns Hopkins University Applied Physics Laboratory where he designs and develops advanced microprocessor-based submarine communications systems. He has a BSEE from Virginia Polytechnic Institute and State University in computer architecture.

bly language software. Using a compiled language would require at least one more CPU, and this additional processor would increase the cost and reduce the reliability of the system.

Use of assembly language alone, however, will not guarantee maximum performance. To achieve highest speed and/or smallest code size, assembly language programmers must carefully consider and exploit the instruction set and CPU architecture. Additionally, the programmer must consider algorithm design. For example, a bubble-sort written in assembler would probably run more slowly than a quick-sort written in C.

This article discusses details, techniques, and considerations for optimizing the speed and size of 8086/8088 assembly language programs. I will conclude with a set of simple rules to follow when optimizing code (see the box). These rules and techniques can also be applied to some high-level languages and other microprocessors. I will assume the reader has a basic understanding of the 8086 instruction set and architecture. Those who need a refresher should review *The 8086 Family User's Manual* and the references listed at the end of this article. All performance considerations and rules discussed in this article apply to both the 8086 and 8088 CPUs, except where otherwise noted. I will not discuss algorithm design; countless authors have devoted books and articles on the subject for sorting, searching, floating point math, random number generation, etc.

Speed versus Size—The Ultimate Tradeoff

Contrary to what you might expect, a program written for minimum execution time is larger than a similar program optimized for size. This means that, for any given application, you must decide which is more important, compact code or high speed.

In most situations, it is easy to decide when to trade off size for speed. Multiple decision-making and repetitive operations, such as sorting and searching algorithms, should sacrifice code size for speed. Interrupt and fast peripheral service routines should also be written for maximum speed. In controller applications where ROM space is at a premium, compact code is more important than speed. It is common to find engineers squeezing the last few bytes of code out of a program so that one or two ROMs may be eliminated from the target hardware.

Counting Clock Cycles and Bytes

The only way to squeeze the last ounce of performance out of your machine is to keep track of the execution time and code size while you write the code. That's not to say you must calculate these numbers for every instruction in your program. Instead you should understand the general characteristics of the instruction set. We will review the instruction set and derive a few simple rules for program optimization.

Table 1 shows typical performance characteristics for most data transfer, arithmetic, and logic instructions. The "execution time" column specifies the number of CPU clock cycles required to perform the instruction. For example, if an 8086 runs at 8 MHz (8 million clock cycles per second), then the instruction ADD AX,5 would take 0.5 microseconds (4 clock cycles) to execute.

It can be seen that instructions are most compact and execute faster when the operands are registers. This brings us to the first rule of Performance Programming: To save time when multiple iterations or operations are to be performed on a piece of data, keep that data in a register(s) as long as possible (as shown in Examples 1

and 2). When used in moderation, this rule can save significant amounts of time. Notice Example 1 is 20 percent faster than Example 2, but it is also four bytes longer.

Unfortunately, the 8086 does not contain many registers. Therefore, you can get into trouble when simultaneously manipulating a lot of data because you end up spending more time swapping registers and memory locations than processing data. In these cases, it is best to keep only the most often used operands in registers.

Memory Operands and String Operations

Table 1 illustrates that execution time of memory operand instructions depends primarily on the number of memory transfers that occur, the addressing mode used, the data bus width (CPU type), and the operand alignment.

All the logic and arithmetic instructions (except for TEST and CMP) require two memory transfers if the destination operand is in memory. Consider the following statement:

```
ADD   SUM,AX
```

In this example, the CPU must fetch SUM from memory, add the contents of AX to it, and then store the result in memory location SUM. Total execution time is 22 clock cycles. If the two operands are swapped so that AX is the destination, the total execution time is reduced to 15 clocks. This is because SUM is only read and not written. Therefore consider this corollary: When memory variables are used in arithmetic and logic operations, try to use them only as source operands as much as possible.

Remember that displacements used in base and indexed addressing modes can be either 16-bit or signed 8-bit numbers. If data can be placed in a 256-byte (or smaller) block, you can cut code size considerably by setting an index (or base) register to point to the middle of the block and using indexed or base offset addressing to access the data. This causes the assembler to generate byte offsets instead of word offsets for the opcodes.

Example 3 shows a one-byte saving over direct addressing because three bytes are required to initialize the index register. This is a small savings in code, but it may prevent an additional ROM requirement. If a large number of data accesses can be executed before the index register is modified, the savings will be substantial.

Example 1

```
        .
        .
MOV   BL,SUM
ADD   BL,AL
NEG   BL
MOV   SUM,BL
        .
        .
        .
12 bytes/35 clocks
```

Example 2

```
        .
        .
ADD   SUM,AL
NEG   SUM
        .
        .
        .
        .
8 bytes/44 clocks
```

Example 3

```
                OPA  DW   ?
                OPB  DW   ?
                OPC  DW   ?
                OPD  DW   ?
                       .
                       .
START: MOV   SI,OFFSET OPC    ; Point to middle of data
       MOV   DX,[SI]          ; Access OPC
       ADD   DX,OPD-OPC[SI]   ; Access OPD
       SUB   DX,OPA-OPC[SI]   ; Access OPA
       MOV   OPB-OPC[SI],DX   ; Access OPB
                       .
                  14 bytes
```

ADVERTISER INDEX

On 8088 machines, 16-bit memory operand instructions require four more clock cycles per data transfer than the execution times shown in Table 1 (assuming no memory wait states) because the CPU's external data path is only 8 bits wide.

On 8086 machines (*not* 8088 machines), memory operand alignment has a direct impact on program speed. Whenever the CPU must access a 16-bit memory operand located at an odd address, it must split the operation into two 8-bit accesses. This adds four clock cycles to each memory access. Instructions that must access the operand twice (NOT, NEG, etc.) will take eight clock cycles longer to execute. This performance degradation can become quite appreciable during string operations. For example, an 8-MHz 8086 executing a 32K word string MOV operation will suffer a performance degradation of 33 ms (almost 50 percent) if the source and destination operands are not word-aligned.

This does not mean that byte string operations will execute as fast as non-aligned word string operations of equivalent length. Using our previous example, we find that performance is degraded by an additional 37 ms if a 64K move is performed instead of a non-aligned 32K word move. Therefore, if speed is of utmost importance and large amounts of data are to be moved (or stored), the operation should be performed with a MOVSW (STOSW) instead of a MOVSB (STOSB) instruction.

If the string length is always an even number of bytes, then converting the string length from a word count to a byte count can be done by simply shifting it right one bit. If there is either an even or odd number of bytes in the destination string, then a routine similar to the one shown in Example 4 can be used to convert a byte string move to a word string move.

For short string move operations, using multiple MOVS instructions instead of the REP MOVS instruction will reduce execution time and can reduce code size by a few bytes. Two different ways to move a four-word string are shown in Examples 5 and 6.

An added benefit illustrated in Example 5 is that the CX register remains unaffected by the string operation. Concerning execution time, the "break even" point for

Table 1. 8086/8088 Data Manipulation Instruction Performance

Instruction	Operands	Execution Time	Transfers	Length
ADD, ADC	REG,REG	3	-	2
SUB, SBB	REG,IMM	4	-	3-4
AND, OR, XOR	ACC,IMM	4	-	2-3
	REG,MEM	9+EA	1	2-4
	MEM,REG	16+EA	2	2-4
	MEM,IMM	17+EA	2	3-6
CMP	REG,REG	3	-	2
	REG,IMM	4	-	3-4
	ACC,IMM	4	-	2-3
	REG,MEM	9+EA	1	2-4
	MEM,REG	9+EA	1	2-4
	MEM,IMM	10+EA	1	3-6
AAA, AAS	-	4	-	1
DAA, DAS	-	4	-	1
NOT, NEG	REG	3	-	2
	MEM	16+EA	2	2-4
MUL (IMUL)	REG8	70-77 (80-98)	-	2
	REG16	118-133 (128-154)	-	2
	MEM8	(76-83)+EA ((86-104)+EA)	1	2-4
	MEM16	(124-139)+EA ((134-160)+EA)	1	2-4
DIV (IDIV)	REG8	80-90 (101-112)	-	2
	REG16	144-162 (165-184)	-	2
	MEM8	(86-96)+EA ((107-118)+EA)	1	2-4
	MEM16	(150-168)+EA ((171-190)+EA)	1	2-4
SHL, SHR	REG	2	-	2
RCL, RCR	REG,CL	8+4/bit	-	2
ROL, ROR	MEM	15+EA	2	2-4
SAR	MEM,CL	20+EA+4/bit	2	2-4
INC, DEC	REG16	2	-	1
	REG8	3	-	2
	MEM	15+EA	2	2-4
TEST	REG,REG	3	-	2
	ACC,IMM	4	-	2-3
	REG,IMM	5	-	3-4
	REG,MEM	9+EA	1	2-4
	MEM,IMM	11+EA	1	3-6
String Instructions				
LODS(STOS)	-	12(11)	1	1
repeated	-	9+13/rep (9+10/rep)	1/rep	2
CMPS	-	22	2	1
repeated	-	9+22/rep	2/rep	2
SCAS	-	15	1	1
repeated	-	9+15/rep	1/rep	2
MOVS	-	18	2	1
repeated	-	9+17/rep	2/rep	2

Miscellaneous Instructions				
MOV	REG,REG	2	-	2
	REG,IMM	4	-	2-3
	MEM,ACC	10	1	3
	ACC,MEM	10	1	3
	REG,MEM	8+EA	1	2-4
	MEM,REG	9+EA	1	2-4
	MEM,IMM	10+EA	1	3-6
PUSH (POP)	REG	11 (8)	1	1
	SEG REG	10 (8)	1	1
	MEM	16+EA (17+EA)	2	2-4
PUSHF (POPF)	-	10 (8)	1	1
LAHF,SAHF	-	4	-	1
CBW (CWD)	-	2 (5)	-	1
XCHG	ACC,REG16	3	-	1
	REG,REG	4	-	2
	MEM,REG	17+EA	2	2-4
LEA	REG,MEM	2+EA	-	2-4
LDS,LES	REG,MEM32	16+EA	2	2-4
XLAT	-	11	1	1
IN	ACC,IMM8	10	1	2
	ACC,DX	8	1	1
OUT	IMM8,ACC	10	1	2
	DX,ACC	8	1	1

Memory Operand Effective Address Calculation Time (EA)

Addressing Mode	Clock Cycles
[BX],[BP],[SI], or [DI]	5
Displacement only	6
[BP+DI] and [BX+SI]	7
[BP+SI] and [BX+DI]	8
disp+([BX],[BP],[SI], or [DI])	9
disp+([BP+DI] or [BX+SI])	11
disp+([BP+SI] or [BX+DI])	12

Note: Execution times assume instruction has already been fetched and no memory wait states are required. Add 4 clock cycles for each memory word transfer on 8088 machines and each odd addressed memory word transfer on 8086 machines. Add 2 clock cycles for each segment override prefix.

using the REP prefix is for strings that are 13 elements long. So, for the fastest possible code, use multiple MOVS instructions for string moves of 12 elements or less.

The above discussion also applies to the STOS and LODS instructions, although I am not sure how much value there is in repeatedly using the LODS instruction without some additional code.

By the way, string instructions that start with CX=0 will not execute! This prevents the machine from operating on a full segment of data (65,536 bytes) with a single byte-string instruction. The only way around this limitation is to perform the operation with CX=65535 and then execute an additional string instruction.

Multiplication and Division

Think twice before using the multiply and divide features of the 8086 hardware. If you are multiplying or dividing numbers by a power of two, shifting left or right by the appropriate number of bits can make performance up to 70 times faster than using the MUL and DIV instructions.

What if you are multiplying by numbers that are not a power of two? You should still consider using a multiply routine instead of the MUL instruction if one of the numbers you are working with is a constant value. For example, suppose the number *y* in the AL register must be multiplied by 10. Two ways to perform this multiplication are shown in Examples 7 and 8. Example 7 is nearly 11 times faster than Example 8, but also requires twice as much code.

If the application requires code size to be minimized, the single word AAD instruction can be used to multiply AH by 10 and add the product to AL. AAM can be used to divide AL by 10.

Example 4

```
        .
    MOV   CX,BYTE_COUNT
    MOV   DI,OFFSET DESTINATION
    MOV   SI,OFFSET SOURCE
    SHR   CX,1              ; Convert byte count to word
    REP   MOVSW             ;   count and move the data.
    JNC   NO_ADD            ; Move 1 more byte if BYTE_COUNT
    MOVSB                   ;   was odd before the shift.
NO_ADD: .
        .
```

Example 5

```
    MOVSW
    MOVSW
    MOVSW
    MOVSW

4 bytes/72 clocks
```

Example 6

```
MOV     CX,04
REP     MOVSW

5 bytes/81 clocks
```

Example 7

```
        .
SHL   AX,1    ;AX=2y
MOV   DX,AX   ;save 2y in DX
SHL   AX,1    ;AX=4y
SHL   AX,1    ;AX=8y
ADD   AX,DX   ;AX=8y+2y=10y
        .
11 clocks/10 bytes
```

Example 8

```
        .
  MOV   DX,10
  MUL   AX,DX
        .
        .
        .
        .
118 clocks/5 bytes
```

Example 9 (fast)

```
SHIFT_IT  MACRO
    REPT  4
    SHL   AX,1
    RCL   DX,1
    ENDM
    ENDM
; Actual code is below:
        .
  SHIFT_IT
        .

  16 bytes/32 clocks
```

Example 10 (compact)

```
                .
                .
        MOV   CX,4
SHIFT_IT: SHL   AX,1
        RCL   DX,1
        LOOP  SHIFT_IT
                .
                .
                .
                .
9 bytes/76 clocks
```

Program Transfers

The program transfer instructions allow for compact coding (CALL/RET, LOOP, etc.) and program decision making (JZ, JGE, etc.). They also take a lot of time to execute. Table 2 gives the performance characteristics for these instructions.

