$3.00

MICRO CORNUCOPIA

The Micro Technical Journal

Dec – Jan 1986 No. 27

The Fast & Cheap Issue

Plus

SAVE OVER 30% ON OUR GIFT PACKS!
60-DAY MONEY-BACK GUARANTEE

How Borland's Three New Holiday Packs Will Fill Your Stocking Without Emptying Your Piggybank.

Three special packs with dazzling discounts that will help get you into a Holiday mood. You can get some of Turbo, most of Turbo, or all of Turbo—including the two newest members of the Turbo family, Turbo GameWorks™ and Turbo Editor Toolbox™. You also get our unmatched 60-day money-back guarantee, quality products that aren't copy-protected.

TURBO NEW PACK $95.00.

You get the two exciting new members of the Turbo Pascal family,

- TURBO GAMEWORKS, Chess, Bridge, and Go-Moku, complete with source code and a 200-page manual.
- TURBO EDITOR TOOLBOX, all the building blocks to make your own editors and word processors, complete with source code and a 200-page manual.

TURBO HOLIDAY PACK $125.00.

You get all three of the Turbo family classics for only $125.00 (about a 30% discount). Turbo Pascal 3.0 and Turbo Tutor and Turbo DataBase Toolbox—all for just $125.00.

- TURBO PASCAL combines the fastest Pascal compiler with an integrated development environment.
- TURBO TUTOR teaches you step-by-step how to use Turbo Pascal with commented source code for all program examples on diskette.
- TURBO DATABASE TOOLBOX offers three problem-solving modules for your Turbo Pascal programs: Turbo Access, Turbo Sort, and GINST, which generates a ready-to-run installation program that lets you forget about adapting your software to specific terminals.

TURBO HOLIDAY JUMBO PACK $245.00.

This is it—the whole thing, the entire Turbo family including its two newest members. You get:

- Turbo Pascal
- Turbo Graphix Toolbox
- Turbo Tutor
- Turbo DataBase Toolbox
- Turbo GameWorks (NEW!)
- Turbo Editor Toolbox (NEW!)

and you pay only $245.00 for all six! Which means that you're getting everything at only about $40 a piece. Quite a holiday deal. (And if you already own one or several members of the Turbo family, be creative—nothing can stop you from buying the Jumbo Pack, picking out the ones you already have and giving the rest as holiday gifts to family and friends. At these prices you can afford to give to others and to yourself.) Speaking of Holidays, this offer lasts until March 31, 1986. (At Borland, we like to make the Holidays last.)

NEW! TURBO GAMEWORKS $69.95.

Our new Turbo GameWorks offers games you can play and replay without Turbo Pascal or revise and rewrite with Turbo Pascal 3.0. We give you the source code, the manual, the diskettes and the competitive edge. Chess, Bridge and Go-Moku. State-of-the-art games that let you be player, referee, and rules committee all at once because you have the Turbo Pascal source code. Learn exactly how the games are made—so you can go off and make your own. And Turbo GameWorks is the only quality game you can buy that is not copy-protected. Sold separately, only $69.95. (Just $47.50 if you buy the Turbo New Pack.)

NEW! TURBO EDITOR TOOLBOX $69.95.

Build your own word processor—for only $69.95! You get ready-to-compile source code, a full-featured word processor that looks and acts like WordStar™, and a 200-page manual that tells you how to integrate the editor procedures and functions into your programs. With Turbo Editor Toolbox you can have the best of all word processors. You can make WordStar behave like Multi-Mate. Support windows just like Microsoft's Word. And do it as fast as WordPerfect does it. Incorporate your new "hybrids" into your programs to achieve incredible control and power. Sold separately, only $69.95. (If you buy the Turbo New Pack, the price drops to just $47.50.)

NOT COPY-PROTECTED

Holiday Gift Packs, Turbo GameWorks™ & Turbo Editor Toolbox™

Available at better dealers nationwide. Call (800) 556-2283 for the dealer nearest you. To order by Credit Card call (800) 255-8008, CA (800) 742-1133.

		Quantity
*Turbo Holiday Jumbo Pack	$245.00	___
*Turbo Holiday Pack	$125.00	___
*Turbo New Pack	$95.00	___
Pascal	$69.95	___
Pascal w/8087	$109.90	___
Pascal w/BCD	$109.90	___
Pascal w/8087 and BCD	$124.95	___
Turbo DataBase	$54.95	___
Turbo Graphix	$54.95	___
Turbo Tutor	$34.95	___
Turbo Editor	$69.95	___
Turbo GameWorks	$69.95	___

These prices include shipping to all U.S. cities. All foreign orders add $10 per product ordered.

The holiday packs include an upgrade coupon for both options so you get BCD and 8087 support for $39.95 (regularly $55.00).

*Gift Pack Offers Last Until March 31, 1986

Carefully describe your computer system!

Mine is: — 8-bit — 16-