If coding for speed is the objective, then subroutines and unconditional jumps should be avoided. Although it makes sense to place an often-used algorithm in a subroutine, the overhead of doing so will add 17 to 67 clock cycles each time the subroutine is called. To avoid subroutines, duplicate the desired code wherever it is needed. You can do this by simply copying the routine, but a better way is to use a macro. (A macro is a single program line that represents one or several instructions. When it is expanded during assembly, the assembler replaces the macro call with the group of instructions it represents.) Macros make the program easier to read and take up less room (in the *source* file) than if the routine is duplicated everywhere it is required.

This rule applies just as well to looping and other iterative operations. Examples 9 and 10 illustrate the case of shifting a 20-bit number in DX:AX left by 1 nibble. Here the time savings is 58 percent.

You can modify this technique by executing several operations during each loop iteration if the program must perform a large number of operations and still remain reasonably compact. The result is fewer program jumps and faster execution. In Examples 11 and 12, a table of 1000 data points is normalized by multiplying each point by a variable power of 2 (contained in CL). CL=4 in these examples.

By doubling code size, we reduce execution time by about 21 percent. If the "brute force" technique of Example 9 was used, execution time would be reduced to 45,004 clock cycles but the code size would be 4003 bytes.

When writing code for minimum size, use subroutines wherever possible and avoid using code macros. However, don't forget to consider the code "overhead" (CALL & RET instruction size) required when deciding which portions of code to locate in a subroutine. In most cases, an instruction sequence that is at least 5 bytes long and appears more than once in the program should be made into a subroutine.

There is a trick you can use to save even more space

Table 2. Program transfer instruction performance

Instruction	Mode	Execution Time	Length
CALL	register indirect	16	2
	direct NEAR (FAR)	19(28)	3(5)
	memory indirect NEAR (FAR)	21+EA (37+EA)	2-4
RET	no pop NEAR (FAR)	8(18)	1
	pop NEAR (FAR)	12(17)	3
JMP	register indirect	11	2
	direct NEAR (FAR)	15	2-3(5)
	memory indirect NEAR (FAR)	18+EA (24+EA)	2-4
Conditional jump	short - relative	16 - jump taken 4 - jump not taken	2
LOOP	short - relative	17 - jump taken 5 - jump not taken	2
LOOPE (LOOPNE)	short - relative	18(19) - jump taken 6(5)-jump not taken	2
INT	type 3	52	1
	not type 3	51	2
IRET	-	24	1

when subroutines call other subroutines. If the last instruction of a subroutine (excluding the RET instruction) is a call to another subroutine, then replace that CALL instruction (and the RET instruction that follows) with a JMP to the called subroutine. This technique works because the first subroutine uses the second subroutine's RET instruction to return to the original calling program. The net result is that a byte or two of code is saved and execution time is typically reduced by 27 clock cycles. You must ensure, however, that the *calling* subroutine and *called* subroutine are the same type (NEAR or FAR). Otherwise, the return address will be the wrong length when it is popped off the stack during execution of the last subroutine's RET instruction.

If program execution depends on the result of a subroutine called by the main program, consider setting the flags in the subroutine (TEST or CMP the applicable parameter) just prior to its RET statement. This allows all calling programs to use a conditional jump instruction without testing the condition of the applicable variable, which can save quite a bit of code if the subroutine is called several times.

Decisions, Decisions!

Conditional jumps are found wherever a program must make a decision. Although these instructions cannot be entirely eliminated, we can usually optimize decision-making to hold program jumps to a minimum. In many cases, this can be done by using the data that the program execution path depends upon as a pointer to the routine to be executed.

For example, suppose we are writing a command dispatcher that must select 1 of 10 routines according to the ASCII digit contained in AL. Two ways to do this are shown in Examples 13 and 14. Example 13 is not only bigger and slower, but also suffers from addressing range restrictions imposed by conditional instructions; each routine (ONE, TWO, etc.) must be within +129 to –126 bytes of its respective jump instruction since conditional jumps contain only a one-byte displacement relative to the location of the conditional jump. We could use an intermediate jump table to get around this restriction, but this would add 30 bytes of code and increase the execution time by 15 clock cycles. Example 14 allows the command routines to be anywhere in the 64K program segment without requiring an intermediate jump table.

What if the allowable values in AL are a set of random ASCII capital letters instead of consecutive numbers? We

Example 11. "Normal" approach

```
        MOV   DX,1000           ; 1000 data points.
NORM:   SHL   WORD PTR [SI],CL  ; Normalize a data point
        INC   SI                ; Point to the next data word.
        INC   SI
        DEC   DX                ; Continue until all 1000
        JNZ   NORM              ;   points are normalized.

              62,996 clock cycles
                  13 bytes
```

Example 12. Faster Loop

```
        .
        MOV   SI,OFFSET TBL_ST  ; SI is the data pointer.
        MOV   DX,1000/4         ; 250 iterations (4 data
                                ;   points each pass).
NORM:   SHL   WORD PTR [SI],CL  ; Normalize a data point.
        INC   SI                ; Next data point.
        INC   SI
        SHL   WORD PTR [SI],CL  ; Normalize a data point.
        INC   SI                ; Next data point.
        INC   SI
        SHL   WORD PTR [SI],CL  ; Normalize a data point.
        INC   SI                ; Next data point.
        INC   SI
        SHL   WORD PTR [SI],CL  ; Normalize a data point
        INC   SI                ; Next data point.
        INC   SI
        DEC   DX                ; 4 points have been normaliz-
        JNZ   NORM              ;   ed - continue for a total
            .                   ;   250 iterations.
            .
           49,496 clock cycles          25 bytes
```

Example 13

```
        .
        .
  CMP   AL,"0"
  JE    ZERO
  CMP   AL,"1"
  JE    ONE
  CMP   AL,"2"
  JE    TWO
        .
  CMP   AL,"8"
  JE    EIGHT
  CMP   AL,"9"
  JNE   ERROR
NINE: .
        .
 20-80 (50 ave) clocks
       40 bytes
```

Example 14

```
                .
                .
                .
                .
        SUB   AL,"0"    ;Convert data to
        CMP   AL,9      ;  a jump table
        JA    ERROR     ;  pointer in DI.
        CBW
        MOV   DI,AX
        SHL   DI,1
        JMP   TABLE[DI] ; Execute routine
TABLE   DW  ZERO,ONE,TWO,THREE,FOUR
        DW  FIVE,SIX,SEVEN,EIGHT,NINE
ERROR:          .
                .
          42 clocks/35 bytes
```

can still use Example 14 with only a slight modification as shown in Example 15. This routine uses much more memory than Example 14, but it is still faster than Example 13.

If we need to use subroutines instead of jumps, Example 13 becomes very inefficient:

```
                .
                .
        CMP     AL,"0"
        JNE     NOT0
        CALL    ZERO
        JMP     DONE
NOT0:   CMP     AL,"1"
        JNE     NOT1
        CALL    ONE
        JMP     DONE
NOT1:           .
                .
ERROR:  CALL    NONE
DONE:           .
                .
```

We could improve it by putting the return address on the stack and jumping to the subroutine instead of calling it:

```
                .
        MOV     BX,OFFSET DONE
        PUSH    BX
        CMP     AL,"0"
        JE      ZERO
        CMP     AL,"1"
        JE      ONE
                .
                .
DONE:           .
```

However, we still have the same jump distance restriction as in Example 13, plus we pick up 15 extra clock cycles of execution time and 4 bytes of code. The code in Examples 14 and 15 can be used directly by replacing JMP TABLE[DI] with CALL TABLE[DI]. This adds only three clock cycles to the execution time without adding any bytes of program code.

The technique discussed above is effective for selecting a single execution path from many choices. When the number of possible outcomes is less than eight, or one outcome has a much higher probability of occurring than the others, the direct approach shown in Example 13 is the fastest and most compact.

Another example of decision-making that vividly illustrates the relationship between code size and execution speed follows. This routine counts the number of leading zeros in the piece of data contained in the AX register; such a routine might be found in data scaling and floating point math programs.

Table 3. Speed compared to program size

relative speed	size (bytes)	average execution time
slow & compact	14	191
faster & bigger	56	73
fastest & biggest	65545	21

```
; Compact Leading Zero Counter
        MOV     CX,16     ;AX=0 => 367 clocks
CHKBIT: SHL     AX,1      ;AX>7FFFH => 29 clocks
        JC      FOUND
        LOOP    CHKBIT
FOUND:  SUB     CL,16     ;Return count
        NEG     CX        ; in CL.
```

Although this is a tight piece of code that looks fast, a

Example 15

```
        .
        SUB   AL,"A"          ; Remove ASCII bias
        CMP   AL,"T"-"A"      ; Highest character allowed
        JA    ERROR           ;   is letter "T".
        CBW                   ; Convert the data to a
        SHL   AX,1            ;   routine pointer.
        MOV   DI,AX
        JMP   TABLE[DI]
ERROR:  .
        .
    ; Command routine addresses. Legal chars. are 'ACDGHJPRST'.
      TABLE  DW   RA,ERROR,RC,RD,ERROR,ERROR,RG,RH,ERROR,RJ
             DW   5 DUP (ERROR)
             DW   RP,ERROR,RR,RS,RT

                    42 clocks/55 bytes
```

Example 16

```
    ; "FAST" Leading Zero Counter
        MOV  CL,8       ;At least 8 leading zeros
        OR   AH,AH      ; if most significant
        JZ   GT8        ; byte is 0.
        XOR  CL,CL
        MOV  AL,AH
GT8:    CMP  AL,8       ;>4 zeros in LS byte?
        JB   GT4
        CMP  AL,20H     ;If not, >2 zeros?
        JB   GT2
        CMP  AL,40H     ;If not, >1 zero?
        JB   INC2       ;If >1, must be 2 leading 0s.
        CMP  AL,80H     ;If not, then there are
        JAE  INC0       ;  no leading 0s or . . .
        JMP  SHORT INC1 ;  only 1 leading zero in LSB.
GT2:    CMP  AL,10H     ;If >2, then there are either
        JAE  INC3       ;  3 leading 0s or . . .
        JMP  SHORT INC4 ;  4 leading 0s in the LSB.
GT4:    CMP  AL,2       ;If >4, are there >6 zeros?
        JB   GT7
        CMP  AL,4       ;If not >6, then there are
        JAE  INC5       ;  5 leading 0s or . . .
        JMP  SHORT INC6 ;  6 leading 0s in the LSB.
GT7:    CMP  AL,1       ;If >6, then there are either
        JAE  INC7       ;  7 leading 0s or . . .
        INC  CX         ;  8 leading 0s in the LSB.
INC7:   INC  CX
INC6:   INC  CX
INC5:   INC  CX
INC4:   INC  CX
INC3:   INC  CX         ;AX >7FFFH => 60 clocks
INC2:   INC  CX         ;AX = 0 => 87 clocks
INC1:   INC  CX
INC0:     .
          .
```

Example 17

```
;"FASTEST" (and largest) Leading Zero Counter

LU_TBLE     SEGMENT PARA
    TABLE   DB 16,15,14,14, 4 DUP (13)
            DB 8 DUP (12), 16 DUP (11), 32 DUP (10)
            DB 64 DUP (9), 128 DUP (8), 256 DUP (7)
            DB 512 DUP (6), 1024 DUP (5), 2048 DUP (4)
            DB 4096 DUP (3), 8192 DUP (2), 16384 DUP (1)
            DB 32768 DUP (0)
LU_TBLE     ENDS

CODE        SEGMENT BYTE PUBLIC
            ASSUME DS:DATA, CS:CODE
        .
    MOV   BX,LU_TBLE   ;Use data as a pointer
    MOV   DS,BX        ;  into a leading zero
    ADD   SI,AX        ;  lookup table.
    MOV   CL,[SI]      ;
        .
CODE        ENDS
```

THE ULTIMATE SHELL ... TYING IT ALL TOGETHER

UNIFYING Custom Menuing & Hard Disk Management

1dir Plus -- The Key To A Truly *PERSONALIZED* DOS Environment

Designed for the *power user* who wants to get the most from their PC, **1dir Plus** provides the most powerful, flexible DOS environment available.

It is the leading modular DOS shell system, bringing together the best of *custom menuing* and *hard disk management* in one *integrated package.*

It provides all the tools to configure systems to meet the most demanding needs, whether for yourself or clients.

Instant Turn Key Systems, Just Add **1dir Plus**

If you are a *consultant, system integrator,* or *VAR* selling customized systems, networks, or vertical market packages, **1dir Plus** is for *YOU.*

The **1dir Plus Professional Developer's System**

***1dir Plus *Multi-User / Security *Muscle Language**

The **Professional Developer's System** is an add-on product to **1dir Plus** that further expands programmability by allowing even more sophisticated and automated commands to be built.

The **Multi-User / Security System** Module provides the key to creating and implementing multi-user configurations. It provides *logon* and *directory* access security, creates customized *Tree* displays, and allows *"Super"* users access to "everything" on the system.

With the **Muscle Language** Module you use an *editor* to create commands, then compile them into the *user / menu file* of your choice.

The Unmatched Power and Flexibility of **1dir Plus**

* Easy To Use Point & Shoot Operation.
* Comprehensive File Management Commands: *Copy / Locate / Erase / Rename ...*
* Build Custom Commands with Help System.
* 8 convenient Alternative Screen Display options.
* Multi-Mode Editor built in.
* Multi-Level Security / Password protection.
* Automatically Unload and Reload when running large programs.
* And More -- *(Personalities, Global File Operations, EMS mode ...)*

1dir Plus is the LAN solution.

It provides virtually unlimited options for designing and controlling networks. Compatible systems include Novell, Token Ring, 3COM, Alloy, ... and other systems.

The Most **Power & The Best Value**

And now there are even more benefits for *power users* who take advantage of **Bourbaki's "SPECIAL INTRODUCTORY OFFER"** on **1dir Plus, The Professional Developer's System,** and **S.F.E. Utilities.** CALL or WRITE **Bourbaki** for details.

Suggested Retail Prices

1dir Plus	**$95.00**
The Professional Developer's System	**$75.00**
S.F.E. Utilities	**$69.00***

* **S.F.E. Utilities** is *$49.00* for registered **1dir Plus** users. Network, Site, Authorized Consultant, and OEM licensing is available from **Bourbaki, Inc.** Dealer and consultant inquiries are invited.

Bourbaki

Software

Bourbaki, Inc.
P.O. Box 2867
Boise, ID 83701
(208) 342-5849

Call the **Bourbaki** *ORDER LINE:* **1-800-BUY-1DIR / 1-800-289-1347**

BOURBAKI'S CONVENIENT FAX LINE: 1-208-342-5823

CIRCLE 70 ON READER SERVICE CARD.

jump is required for every leading zero. Assuming equal probability of any number of leading zeros, the average execution time of this program is 191 clock cycles.

If we use a binary-search type of algorithm, we find we only need to make four decisions (program jumps) at most ($2^4 = 16$ bits). An example of such a routine is shown in Example 16. Again assuming equal probability of any number of leading zeros, this example is more than twice as fast, on average, but the code is four times as large.

Finally, if speed is of utmost concern and size is not a consideration, we can use the "brute force" approach shown in Example 17. The amount of memory used by this example may seem to make it a ridiculous approach, but it is the fastest and can be readily justified when minimum execution time is required. After all, memory is cheap!

Although this is a specific case, the leading zero counter program is a good example of how speed and size can be traded off when a multi-outcome decision must be made. The performance characteristics of the three examples are shown in Table 3 for comparison.

In most decision matrices, execution time varies according to the value of the input data. Where the data falls within a specific range most of the time, the code should be written to minimize the number of program jumps executed when the data is within the "expected" range.

Miscellaneous Tricks and Techniques

When using register operands, there are several common tasks that can be accomplished in a few different ways. With a little thought, these tasks can be streamlined to cut clock cycles.

XORing a register with itself is one clock cycle faster, and one byte shorter (for 16-bit registers), than MOVing 0 into it. If you need to clear several registers and/or memory locations, clear a register and then MOV it into the other operands.

How about testing a register for 0? Normally you do this by comparing the register with 0. ORing a register with itself accomplishes the same objective while saving one clock cycle and a byte of code. If you need to test a memory operand for 0 and have a cleared register, CMPing the register with memory is one clock faster and saves a byte over directly comparing the memory operand with 0.

Don't forget that the stack is really memory. If you need to discard some values off the stack, manipulate the stack pointer (ADD SP,2) in lieu of POPing numbers off the stack. For one discarded value, this method will save four clocks; for two values, 12 clock cycles will be saved. In situations where code size is restricted and only one or two values must be discarded, use POP instructions.

If you need to add or subtract 2 to or from a 16-bit register, use two INC (DEC) instructions instead of an ADD (SUB) instruction. The execution time is the same but you will save a byte of code.

Prefetch Queue Operation

Unlike previous generation microprocessors (Z80, 8080, 6800, etc.), the 8086 family CPU architecture supports instruction pipelining. This is made possible by the use of a Bus Interface Unit (BIU) and an Execution Unit (EU) in the CPU that operate separately and relatively independ-

Performance Programming Rules of Thumb

To save execution time:

1. Keep often-used data and often-manipulated data in registers for as long as possible.
2. Minimize the use of memory variables as destination operands in instructions to save seven clock cycles per instruction.
3. On 8086 systems, align all 16- and 32-bit memory operands on even address boundaries.
4. Convert byte-string MOVE and STORE operations to word-string operations.
5. Use individual MOVS (STOS) instructions instead of a REP'ed string instruction for strings of 14 elements or less.
6. Use the shift and rotate instructions for multiplication and division (instead of MUL and DIV) when one of the operands is a constant.
7. Use macros instead of subroutines.
8. Eliminate unconditional jump instructions in looping routines by replacing each required iteration with a macro. When a large number of iterations is required, perform multiple operations in each loop iteration.
9. Adjust SP (instead of using the POP instruction) to discard values off the stack.
10. To save a few extra clock cycles, intermix fast and slow executing instructions when possible.

To save code size:

1. Group often-used memory variables into a 256-byte block and use indexed (offset) addressing mode to access the data.
2. Use AAM and AAD instructions when dividing and multiplying by 10.
3. Avoid macros. Replace all sections of code that appear more than once in the program (and longer than 4 bytes) with a subroutine.
4. Use the SHORT assembler directive with the JMP instruction when forward referenced labels are close (within 129 bytes) to the JMP instruction.
5. Use individual MOVS (STOS) instructions instead of a REP'ed string instruction for strings of four elements or less.

To save execution time and minimize program size:

1. Minimize forward references by placing all data and constant declarations at the beginning of the source code.
2. In large programs, distribute memory variables among the smallest number of 64K segments possible. Set the segment registers to take advantage of the default segments used by the instructions that access the data.
3. Command dispatchers and simple-decision matrices should use table pointers (derived from the input data) instead of several conditional jump instructions when the number of possible outcomes is eight or more and all outcomes have a fairly equal probability of occurrence.

ently of each other. The BIU fetches instructions and data from memory while the EU executes the instructions fetched by the BIU. Combined with an internal prefetch queue, this architecture enhances program speed by allowing instruction fetches and instruction execution to occur in parallel.

During normal operation, the BIU fetches instructions from memory and stores them in a 6-byte Instruction Prefetch queue (4-byte on the 8088) while the EU executes these instructions. When the EU needs a memory operand to execute an instruction, e.g., MOV AL,[BX][DI], the bus interface unit fetches the required memory operand. Under ideal conditions, the BIU will maintain at least one unexecuted instruction in the prefetch queue while responding to all EU-generated memory access requests. This causes the CPU to run at maximum efficiency; the EU continuously executes instructions and the CPU bus is busy most of the time.

Any instructions that alter program flow will reduce CPU throughput because after the program transfer occurs, all instructions are flushed from the prefetch queue.

Any instructions that alter program flow (prevent in-line code execution) will reduce CPU throughput (instructions executed per second) because after the program transfer occurs, all instructions are flushed from the prefetch queue. After the Instruction Pointer is loaded with a new value, the EU must wait for the BIU to fetch the next instruction before the EU can continue program execution. This is why the control transfer instructions take so long to execute. If program jumps occur too often, the BIU wastes a lot of time fetching instructions that are never executed. Therefore, to enhance program speed, minimize use of software interrupts, subroutines, and program jumps as discussed earlier.

If the program contains many fast executing instructions in a row, the EU will "empty" the prefetch queue faster than the BIU can "fill" it. This will result in short periods of time where the EU is idle while waiting for the BIU to fetch instructions. In 8086 systems with no wait-state, 16-bit memory, this problem occurs if consecutive instructions execute faster than two clock cycles per opcode byte because one instruction fetch cycle (16-bit memory read) takes four clock cycles to execute. The situation is aggravated further in systems that are already bus bandwidth limited much of the time; computers that use large amounts of slow memory (one or more wait states) and/or dynamic RAM fall into this category.

The bottom line here is that actual execution time may differ from calculated execution time by several percent. The amount of deviation depends on the number of control-transfer instructions and the sequence of instructions in the program.

Example 18 illustrates this point. It calculates the average of two 40-bit numbers located in SOURCE and DEST. As shown in Table 4, the actual execution time is 11 percent greater than the calculated execution time. By simply rearranging the program so that the fast (RCR) instructions are interleaved with the slower (MOV DEST[SI+6],DI, etc.) instructions, the "excess" execution time is reduced by 4 percent. The performance improvement is admittedly small, but it may be significant in severely time-constrained applications.

I am certainly not suggesting you conduct a prefetch queue performance analysis whenever you write code. Just remember its impact on performance if you need to squeeze a few more clock cycles out of your program. And if you are writing a routine that must provide a precise time delay, you should *measure* its execution time rather than relying on a timing calculation. Section 4.3 of *The 8086 Family User's Manual* contains a sample detailed analysis of prefetch queue operation.

Memory Organization and Segment Register Management

The 8086-family CPU uses segment registers to define separate code and data portions (called "segments") of memory. These segments may only be 64K (65,536) bytes long because the memory addresses specified in instructions and registers are restricted to 16-bit numbers. Whenever the CPU accesses system memory, it combines the 16-bit address (specified in the instruction or instruction pointer during memory operand and instruction fetches) With one of the segment registers to generate a 20-bit *physical* address. This means the Data Segment (DS) and Extra Segment (ES) registers must be initialized to point to data memory before any memory operations may be performed. For DOS .COM files, the operating system takes care of this task during loading and the program should not normally manipulate the segment registers.

All DOS .EXE program files require the program to set the DS and ES registers to point to data memory. Programs that use less than 64K of *data* memory can get away with simply setting these registers once at the beginning of the program.

Memory variables in programs requiring more than 64K of data must be distributed between two or more segments (64K maximum per segment). This is done by declaring the variables in different segments (SEGMENT and ENDS assembler directives) in the source program. The manner in which this is done affects performance because a combination of segment register manipulation and segment register override techniques are required to access the data.

Segment register operations can become fairly cumbersome because only a limited number of instructions (LES, LDS, PUSH, POP, and MOV) can directly manipulate the registers. Segment register contents and immediate operands cannot be MOVed directly into a segment register. Thus, a few instructions are required just to set a couple of segment registers as shown in Example 19. While the code may seem trivial, the resulting performance degradation becomes significant when the segment registers need to be changed frequently to access data in different segments. If memory operands are divided into the smallest number of *segments* possible, program speed will be improved and program size will be reduced because less segment register manipulation will be required.

Most of the data manipulation instructions and addressing modes use the DS register as the default segment for calculating the 20-bit physical address. You can use the segment override prefix with most instructions to change this when convenient. Table 5 illustrates which segment registers may be used in the various types of memory accesses and addressing modes.

You should keep segment register usage in mind when writing code. For example, although it seems convenient to place a program jump table right after the code that

Example 18

```
CODE    SEGMENT PARA PUBLIC
        ASSUME     CS:CODE,DS:CODE

    SOURCE   DW     5 DUP (?)
    DEST     DW     5 DUP (?)

        .
      MOV   SI,OFFSET SOURCE   ; Initialize data pointer.
      LODSW                    ; Move SOURCE to registers
      MOV   BX,[SI]            ;   for fastest execution.
      MOV   CX,[SI+2]
      MOV   DX,[SI+4]
      MOV   DI,[SI+6]
      ADD   AX,DEST[SI-2]      ; Sum SOURCE and DEST in
      ADC   BX,DEST[SI]        ;   registers.
      ADC   CX,DEST[SI+2]
      ADC   DX,DEST[SI+4]
      ADC   DI,DEST[SI+6]
      RCR   DI,1               ; Divide the sum by two.
      RCR   DX,1
      RCR   CX,1
      RCR   BX,1
      RCR   AX,1
      MOV   DEST[SI+6],DI      ; Save the results in DEST.
      MOV   DEST[SI+4],DX
      MOV   DEST[SI+2],CX
      MOV   DEST[SI],BX
      MOV   DEST[SI-2],AX

CODE    ENDS
        END     START
```

Example 19

```
Initialize segments

        .
MOV   AX, OFFSET DATA_SEG
MOV   ES,AX
MOV   DS,AX
        .
        .
 7 bytes/8 clocks
```

```
Swap source and destination
string data segments

        .
      MOV   AX,ES
      MOV   BX,DS
      MOV   DS,AX
      MOV   ES,BX
        .
        .
  8 bytes/8 clocks
```

Table 4. Prefetch queue operation example results

Calculated execution time	Actual Execution Time (clock cycles)			
	Example 17		Modified example 17	
	0 wait state	1 wait state	0 wait state	1 wait state
262	291	321	281	321

Table 5. Matching segment registers and accesses

Memory Access	Default Segment register	Alternate Segment registers*
Instruction fetch	CS	none
Stack operation	SS	none
String source operand	DS	CS, ES, SS
String destination operand	ES	none
Memory operand (addressing modes that use BP)	SS	CS, DS, ES
Memory operand (all other addressing modes)	DS	CS, ES, SS

* Altered by using segment override prefix.

CHARTER SUBSCRIPTION INVITATION

Here At Last!
The Authoritative Source of Database Application and Product Information

Inside DBMS Magazine

DBMS Magazine offers in-depth articles examining the capabilities of programmable databases as tools for solving real business problems. Authoritative features show how a specific product is used to solve a specific problem. Sample some of the feature articles to be presented:

- 50 Nasty dBase Bugs, and How to Squash Them
- Has Ashton-Tate Finally Designed a Decent, Multiuser Database?
- Why 386 Databases Can be Either the Fastest, or a Waste of Money
- Is FoxBase+ dFastest Around?
- Why Microsoft and Ashton-Tate teamed up on SQL: Server – What this Ambitious Product Will and Will Not do for You
- R:Base Puts OS/2 to Work
- Understanding Paradox's PAL
- Can Oracle Squeeze Into a PC?

...with regular columns written by experts, providing tips on developing and modifying databases, such as:

- Putting dBase IV's Arrays to Work
- Cross tabulations in R:Base
- Sharing Files Between 1-2-3 and dBase
- Using Clipper's Valid Statement
- Tips on Creating Reports in Paradox

Who Should Read DBMS Magazine?

Professional developers, MIS/DP departments, consultants, VARs and serious users will find DBMS Magazine to be the source of important information for serious users of microcomputer database software.

Charter Offer and Benefits with No-Risk Guarantee

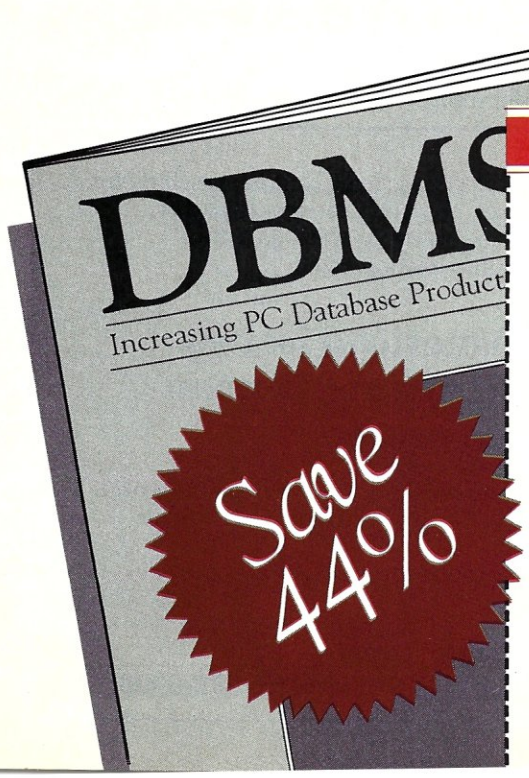
For a limited time DBMS Magazine will offer Charter Subscriptions with substantial cash savings.

You will also be entitled to continued savings — on all renewals and on any gift subscriptions you may order.

If at any time you decide DBMS is not for you, simply cancel. We will send a full refund for all unmailed issues.

Order Today!
Send your order form to:

DBMS Magazine
P.O. Box 3713
Escondido, CA
92025



CHARTER SUBSCRIPTION ORDER FORM

☐ **Yes,** please send my first issue and enter my Charter Subscription to DBMS Magazine. My full year's subscription (11 additional issues) is only $19.97. I save 44% off the newsstand rate.

SEND NO MONEY!
We will bill you later, after you receive your first issue. GUARANTEE! You may cancel at any time for any reason and receive a full refund on all unmailed issues.

MAIL TO: **DBMS Magazine, P.O. Box 3713, Escondido, CA 92025**

NAME

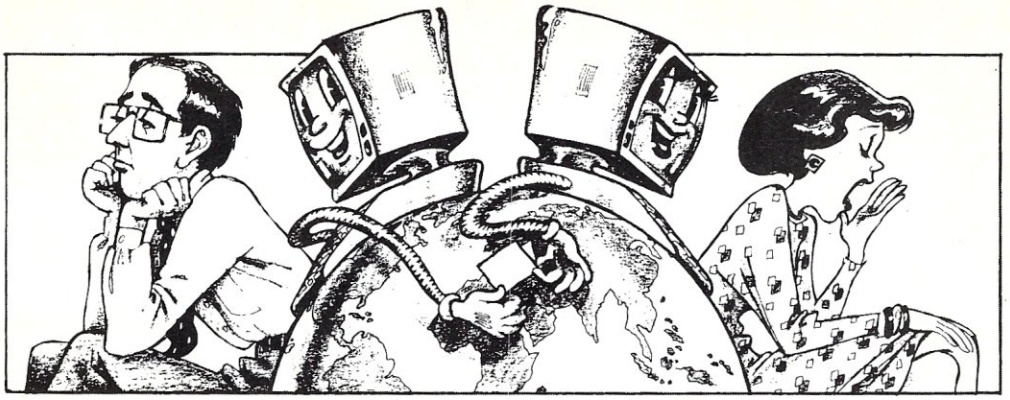
COMPANY

ADDRESS

CITY/STATE/ZIP

Allow 4–6 weeks for delivery of first issue.

Newsstand rate (1year): $35.40



Ten Reasons Not to Use PeaceNet

1. I don't like working with others

PeaceNet is a computer network and communication system for people who believe that global planning and cooperation are necessary to reverse a trillion-dollar-per-year arms race; it is linking users throughout the United States and in over 70 other countries.

2. I've got all the information I'll ever need

PeaceNet is for those who appreciate that information is always growing and changing; its bulletin boards, conferences, and databases provide information about everything from Central America to Star Wars.

3. I love playing phone tag

PeaceNet's electronic mail system renders those endless conversations with secretaries and answering machines obsolete.

4. I don't know how to use my computer

PeaceNet helps novices with simple, entertaining manuals and round-the-clock staff for answering their questions.

5. I enjoy copying, labeling, and stamping letters

PeaceNet enables you to send messages to hundreds of other users with one simple command.

6. I've got plenty of money to waste on postage and phone bills

PeaceNet is for people who want to save money; it lets you send documents across the world faster than Federal Express™ for pennies per page.

7. I don't mind getting action alerts a week late

PeaceNet does mind and can help your organization send out time-urgent alerts instantly.

8. I don't have the right kind of computer equipment

PeaceNet is available to anyone with a computer terminal and a modem.

9. An effective peace movement isn't worth 50 cents a day

PeaceNet users disagree.

10. It's all hopeless, anyway

Then why read this magazine when Modern Wrestling would suffice?

Use PeaceNet's electronic mail, conferences, bulletin boards, databases, and Telex services to gather, organize and share valuable up-to-date and comprehensive information about:

Peace and anti-nuclear efforts, Central American issues, environmental protection, anti-apartheid efforts, human rights, Native American issues, and much more.

PeaceNet is the largest and most rapidly growing on-line community of progressives anywhere in the world. And by using any home computer and a modem it is only a local phone call away. Call or write today for more information.

PeaceNet:

3228 Sacramento Street
San Francisco, CA 94115 (415) 923-0900

uses it (in the code segment), the instructions that access the table require a segment register override prefix if the DS register is not pointing to the code segment. (This is the normal case.) This segment override prefix adds a byte of code and a couple of clock cycles. If you have used the ASSUME assembler directive correctly, this will go unnoticed because the assembler will automatically generate the segment override prefix for you. The prefix can be eliminated (making the code more compact and faster) by moving the jump table into a segment that is pointed to by the DS register.

In programs containing many data segments, the memory operands should be distributed to take advantage of the default segment registers that the various instructions use. And, if all data can be contained in two or three segments, you can save execution time and minimize code size by assigning a segment register to each segment and using the segment override prefix as necessary. For example, if a spelling checker program uses a 32K string variable area for text manipulation, a 32K string constant area for string comparison operations, and 4K of miscellaneous memory variable area, the memory variable areas should be placed in one segment and the string constant area should be placed in a different segment. The DS and ES registers would be set to point to the variable and constant data segments respectively. This would minimize segment register manipulation and segment override prefix usage because most data manipulation instructions use the DS register while the string comparison instructions (CMPS and SCAS) use the ES register.

Optimizing assembly language programs . . . is not difficult, but it requires strict attention to detail and an in-depth knowledge of the instruction set and CPU architecture.

Programming Practices

Despite your best efforts, there are certain occasions when the assembler will generate more code than you intended. This can be prevented by understanding how the assembler operates and by following good programming practice.

Most assemblers are two-pass assemblers; they scan the code twice before the machine code is completely generated. During the first pass, the assembler allocates space for the opcodes and data, assigns values to labels and constants, etc. The assembler allocates the maximum amount of room for instructions that can contain either a *word* or *byte displacement* (indexed memory operands with a displacement and short JMP instructions) if the size of the displacement is undefined due to a *forward reference* (a label or operand definition that appears after it is used in an instruction). Additionally, immediate operands in 16-bit ADD, ADC, SUB, SBB, and CMP instructions are allocated one word of memory if the value of the operand is defined *after* the instruction. (A 16-bit immediate operand in the range +127 to –128 can be encoded as a byte to reduce code size. The byte is then sign-extended to its 16-bit equivalent during program execution.)

During the second pass, the assembler fills in the opcodes with the displacements and immediate operands that were assigned upon completion of the first pass. If any displacements/immediate data can be reduced from 16-bit to 8-bit numbers, the assembler will modify the opcodes of the affected instructions and insert NOP instructions in the unused portions of opcode area allocated during the first past. This adds one byte of code and three clock cycles per NOP instruction.

You can avoid this problem by putting all constant definitions (EQU statements) and data variable allocation directives at the top of the program. Short unconditional jumps to forward-referenced locations within 129 bytes of the JMP instruction can be forced to assemble in two bytes (instead of three) by using the SHORT pseudo-op. This causes the assembler to allocate only one byte for the displacement during the first pass. (Refer to *The Macro Assembler Reference Manual* for details.)

To prevent performance degradations due to memory operand alignment in 8086 systems, be sure all word operands are located at even addresses. Do this by using word- or paragraph-aligned data segments (defined in the SEGMENT assembler directive) and placing all word-data variables at the beginning of the segment. If you must place word variables in the middle or end of the data segment, group the variables together and use the EVEN directive to restore operand alignment.

Conclusion

Optimizing assembly language programs for speed and size is not difficult, but it requires strict attention to detail and an in-depth knowledge of the instruction set and CPU architecture. This article should provide some insight into methods of exploiting the instruction characteristics and architecture of the 8086 family CPU to achieve maximum performance.

Although the performance of the 80286, 80386, V20, and V30 CPUs was not covered, all the techniques described still apply. In general, instructions execute in fewer clock cycles on these processors than on the 8086 and 8088. □

References

Aho, A. V. & Ullman, J. D. *Principles of Compiler Design*, Reading, MA: Addison-Wesley Publishing Co., 1977.

Intel Corp., *The 8086 User's Manual*, Santa Clara, CA: Intel, 1980.

Intel Corp., *MCS-86 Macro Assembly Language Reference Manual*, Santa Clara, CA: Intel, 1978.

Intel Corp., *MCS-86 User's Manual*, Santa Clara, CA: Intel, 1978.

Microsoft Corp., *Macro Assembler Reference Manual*, Bellevue, WA: Microsoft, 1984.

Morse, S. P. *The 8086 Primer*, Rochelle Park, NJ: Hayden, 1980.

Titus, C. A., et al. *16 Bit Microprocessors*, Indianapolis, IN: Howard W. Sams, 1981.

Technical Brief (continued from page 19)

OS/2 has effectively two command-line interfaces, both of which can be replaced like COMMAND.COM. One interface is for the DOS compatibility box that runs in real mode while the other interface is for protected mode. There may be only one of the former, the DOS command processor, and any number of the latter, the OS/2 command processor, due to the limitations of OS/2. Multiple command processors (command-line interface programs) run concurrently under OS/2. The two different types of command processors can cause a number of problems. The DOS command processor is essentially compatible with DOS's COMMAND.COM.

This allows most applications and batch files to run under OS/2. However, there are a number of commands that are not replicated in the OS/2 command processor and an equal number that the OS/2 command processor supports that its counterpart does not. This can be quite confusing as well as frustrating. It is also one reason why the DOS compatibility box will be used sparingly.

One thing that may keep people away from these command processors is the menu front end supplied with OS/2. This menu system is used primarily for starting applications which, for the most part, is what the command-line programs do.

The UNIX command processor, like DOS and OS/2, is also replaceable. There are a number of UNIX *shells* (the same terminology is often used with DOS and OS/2) that have become popular like the Cshell or *csh*, and the Bourne shell. The UNIX shells perform the same functions as the DOS and OS/2 command processors: execute programs, built-in command files, and batch files. There is no comparable standard to the OS/2 menu front end for UNIX.

There are differences between the UNIX and OS/2 command processors, such as differences in built-in commands and utility programs. The built-in DOS directory function, DIR, is functionally identical to the UNIX *ls* program. Batch files have different file extensions, and the file name sizes are different between UNIX and OS/2.

Both the OS/2 and UNIX command processors provide support for the underlying multitasking system by allowing multiple programs to be started at the same time as well as linking programs together via pipes. Pipes appeared first in UNIX and are found in DOS, although DOS only simulates pipes using files because DOS cannot run multiple programs simultaneously. An example of a command line using pipes would be:

```
DIR | FILTER | MORE
```

The command is actually syntactically correct for both OS/2 and UNIX (provided the three files exist on the UNIX system). The command processor starts off three programs with the output of one being supplied to the standard input file of the next program. Programs that process information in this fashion are called *filters*. Combining batch files, pipes, and other multitasking features of the command processors is often all that is necessary to build or support an application. Power users tend to take advantage of these features, but most users of OS/2 and UNIX do not even know they exist let alone their implications.

Programming Interface

OS/2 and UNIX provide a programming interface that is very similar but differs from DOS. Functionally, all three provide basic operating system functions including file manipulation and memory allocation. Both OS/2 and UNIX, however, provide more complex process control as well as interprocess communication functions.

DOS is designed specifically for the Intel 8086 microprocessor family and uses an .INT instruction to access low-level and operating-system functions. Parameters are passed in registers and results are sometimes returned in the flag register. DOS uses a linking loader that can resolve memory references within the program itself. DOS provides a conventional set of file, memory, and process functions. Low-level peripheral functions are accessed using BIOS interrupts.

OS/2 and UNIX also use a linking loader, but access to operating system functions is done through calls instead of .INT instructions. All parameters are passed on the stack and results are returned in a register. This makes the operating systems more compatible with high-level languages. It also shows the influence of C, which was used to write a good deal of both operating systems. This also makes it easier to interface high-level language applications. The use of calls for all system functions has an additional advantage because there is no difference between an operating system function call and a call to a local function. This feature is how the operating systems can be transparently extended to applications. The extensions are called *dynamic link libraries* in OS/2 and *shared libraries* in UNIX. In fact, most of the operating system features are implemented using these mechanisms.

Both OS/2 and UNIX are designed to operate in a protected environment. This prevents an application from directly accessing peripheral functions using .INT instructions or directly accessing peripheral ports. The protected environment means all support functions are provided by the operating system or its extensions.

OS/2 and UNIX both provide file-access functions using handles. The file system is hierarchical in both cases but OS/2, like DOS, has multiple roots, one for each disk drive. Both operating systems provide access to devices using logical file names.

Memory allocation functions with the ability to share memory between tasks are provided by both operating systems. However, tasks within OS/2 differ slightly from UNIX. OS/2 calls a *process* a *logical entity* that executes one or more *threads* that share resources, like files and memory, allocated to the process. A thread has an execution state and priority. A process is part of a session, which is a group of related processes. The idea is to keep the overhead associated with threads low so they can be quickly created and terminated to make multitasking more suitable for general use.

UNIX provides a similar multitasking environment. A UNIX process is more like an OS/2 process with a single thread. UNIX provides a *fork* function that splits a UNIX process into two separate execution units. The UNIX forked processes share the same code and data, but the overhead per item tends to be higher than with OS/2. Both operating systems provide a way to start up a completely new and independent program that can be under the control of the parent program

or independent of the parent.

OS/2 and UNIX provide various interprocess communication facilities. Both provide unnamed pipes, that are sequential files that one process can read and another can write to. The OS/2 LAN Manager provides named pipes which is a standard function provided by UNIX. This makes UNIX a bit nicer to program in than OS/2. UNIX also has an extension being supported by AT&T called STREAMS, which is an extension on pipes that provides a number of features including *message-oriented queues* and *remote procedure calls* (RPC). The OS/2 LAN Manager can be used to build RPC support, but there is no standard similar to STREAMS. However, OS/2 does provide a queue mechanism unrelated to the file system that can provide features similar to STREAMS.

OS/2 and UNIX provide semaphores and asynchronous signals that are essentially identical. *Shared memory* can also be used for interprocess communication. OS/2 inherently uses the 80286-segmented architecture for sharing memory. UNIX-shared memory differs, depending upon the microprocessor on which it runs.

So far, OS/2 and UNIX seem a lot alike. In fact, keeping to a high-level language such as C and the system functions already addressed will allow many applications to be easily ported between these two operating systems. The areas where the two operating systems differ are in their handling of peripherals. OS/2 has the edge because of the limited number of different platform types. Essentially, each machine must be an 80286 or an 80386 running in the 80286 compatibility mode, have a memory-mapped display, and have a keyboard capable of returning scan codes for each key movement. A mouse has a standard but optional interface. UNIX, on the other hand, runs on a multiplicity of processors with more options for the display and keyboard. This difference reveals itself in the peripheral control provided by the operating system.

OS/2 has an extensive video display support for character-mode operations and access to graphic-mode displays but not at the same level as provided by the OS/2 Presentation Manager. Keyboard and mouse support has a similar functional complement. The main advantage of OS/2 is that a programmer can count on these functions and their associated support. UNIX systems usually come with libraries to support displays and input devices but applications often must be programmed to the lowest common denominator because you cannot count on having a memory-mapped display. The *Curses* library is one UNIX library that provides display and keyboard support. It supports a wide variety of terminals and displays by using a configuration file that describes the features supported by the terminal. An application does not have to worry about how functions are implemented.

OS/2 and UNIX each have their share of unique functions. These range from handling a disk directly and manipulating the file system at a low level to handling peripherals such as printers and serial ports. Applications that deal with these functions tend to be very system-specific.

Graphics Programming Interface

Graphics tends to throw a monkey wrench into compatibility. Windows from Microsoft and GEM from Digital Research are two graphics platforms for DOS. They are also several user interfaces in addition to these two graphical programming environments. OS/2 Version 1.1 will have an extended version of Windows called the OS/2 Presentation Manager. Like Windows, the Presentation Manager will be both a graphics interface for programs and a user interface. The complexity and number of functions in these graphics systems are actually greater than those provided by the operating system. However, these environments provide a standard way of using graphics and integrating applications. In general, Windows is the dominant graphic environment for DOS, and Presentation Manager will dominate for OS/2. The visual similarities are striking.

The UNIX user interface is both better and worse than OS/2. X-Windows is the graphics environment standard. There are others but X-Windows seems to be the dominant standard. Unfortunately, it is a graphics programming environment and not a user interface. There are a number of user interface contenders based upon X-Windows, including the recently announced *Open Look* system. Open Look provides a way to define windows and dialog boxes like those found on the Apple Macintosh, Windows, and the Presentation Manager.

The environments differ enough to make most applications environment-specific. It is possible to modularize programs to isolate the differences and use libraries to let a single program run under each of these graphic environments. However, the job is neither easy nor straightforward when it comes to implementation.

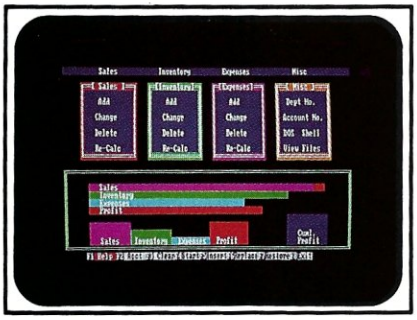
Porting DOS Applications

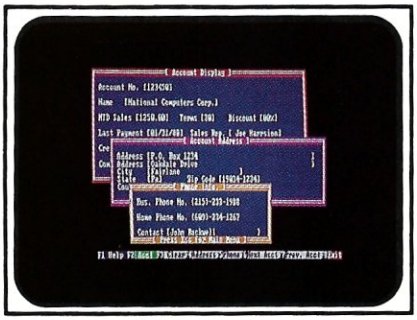
OS/2 and UNIX promise to provide platforms to which many companies will want to port existing DOS applications. IBM recently displayed more than 300 OS/2 applications ported from DOS at the Comdex show. It is also a way to add multitasking features and access to multiuser environments that can enhance the functionality of many applications. Porting applications from DOS to either OS/2 or UNIX is possible and the ease with which it can be done depends on the application, how it is built, what facilities it uses, and what language it is written in.

The language is the easiest to discuss. COBOL and FORTRAN tend to be the best for porting applications, not because they are great languages but because they have a standard base. Of course, using nonstandard features of these languages makes it just as hard to port an application written in another language. C is a favorite for porting applications because of the compatibility among compilers, but the ability to get close to the computer can also cause problems in porting applications. All bets are off using assembler, but going from DOS to OS/2 with an assembler application tends to be easier than going from DOS to UNIX unless the target version of UNIX is running on an 80286 or possibly an 80386. However, problems tend to occur when dealing with memory segments and segment addressing.

The facilities within the implementation language are one aspect of porting an application. System-specific facilities are another aspect. DOS-to-OS/2 application migration tends to be easier in this area because the peripherals tend to be the same and so is the interface. UNIX usually falls short because the types of peripherals, like the terminals, tend to be

Don't fall flat on your Ashton.





Power up Q-PRO4™ Version 5.0. Get all the reliable, 'bug-free' features you need NOW!

The Most Powerful Screen Handler You've Ever Seen.

Q-PRO 4 offers all the features, the power, and the reliability you need. And it's available now!

- What you see is what you get (WYSIWYG) no coding
- Fill-in-the-blanks screens
- Field editing with attributes
- Entry fields addressable as variables in the program
- Yes/no fields
- Multiple choice fields
- Entry fields symbolically addressable
- Programmable screen field types
- Picture data editing
- 16 color palette
- Active field color
- Painted background

Fully Featured Windows (optional)

- 100 windows per screen
- Multiple windows displayed simultaneously
- Frame and title
- Any size, any location
- Create WYSIWYG with included editor
- 3 Types: data entry, scrolling help, and data display

No Limits

Unlimited files open simultaneously (we have overcome the DOS limits).

- 2,000,000,000 bytes per file
- Unlimited record size
- Unlimited number of fields on screen
- Unlimited memory arrays
- Unlimited size of one and two dimensioned memory arrays

Extraordinary Data Files

- Data dictionary
- Nine indexes per data file
- All indexes in one file
- Ascending and descending indexes
- B+ tree (CBTREE) state-of-the-art speed
- Also random, ASCII, and comma delimited

Advanced Fourth Generation Language

- Event driven architecture
- Procedural, with over 120 commands
- Lightbar menus
- Global subroutines
- Global variables
- Directory and path management
- Two dimensional tables
- Variables: string, numeric, floating point, Boolean
- DOS shell
- On line debugger
- Julian dates
- Sophisticated editor included

Report Generator

- Relational, multifile WYSIWYG

Modern Transaction Processing

- Transaction recovery and rollback

Security

- Optimizer encrypts and speeds up programs
- Data encryption
- Screen security
- Easy, consistent, predictable, corrupted file recovery

LAN and Multiuser

Record and file lock for:

- Novell Netware
- IBM PC-Network
- 3COM 3PLUS
- Software Link PC-MOS

Join the next generation! Order your Q-PRO 4 today.
Only $795.00 with windows.
$495.00 without. $50.00 evaluation

q-n-e™ international

136 Granite Hill Court
Langhorne, PA 19047

(800) 333-0448

TLX 291765

CIRCLE 82 ON READER SERVICE CARD

different compared to similar peripherals found on the standard DOS-based PC. Moving an application that requires a mouse or graphics display can be impossible if the version of UNIX does not support these facilities.

Everyone is familiar with modular design and the reasons for it. It will be much easier to port a DOS application that isolates system-specific features into modules. Modularity will definitely make a difference when moving applications to either OS/2 or UNIX. How an application is partitioned is important, especially for graphic applications. Although there are similarities, Windows, Presentation Manager, and X-Windows are significantly different in almost all aspects from dialog boxes and menus to drawing functions and text support.

The application itself can also be a factor in determining the difficulty of porting from DOS to OS/2 or UNIX. Applications that require a great deal of processing power may find the UNIX environment stifling because resources must be shared with other users. The throughput to the display terminal may also limit what applications will run under UNIX. Single-user UNIX workstations fair better than multiuser UNIX systems in these aspects, and the single-user version is on a par with OS/2 for porting DOS applications.

It is difficult to make a sweeping statement about the suitability of OS/2 or UNIX as platforms when porting existing DOS applications. OS/2 tends to have a better match because the file system and file names are identical and the peripheral environment tends to match that found in DOS. Graphics-based applications will have the same problems with both operating systems although a Windows application will probably port to the Presentation Manager much more easily than to X-Windows.

Summary

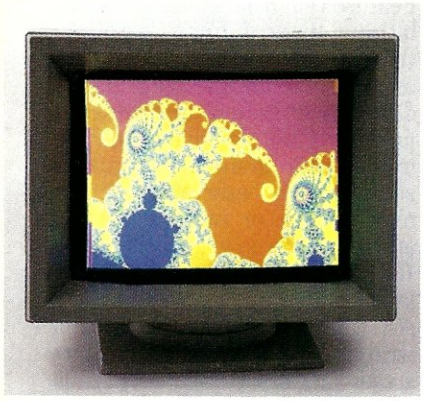
OS/2 and UNIX are the two dominant, multitasking operating system alternatives available for the PC. The former is designed as a single-user environment and could grow into a multiuser system while the latter is designed as a multiuser platform and performs well as a single-user environment. Both work well in a LAN environment. Although neither provide the solution for all problems or all users, there is a significant overlap in the solutions the two do provide.

Porting DOS applications to either OS/2 or UNIX can be a major endeavor. Neither operating system offers a direct method for porting applications to their native environment, although some 80386 versions of UNIX do offer DOS compatibility modes similar to the DOS compatibility box found in OS/2. These tend to be interim solutions, however, and tend to have significant limitations which makes it preferable to port DOS applications to the native environment for integration purposes in addition to taking advantage of the features found in these operating systems.

Porting applications written in high-level languages such as C, COBOL, and FORTRAN tend to be the easiest approach because of the compatibility between compilers and the similarity of system functions in DOS, OS/2, and UNIX. Problems will tend to occur with programs written in assembler and BASIC (due to lack of standards for BASIC compilers). Dependence on peripherals found on PC-compatible machines is also a major source of problems. As always, a well-designed, modular program will be easier to port. □

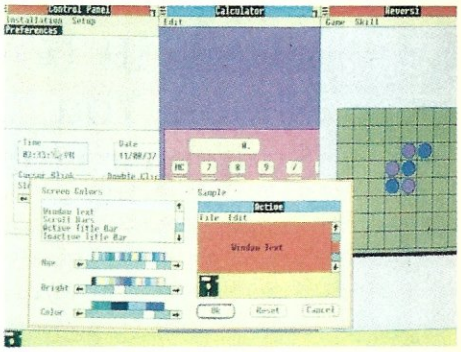
VERY HIGH RESOLUTION

The PC Tech COLOR and MONOCHROME video processor boards employ the TMS 34010 high performance graphics co-processor to insure the best possible video performance at reasonable prices.



Color 34010 Video Processor:

- **Featured on the cover of Micro Cornucopia.**
- **From 800 x 512 through 1024 x 800 resolution (depending on monitor and configuration).**
- **8 Bits per pixel for 256 simultaneous colors**
- **Hardware support for CGA/MDA emulation.**
- **PC, XT, and AT compatible**



The PC Tech Color 34010 video processor is a superior 34010 native code and DGIS development tool. We support up to 4 megabytes of program (non-display) 34010 RAM as well as up to 768K bytes of display RAM. ***Compare our architecture and prices to any other intelligent graphics board. Then choose the PC Tech Color 34010 Video Processor for your development engine and your production requirements as well.***

Color 34010 Video Processor $1,195.00

Price includes 512K display RAM, 1024K program RAM, and utility software. Monitor not included.
Also available: DGIS, 34010 C compiler, assembler, 34010 fractal software,
additional display and program memory, and various monitor options.

PC Tech Monochrome 34010 Video Processor and Monitor

- **736 x 1024 resolution (other options available)**
- **2 bits per pixel for 4 hardware gray shades**
- **Hardware support for CGA/MDA/Hercules emulation**
- **PC, XT, and AT compatible**
- **Full page 66 line text editing with many popular editors**
- **Excellent windows 2.0 application development system**

The graphics and bit manipulation capabilities of the TMS 34010 make the PC Tech Monochrome 34010 Video Processor 66 line full page text and graphics display faster than many 25 line systems. The video processor is available separately or with the high resolution white phosphor monitor shown above.

Monochrome 34010 Video Sub-System $1,295.00

Price includes Monochrome Video Processor and monitor pictured above.
Also available: DGIS, TI 34010 C compiler, TI assembler.
Monochrome 34010 Video Processor also available separately.

Designed, Sold and Serviced By:

PTech

904 N. 6th St.
Lake City, MN 55041
(612) 345-4555
(612) 345-5514 (FAX)

CIRCLE 46 ON READER SERVICE CARD

by P. L. Olympia, Ph.D.

dBASE III TO dBASE IV: Should You Switch or Fight?

Unless you've been hibernating during the last few months you now know that dBASE IV is promised for release at the end of July. The amount of press it has been getting since the formal announcement last February 17 leaves little doubt that this is one product that will have a profound influence on the microcomputer market. At the same time, it is easy to become confused by the daily deluge of information from the media about product features and "misfeatures." Normally, I do not discuss unreleased products, but the debate that already has surrounded dBASE IV merits clarification and comment.

In this column, and in future columns, I hope to be able to provide some of the information you need to assess whether dBASE IV is for you, and what conversion problems, if any, you may have in switching over from dBASE III Plus. Rather than simply recite the list of anticipated changes or describe those changes superficially, I prefer to focus on a select group of features in each issue. However, bear in mind that dBASE IV is not even in beta test as I write this column (early May). Any features I discuss are always subject to change. By the same token, press reports, such as those alluding to the existence of 3300 bugs (amazing how one can keep score) in an alpha version of the product, are meaningless and should not be taken seriously. That is not to say that dBASE IV will be completely free of bugs when it is released. It may not be, but no one should be criticized for having program bugs in an alpha release. That, after all, is why we have beta tests.

P. L. Olympia, Ph.D. is a scientist with a doctorate in Chemical Physics. He is co-author of a new book, dBASE Power: Building and Using Programming Tools, *published by Ashton-Tate.*

Overview

dBASE IV is a very large and complex product. If you thought the changes from dBASE II to III were mind-boggling, you'll appreciate knowing that those pale in comparison to what dBASE IV offers beyond dBASE III Plus. Some of the noteworthy enhancements are:

- A pseudo compiler that automatically compiles a program file (.PRG) into object form (.DBO). Commands such as *DO <prg>* and *SET PROCEDURE TO <proc>* will invoke the compiler automatically.
- A new and more powerful, nonprocedural interface called the *Control Center* that will make you forget the tortuous dBASE *Assistant* (if you haven't already). Using pull-down menus, the Control Center gives you access to almost everything in dBASE IV, and without you having to write a single line of code. The help system is also improved. It allows you to branch to Rolodex-style help panels of related topics. You may invoke the print option of a help panel to get a hard copy of it.
- Support for Structured Query Language (SQL). *SET SQL ON/OFF* determines whether dBASE IV interprets queries in SQL or native mode. SQL queries cause the program to translate the query into dBASE language, compile it, then execute it. There are no current plans to provide access to the intermediate dBASE code for use, perhaps, in learning how the translation is done.
- Vastly improved multiuser capabilities. The program provides automatic record and file locking, automatic retry, and a facility to indicate which network user has a lock on a record or file.
- Features such as multiple child relations, user-defined functions (UDF), and arrays for which most of us went to competing products. dBASE IV UDFs may be written only in the dBASE language and not traditional programming languages, such as C or assembler. Arrays may be one- or two-dimensional, and may have up to 1025 elements. (I will discuss UDFs and arrays in later columns.)
- A useful forms manager and report writer. You may design data entry screens and reports simply by "painting" the screen with data fields, computed fields, boxes, and text. dBASE IV translates those into .PRGs which, obviously, can be compiled.
- A greatly enhanced dBASE language including new commands and functions. I will be discussing improvements on the dBASE language in more detail later.

dBASE IV actually comes in two flavors: the end-user version and the developer version. The developer version includes everything offered in the end-user version, plus an interactive debugger, unlimited run time, a three-user LAN pack for program development, and additional technical reference manuals, including a description of how dBASE handles parameters to the CALL statement. The end-user version costs $795, and the developer version costs $1,295.

Program Limits

Remember when dBASE II always tested your ingenuity because its data files could only have 32 fields, and it allowed you to open only two data files at one time? You may still remember the relief with which you greeted the arrival of dBASE III, which gave you the facility to have 128 fields per file and up to 10 data files open at one time. But, just as there is no such thing as too much speed or too much disk space in the world of computing, it didn't take long to outgrow even those limits. Here, dBASE IV provides some relief, but not much. A dBASE IV data file may have up to 255 fields. Supposedly, up to 99 files may be open at one time (compared to dBASE III's 15), but like dBASE III, only 10 of those may be data files. In essence, you still have only 10 work

areas.

dBASE IV raises the limit on the number of user-defined memory variables from 256 in dBASE III to 2048. Additionally, it maintains automatic printer system variables that are accessible by any program. The dBASE IV text editor is also much improved; it is now able to handle lines up to 1024 characters long—the same as the maximum length of a command line, up from 256 in dBASE III.

The maximum number of procedures in a procedure file is up from 32 to 1170. One of the problems with dBASE III Plus and its clones is their failure to support more than one procedure file at a time. There is talk that dBASE IV may have multiple procedure files open simultaneously, but even this may not be necessary. Any .PRG file may contain a number of procedures within it. Once the file is compiled to object form, all procedures embedded within the .PRG file are accessible by other command files in the application.

File Structure Changes

dBASE IV data files have a new field type, F, to accommodate floating point numbers requiring a high degree of precision. The old numeric field type N remains fully supported and can be used for fixed point numbers to help make a comparison between numeric variables more predictable. Internally, dBASE IV stores F and N variables differently.

When you create or modify a dBASE IV database structure, you may specify up to 47 fields in the file to be index fields. This fact is recorded in an area of the .DBF file that is unused by dBASE III so this change preserves data file compatibility between the two products. When you identify a field as a key field, dBASE IV automatically creates a multiple index (.MDX) file with the same name as the .DBF file. A .MDX file may contain up to 47 index tags, each of which consists of a name and an index expression. Thus, one .MDX file serves the function of 47 old .NDX files. The latter are still supported although they should make the list of endangered species in no time. With an .MDX file, you need to have only one file open instead of 47, all tags are updated automatically, and the index expression also supports descending keys. You may have more than one .MDX file open at one time, so in principle, there is no practical limit on the number of indexes concurrently active.

Lest you think that .MDX key expressions are confined to individual fields in the file, remember that tags containing complex expressions may be assigned using the INDEX command. For example:

```
INDE ON STR(acct,4)+SUBS(categ,4)
TAG acctype DESC
```

defines the tag *acctype* as a descending key built from the two fields *acct* and *categ*. You may specify the controlling index in the USE command, for example:

```
USE mydbf ORDER acctype OF mymdx
```

assigns tag *acctype* of *mymdx.mdx* to be the controlling index for the data file *mydbf.dbf*.

> *dBASE IV raises the limit on the number of user-defined memory variables to 2048.*

dBASE IV can use and create the .NDX files of dBASE III. I am not able to tell any difference in the structure between .NDX files of the two versions. Unless, you have a need to go back and forth between dBASE III Plus and dBASE IV, there is no advantage to using .NDX over .MDX files. Naturally, you can't have a *descending* clause in an INDEX command meant to create an .NDX file.

Preparing for the Transition

Will your dBASE III Plus programs and data files be usable under dBASE IV without change? For the most part, the answer is "yes." Since dBASE IV automatically compiles programs to object form, my favorite technique of embedding comments inside IF .F.-ENDIF blocks should be modified slightly to include a TEXT-ENDTEXT block, just as you would write the comments in Clipper. It is possible to have an application system run under the two dBASE versions as long as you avoid using dBASE IV enhancements.

dBASE IV has no problem reading dBASE III index, report (.FRM), and label (.LBL) files, even though it creates .PRG files in place of the latter two. Using .PRG files in place of .FRM and .LBL has the dual advantages of using fewer file handles and being able to compile the resulting code. dBASE IV also reads dBASE III memo field files (.DBT) without difficulty.

Given that dBASE III Plus applications may be ported into dBASE IV with a minimum of problems, any concerns you may have about switching over when the time comes must be for reasons other than compatibility. As you may have guessed, the power that dBASE IV gives you comes at the price of requiring more hardware resources. While dBASE III Plus could be run on a machine with dual floppies and 256K of RAM, dBASE IV requires 640K of RAM, and a disk space of at least two megabytes.

What about retraining? dBASE III Plus programmers may require some training on the new features and capabilities of dBASE IV to gain maximum benefit from the software. Any experienced dBASE user should be able to learn the new features by self-study and experimentation.

I expect more inexperienced users to develop custom applications with dBASE IV because of the ease and power of the Control Center, the forms manager, the application generator, and the report writer. Next time, I'll discuss report writing and memo fields support in dBASE IV. □

Did you find this article particularly useful? Circle number 11 on the reader service card.

by A.G.W. Cameron

Transputers and the Sun 386i

For some time I have been curious about the transputers, and particularly about the transputer boards from MicroWay that have been widely advertised. A transputer is a CPU chip made by Inmos, a British firm, which is supposed to be very fast both for integer operations (the T414 chip) and for floating point operations (the T800 chip, a T414 with floating point circuitry added). Single T800s have been advertised to deliver 1.5 mflops (millions of floating point operations per second). However, the feature for which transputers are most noted is that they can be hooked up to operate in a parallel network, thus giving extremely fast throughput for certain kinds of problems.

Therefore, it was with a considerable degree of interest that I attended a single-day transputer seminar hosted by MicroWay. It turned out that the advertised 1.5 mflops is quite misleading; that value is for operations on 3×3 matrices that can be installed on internal memory in the transputer chip. A somewhat more realistic problem such as the LINPACK, which involves matrices of at least 100×100, gives only 0.36 mflops as reported to me by one of the attendees who had made the measurement. This is about twice as fast as a microVAX II. Several other people mentioned other problems that also ran about twice as fast on a single transputer (a Monoputer) as on a microVAX II.

This performance would have been considered extremely good in 1986, and quite good in 1987, but in 1988 it is so-so. Part of the reason is that I am beginning to detect a widespread sentiment that "if it's DEC, it's slow!" However, since MicroWay will sell you a 20-MHz Monoputer with 2 megabytes of RAM that inserts into an XT or AT bus for $1,995, this is still a very good value for the money. For the same price they will sell you a 20-MHz Weitek 1167 daughter board that can be used with certain 386 machines to give about four times the speed of a microVAX (see my column, "Number Crunching Coprocessor Boards," *Micro/Systems*, February 1988). Thus, you have little to gain unless you can use a network of transputers.

A single T414 chip contains several distinct components. The main component is a 32-bit CPU. This talks to a bus on which there is a block of circuitry providing system services. There are 4 kilobytes of on-chip RAM, there is a memory interface, and there are 4 link interfaces. The T800 then adds a 64-bit floating point unit to the chip. This follows the IEEE-754 standard for floating point arithmetic except that it does not support transcendentals or trigonometric operations.

The link interfaces provide a means of connecting transputers together into a network. MicroWay sells a board called a Quadputer that contains four transputers that can be connected together in several possible ways. The transputers may be either T414s or T800s in any desired mix. One of the transputers talks to the PC motherboard and it and the rest of the transputers talk to one another. The Quadputer board may be only partially populated, leading to Biputers and Triputers. The Quadputer may typically be set up with 1 or 4 megabytes of RAM per transputer. If you want more memory than the 2 megabytes provided with a Monoputer, I suspect that MicroWay will be willing to sell you a Quadputer with only one transputer on it. MicroWay is also planning to offer more flexible memory options in the future. To get a rough idea of prices, assume it will cost $1,500 for each T414 with 1 megabyte of RAM, $2,000 for each similar module where you substitute a T800 for a T414, and $1,400 for each upgrade of 1 megabyte to 4 megabytes.

Parallelism in computing can mean a lot of different things. The lowest level of parallelism is a vector processor in which successive steps of an instruction set are overlapped for a stream of incoming data. In a tightly coupled computer architecture, there may be many CPUs executing the same instruction set with multiple input data streams. A transputer network is a loosely coupled architecture with each CPU having its own task (and accompanying instruction set) and working with its own set of data. As a consequence, in a transputer network, communication between transputers takes place only when both sides of the communication link are ready for it. In practice, what this means is that the transmitting transputer sends a byte of data and waits for acknowledgment. When the receiver is ready to receive, it takes the first byte out of its buffer, sends the acknowledgment, and then the communication proceeds. The transmitter need not be suspended while waiting because it may have a number of parallel tasks to perform, of which the transmission is only one.

A common linkage arrangement for a Quadputer is what is called a transputer farm, in which one transputer talks to the external bus and runs a main control program. This transputer delegates tasks to the other three transputers, called "workers."

In the future, it is planned to increase the transputer operating speed to 30 MHz, which will result in a 50 percent increase in throughput, unless that throughput is limited by the speed of the external bus.

Transputer Software

A special language called OCCAM was developed for programming transputers at the same time that the transputer was designed. In order to achieve greatest efficiency at present, programs must by written in OCCAM. Programs written in conventional high-level languages, such as C, FORTRAN, or Pascal, can have their tasks sent out to other transputers for execution if special versions of those languages are used to compile components of a problem, and if those components are linked together by a framework of channels written in OCCAM. This framework is then referred to as a "harness."

At the lowest level in OCCAM there are three types of process. An assignment sets a variable equal to the value of an expression: $v := e$. An input receives a value from a channel c and assigns it to a variable x; the notation is $c ? x$. An output places the value of an expression e in the channel c, with notation $c ! e$.

These processes can be combined to form a construct, which is itself a process, and which in turn can be a

A. G. W. Cameron is Professor of Astronomy at the Harvard-Smithsonian Center for Astrophysics.

component of other constructs. Examples of a few kinds of constructs are as follows: A sequential construct consists of a header *SEQ* followed by a list of component processes that are to be executed in the order indicated. A parallel construct consists of a header *PAR* followed by a list of processes that are to be executed together (in any order), and which are called concurrent processes. Communication, consisting at the lowest level of a combination of an input and an output process, is itself a construct. A conditional construct, consisting of a set of processes, one of which will be executed when a condition becomes true, is written as a header *IF* followed by pairs of condition statements and the action to be taken. The alternative construct *ALT* is similar, but in this case a list of alternative input conditions is associated with actions to be taken, only one of which is performed, depending upon which input first becomes ready. There are loop (WHILE) and selection (CASE) constructs. The equivalent of a DO loop, known as replication, may be written as *SEQ i = 0 FOR n* or as *PAR i = 0 FOR n*. The language goes on to define types, declarations, procedures, functions, expressions, timers, peripheral access mechanisms, and configuration mechanisms, including a two-level priority declaration.

It may be seen that OCCAM differs from a conventional programming language primarily in its treatment of concurrent operations, including signaling. Each process has its own workspace and linkage area allocated in memory. The local workspace contains local variables and arrays. The linkage information contains pointers that the transputer uses to do housekeeping on multitasking and I/O protocols. Processes run to completion unless they are de-scheduled or suspended. De-scheduling occurs if the process is waiting for input, output, or a timer event. A low-priority process is suspended if a high-priority process becomes ready to run.

Various efforts are under way to write a transputer operating system. At present, OCCAM simply executes a single program unless several independent tasks are written together into one program, which would be awkward. Two versions of "Parallel C" are in late stages of development. Promised for the future are "Parallel FORTRAN" and "Parallel Pascal."

The Sun 386i Workstation

On the same day as MicroWay's transputer seminar, Sun Microsystems introduced its new 386i workstations. I went into Boston the following day to see them demonstrated and I was very impressed.

These workstations compete with the top-of-the-line 386 computers from IBM, Compaq, and AT&T. They come in two flavors: The 386i/150 uses 20-MHz 386 and 387 chips, and it is optional whether you purchase a 32-kilobyte cache memory with the system. The 386i/250 is the top of the line, using 25-MHz 386 and 387 chips and with the cache as standard equipment. The introduction of the 250 coincided with Intel's announcement of the availability of the 25-MHz chips.

> *The state of the art of transputer hardware is still quite primitive . . . but it is getting better quickly.*

There are 4 megabytes of RAM standard with the 150, and 8 megabytes of RAM standard with the 250, and both may be expanded to 16 megabytes.

However, the most distinctive aspects of the 386i are in the realm of software. It is a UNIX machine in which Sun has melded the AT&T System V.3 and the 4.3/4.2 BSD versions of UNIX, and support for DOS 3.3 has been intimately integrated into the operating system. Several DOS sessions may be run simultaneously in windows; Sun calls this its DOS Windows facility. One limitation that seems unnecessarily restrictive to me is that the amount of extended memory that can be associated with a DOS Window is only 2 megabytes. Sun does not intend to support OS/2 at present, since both UNIX and the DOS Windows system are inherently multitasking.

Sun is intending to storm the corporate PC market with these machines. This means that, in order to be credible and also to be user-friendly, it has had to write a completely new UNIX front end, called the *Sun Organizer*. This provides the use of screen icons and a mouse, and thus it resembles the user environment of the Macintosh (or perhaps, in these litigious times, one should say that both resemble the user interface designed at the Xerox Palo Alto Research Center). Thus the normal, bare-bones UNIX interface is well hidden from the user, although Sun says that confirmed UNIX users will be able to use this interface if they want to. Among other things, the Sun Organizer presents the user with a diagram of the directory tree structure and makes it easy to navigate through that structure. There is a "Help" key that gives a help message about the operation being attempted, or about subtopics of that operation; this uses a hypertext facility.

UNIX is, of course, a huge operating system, and so a big disk is needed with the 386i. That big disk may be on an Ethernet network server; Sun emphasizes the use of the network for connecting groups of these workstations for a given operation within a corporation. But 91- and 327-megabyte disks are available for use directly with the workstation. There are SCSI interfaces for these disks; the 91-megabyte disk has an 18 millisecond seek time and the 327-megabyte disk has an even shorter seek time at 16.5 milliseconds. Using a small expansion unit, up to three 327 megabyte disks can be installed together with a 60-megabyte tape cartridge drive.

Within a few months the 386i will also be of considerable interest to the scientific and technical market. A FORTRAN compiler (to run under UNIX) is currently in beta test. The machines currently carry a 387 coprocessor, but this plugs into a Weitek socket, and the Weitek 1167 coprocessor board is expected to be available to plug into this socket about in September (this is currently waiting for the availability of 25-MHz Weitek chips). With a 386i thus enhanced, it should run floating point problems faster than any other existing Sun workstation, which will be of particular interest to scientists and engineers. In September, Sun is expected to announce a much faster SPARC-based workstation, but it will be much more expensive than the 386i. □

Product Information

MicroWay
P. O. Box 79
Kingston, MA 02364
(617) 746-7341.
Circle reader service #262.

Sun Microsystems Inc.
2550 Garcia Avenue
Mountain View, CA 94043
(415) 960-1300.
Circle reader service #263.

by Robert Blacher

The Shareware Alternative

Let's get a housekeeping matter out of the way first. The title of this column notwithstanding, there is almost no "public domain" software available for downloading from DOS bulletin board systems (BBS). A program is in the public domain only if the author claims no rights to the program. That's a mistake. If a program is public domain, someone else can actually sell the program without violating the author's rights.

Most software on BBS either is, or should be, copyrighted. What we loosely refer to as public domain really is copyrighted software that the author is distributing for free. The author hasn't abandoned all legal rights to the software; you're just not being charged for the license to use it.

The other major category of software on BBS systems is variously referred to as shareware, freeware (so much for the English language—freeware isn't free), or user-supported software. Again, the software is copyrighted, but this time you are not being given the right to use the software for free. The author is providing you with an opportunity to try the software and see if it meets your needs. If you find yourself using the program regularly, you are requested to send in a license fee.

When I first started running an RCP/M over four years ago, almost all of the software was free. (Actually, I can't recall any CP/M programs for which the author requested money.) The same still holds true for UNIX software. Almost all software that is distributed over USENET, the anarchic "uucp" network that distributes mail and programs to thousands of UNIX sites worldwide, is free. There has been hot and heavy discussion on the net from time to time about whether shareware should even be allowed.

I don't have any problem with shareware. Many authors of shareware programs spend a great deal of time creating, testing, and supporting their programs. It seems only fair to ask users of those programs to contribute financially. But, a lot of authors have unrealistic expectations of the returns from the shareware "market" and are disappointed when they receive few registrations. That's leading to yet another category of software on BBS systems called "crippleware"—programs where the BBS version is intentionally limited in some way to encourage users to register the program and receive a full operating version.

I do have a problem with crippleware. But, rather than going off on a tirade about that subject, I want to talk about reasonable expectations with regard to BBS-distributed software for both users and authors. Crippleware stems from unrealistic notions about getting rich from shareware programs. Forget it. A little thought about what's really going on out in the BBS world may help clear up those misconceptions and save us all a lot of grief.

Why Shareware?

Let's look at it first from the caller's perspective. There are three major categories of PC applications: (1) word processors; (2) spreadsheets; and (3) database managers. Most of us bought our PCs precisely because we wanted to use them for one or more of these tasks. Is it realistic to think that someone who uses his PC primarily for word processing is going to forsake WordPerfect or Microsoft Word in favor of a program downloaded from a BBS? Will he give up Lotus 1-2-3 for a shareware spreadsheet? Will he abandon dBASE for a file manager found on a BBS? C'mon now!

I'm not saying there aren't good programs in each of these categories available on BBS systems. There most certainly are: ***NYWord*** and ***Galaxy*** are both good word processors; ***PC-Calc Plus, QubeCalc***, and ***Insta-Calc*** are fine spreadsheets; and ***File Express*** and ***PC-File Plus*** are competent, easy-to-use file managers. But look at the competition. It costs about $200 to pick up a copy of WordPerfect by mail order these days. I recently bought Borland's Quattro for $99. Reflex goes for about the same, and Symantec's Q & A, which combines simple word processing with powerful database capabilities, is advertised by mail order for $209. Sure, dBASE and the like go for a lot more, but there isn't a full-fledged, relational database manager with a procedural language available on BBS systems.

In short, the competition from commercial programs in these three categories is very stiff indeed, and some of the programs aren't selling for much more than their shareware counterparts. That doesn't mean there's no hope for shareware in these areas. Someone who uses a computer primarily for word processing may find that File Express is exactly what he needs in the database area. Hopefully, that person, and others, will register the program if he continues to use it and bring the Expressware folks their just returns. But, I doubt this is where the future of shareware really lies.

Shareware excels at doing things the commercial software houses can't do. The game isn't to compete with Microsoft, Lotus, and Ashton-Tate. Rather, it's to figure out what needs those companies *can't* serve and target those areas. The area of DOS utilities is a good place to start, and Vern Buerg's indispensable *LIST* program is a perfect example.

LIST

LIST is a file viewer, and a whole lot more. It's perfectly silly, but DOS fails to provide a convenient way to look at text files that are larger than one screen. If you *TYPE* such a file, you'd better have fast fingers or it will go scrolling off your monitor. DOS does provide the *MORE* program, but it's pretty clumsy stuff. If you enter *MORE < TEXT.FIL*, you'll get the file one page at a time. But look out. Enter *MORE > TEXT.FIL* and you've just erased it! Is this any way to run an operating system?

You could load your favorite word processor to read text files, but that's a case of gross overkill. LIST offers the ideal solution. Into a tiny 8K .COM file, Buerg has packed an incredible array of capabilities. Of course, it allows you to page through a file, using the Home, End, PgUp, PgDn, and arrow keys that are intuitive and

---

Robert Blacher operates the popular Computer Connections PCBoard bulletin board system in Washington, D.C. It is quite busy, but those willing to place their modems on redial may call (202) 547-2008. In his other life, Bob is an attorney.

handy. That's just the beginning. Experienced users of LIST know you can use it to search through a file, mark a block of text and dump it to another file, print all or part of the file, put a ruler on the screen to help you count column positions, and even examine binary files in hex mode. There's much more to be found in this wonderful program.

As the above makes clear, I think LIST is invaluable. But in monetary terms, how much is it worth? Buerg requests a contribution of $15. I happen to think that's a steal and suspect the program would readily sell for more, but there's an upper limit for a "small" program of this kind. $25? $35? $50? I'm not exactly sure, but it's somewhere in that range.

Look around at some of the software ads. How many $15 programs do you see? Precious few, if any. It's just not practical for a major software house to market a program at that price. With the costs of development, testing, marketing, advertising, and distribution, it's impossible for a Microsoft to sell a "small" program, no matter how badly needed that program is. Right there, you begin to see at least one market niche for shareware. There are whole categories of indispensable, "small" programs that just can't be marketed commercially. Shareware provides a perfect, alternative method of distribution.

What's Hot on BBS?

In general, you'll find utility programs on BBS systems that simply have no parallel in the commercial market. Here's a quick look at a few more that you'll find on any good BBS:

- ***PCOPY***: Created by Norm Patriquin, this excellent program has all the capabilities of DOS's XCOPY and REPLACE programs and much more. You can move files, as well as copy them, and the selection criteria are very extensive.

- ***OVERVIEW*** is an excellent, sweep-type file manager that allows you to view all files in a directory and tag them for copying, erasure, etc. There are commercial programs in this area, such as XTREE, but the BBS world really has those beat.

- ***FGREP***: Written by Chris Dunford, this is a text search utility. DOS's FIND command is a pale counterpart to this program in terms of both capabilities and speed. And FGREP is free!

- ***SORTF***: Also created by Vern Buerg, the DOS SORT program just can't hold a candle to this one. DOS's program drops dead on files larger than 64K. Buerg's program will sort much larger files and do so faster and with more options. SORTF is also free.

- ***WSSINDEX***: This shareware program keeps a catalog of what's on your floppy disks. WSSINDEX allows you to add comments about each file, catalog ARChive directories, and sort and print the catalog in a variety of ways.

- ***MARK*** and ***RELEASE***: These excellent programs by Kim Kokkonen give you control over all those TSRs you're loading these days. With this package, you can remove some or all resident programs from memory when you need to free up RAM for large applications. MARK and RELEASE are free, and source code is available.

People who haven't looked at the software available on BBS systems are missing out on programs like these. It's not just a question of saving money. There's software available on BBS systems that simply has no commercial counterpart because it's economically impractical for companies to market low-cost programs of this kind.

If you're an author of a program and are considering distribution via BBS systems, let me urge you to go into this venture with your eyes wide open. Think about the programs that are already available, both in the commercial and the shareware arenas. Then, think hard about why you're doing this. If your primary motivation is financial, you're likely to be disappointed. Regardless of efforts to beg, bribe, or otherwise coerce shareware program users, most users won't contribute. Ask a public television station how many of its viewers pay the yearly membership fee. Shareware is no different, and most users will take the "free ride." But, if your program truly meets a need and you get it into the hands of enough people, you can hope for some fair reward—if not wealth, at least some congratulatory comments in messages on BBS systems and a favorable mention in future editions of this column. □

Did you find this article particularly useful? Circle number 10 on the reader service card.

New Products

New Software Products

Novell Branches Into Apple's Network Market
Novell, Inc., recently unveiled the latest version of its network operating system, NetWare Version 2.15, which will support Apple technology as well as DOS and OS/2. NetWare v2.15 is AppleShare compatible and allows users in an AppleTalk Network System to become fully NetWare compatible. Each workstation uses its native operating system, so users of Apple, PC, and PS/2 workstations each operate in their own environment while sharing files and data across the network without conversion. NetWare for the Macintosh may be installed in either the NetWare file server or in an external NetWare bridge workstation, and provides access to Apple LaserWriter printers to all network users. NetWare v2.15 also offers Mac users access to Novell's System Fault Tolerance (SFT) and security. NetWare also supports AppleTalk protocols inside the server, and provides full support of AppleShare security and privacy controls, and provides NetWare security and accounting functions to AppleShare. The initial release includes support for LocalTalk and EtherTalk adapters for the Macintosh, and future compatibility is slated for Token-ring, ARCnet, and other adapters.

NetWare Version 2.15 is scheduled for distribution in the fourth quarter in the SFT ($4,695), and Advanced ($2,695) versions. Entry Level System (ELS) Level II will be available early in 1989 for $1,395. For existing NetWare v2.1 and v.2.11 sites, NetWare v2.15 will be available as an upgrade for $200 per file server. For more information, contact **Novell, Inc.**, 122 East 1700 South, Provo, UT 84601; (801) 379-5900, or contact your Novell authorized reseller.

Circle number 15 on the reader service card

Toolkit Hammers DOS Into Shape for UNIX
The MKS Toolkit Version 2.3 adds more than 115 UNIX System V-compatible commands to DOS on the PS/2, PC, and compatibles. Included in MKS Toolkit is a parser-generator language, YACC (Yet Another Compiler Compiler), that takes context-free grammar, converts it into a set of tables, and generates a program that will recognize that language. MKS YACC is fully compatible with UNIX YACC. Also included is a spell function, a *diff* command to compare three versions of a text file, and data compression.

The MKS Toolkit lists for $169 and is available to run under DOS 2.0 an later. For more information, contact **Mortice Kern Systems Inc.**, 35 King Street North, Waterloo, Ontario, N2J 2W9; (519) 884-2251.

Circle number 16 on the reader service card

New Hardware Products

Laptop Analyzer Can Smell Trouble on LANs
Network General Corporation has introduced a laptop version of its Sniffer, Model PA-302. Housed in a Toshiba T3200 with a 12-MHz 286 processor, the Laptop Sniffer is a portable protocol analyzer and diagnostic tool for Ethernet-based networks. The Laptop Sniffer operates with all Ethernet protocols, including TCP/IP, DECnet, XNS, NFS, ISO, APPC, NETBIOS, and NetWare. Network General is also expanding its Sniffer series to support Token-ring, ARCnet, and StarLAN technologies. The self-contained unit can capture network data transmissions and break them down to isolate problems that degrade network performance. The Laptop Sniffer includes Version 1.3 of the LAN troubleshooting software that offers graphical display of network activity, frame size control, an improved frame capture rate, traffic generator capabilities, and detailed protocol analysis. The new software version also includes TeleSniffer, which enables the laptop computer to conduct LAN diagnosis via telephone lines.

The Laptop Sniffer is priced at $15,000. For more information, contact **Network General Corporation**, 1945A Charleston Road, Mountain View, CA 94043; (415) 965-9608.

Circle number 17 on the reader service card

WAN Controllers Conform to OSI Standard
Retix has unveiled the PC 300 series of Wide Area Network Controllers to connect PCs, XTs, ATs, and PS/2 units to OSI WANs through leased lines. This series of WAN controllers supports Open Systems Interconnection (OSI) and both the connection-oriented (CONS) and connectionless (CLNS) network services. The PC-320 has 512K of RAM and an onboard, low-speed RS-232 port. Other versions can be configured with up to 1.5 MB of RAM, and a high-speed port (up to 64 kbps) is optional. The PC-321 supports RS-449, RS-530, or X.24 connections. The PC-322 supports RS-232 or V.35 connections. Software provided with the controllers includes the SW-330 package with ISO Transport Class OS/2 and 4, the ISO Internet Protocol IP to x.25, Subnetwork Dependent Convergence Protocol (SNDCP), x.25 Packet-Level Protocol, and HDLC.

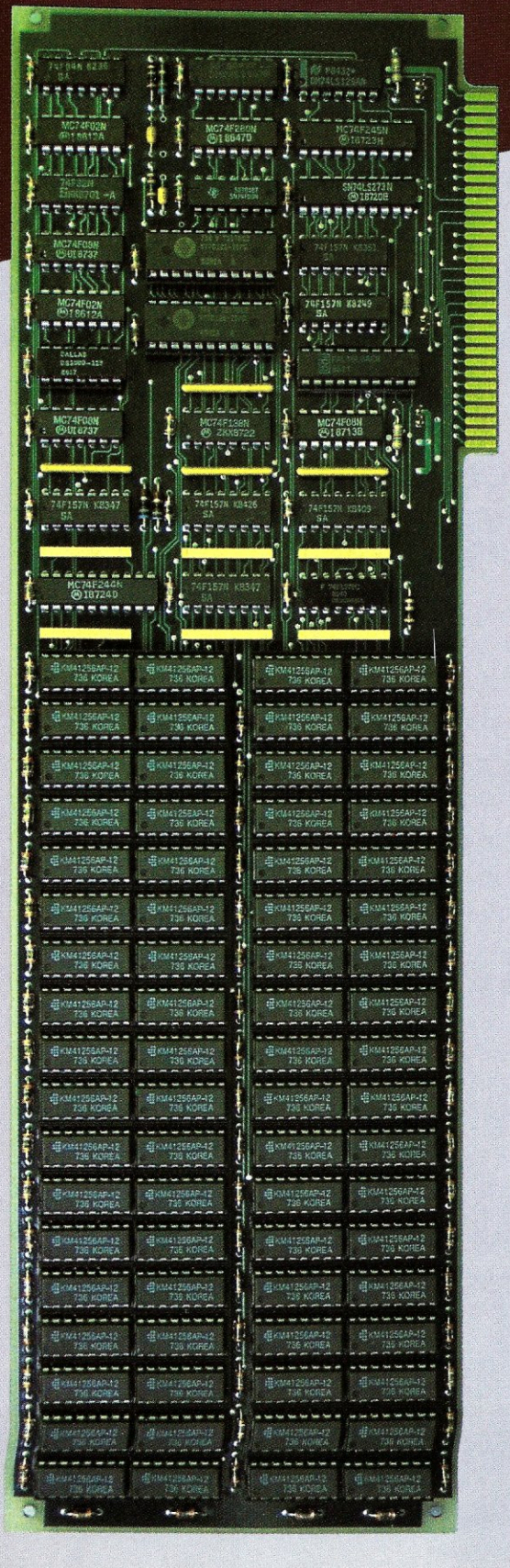
The PC-300 is scheduled for shipment in the fourth quarter. For more information, contact **Retix**, 2644 30th Street, Santa Monica, CA 90405; (213) 399-2200.

Circle number 18 on the reader service card

Novell's Netware, Version 2.15 supports DOS, Apple, and OS/2

X-BANDIT

Designed for the EMS 4.0 Standard



DESIGN PHILOSOPHY

• The Teletek X-Bandit was specifically designed to utilize the advanced features of the Lotus/Intel/Microsoft EMS 4.0 Specification. It is available in both 8 and 16 bit versions for use in the IBM XT, AT, and compatibles.

MEMORY

• Segmented Memory Mapping allows the user to fill out unused memory segments between 640K and 1 Megabyte.

• Split Memory Addressing allows the user to fill out conventional memory to 640K.

• Extended Memory Addressing is available for the PC/AT version.

• 2 MB capacity in a single slot. Up to 8 MB per system.

• Parity checking.

SOFTWARE

• Easy menu-driven auto configuration software.

• Device driver includes print spooler and RAM drive.

• Supports multitasking with the appropriate shell-resident software package.

SPEED

• 6/8/10/12 MHz speed with 0 wait states. 16 MHz speed with 1 wait state.

WARRANTY

• One year parts and labor.

TELETEK

4600 Pell Drive
Sacramento, CA 95838
(916) 920-4600
Telex #499-1834

DOS system running Lotus 1-2-3

SCO XENIX system running SCO Professional

THIS IS AN IBM PS/2 MODEL 80 RUNNING DOS

Under DOS, this PS/2™ is a powerful 80386-based single-tasking, single-user computer that can run thousands of DOS applications. In 16-bit, 8086 mode.

One at a time.

When OS/2™ software becomes available, the PS/2 can become a multitasking, single-user computer running in 16-bit, 286 mode that can also single-task those DOS applications under OS/2.

One at a time.

With DOS or OS/2, the PS/2 will support one user.

	1 user (DOS)	1 user (OS/2)
Cost per system**:	$12,389	$12,594
Cost per user:	**$12,389**	**$12,594**

THIS IS AN IBM PS/2 MODEL 80 RUNNING SCO XENIX

Under SCO XENIX,® this PS/2 becomes a powerful 80386-based multitasking, multiuser computer that can run thousands of XENIX applications. In full-tilt, 32-bit, 386 mode.

Many at a time.

And using SCO VP/ix,™* the PS/2 can multitask DOS applications under SCO XENIX.

Many at a time.

With SCO XENIX, the PS/2 will support one user.
Or 9 users. Or even 33 users.

And it can do all that today because you can get SCO XENIX for the PS/2—*now!*

	1 user	9 users	33 users
Cost per system**:	$14,559	$19,726	$40,402
Cost per user:	**$14,559**	**$2,192**	**$1,224**

SCO XENIX System V and the SCO XENIX family of software solutions are available for all industry-standard 8086-, 80286-, and 80386-based computers, and the IBM® Personal System/2™ Models 50, 60, and 80.

SCO™
THE SANTA CRUZ OPERATION

(800) 626-UNIX (626-8649)
(408) 425-7222
FAX: (408) 458-4227
TWX: 910-598-4510 SCO SACZ
uucp: ...decvax!microsoft!sco!info!

*SCO VP/ix available as separate product.

Cost comparisons are based on most recently published U.S. domestic suggested list prices. **Cost model: Base machine: IBM PS/2 Model 80, 70Mb disk, 1Mb RAM, IBM 8512 color monitor, 1Mb additional IBM RAM, IBM ProPrinter XL.® **1-user DOS system:** Base machine, plus DOS 3.3, WordPerfect 4.2, Lotus® 1-2-3® dBASE III PLUS.® **1-user OS/2 system:** 1-user DOS system; substitute OS/2 for DOS. **1-user SCO XENIX system:** Base machine, plus SCO XENIX 386 for PS/2, SCO VP/ix, SCO Lyrix® (word processing), SCO FoxBASE+™ (dBASE III PLUS workalike), SCO Professional™ (1-2-3 workalike). **9-user SCO XENIX system:** 1-user SCO XENIX system, plus intelligent 8-user multiport card, 8 IBM 3151 ASCII terminals. **33-user SCO XENIX system:** 9-user SCO XENIX System, plus 3 more intelligent 8-user multiport cards, 24 more IBM 3151 ASCII terminals, 4 Mb additional RAM, additional 70Mb disk.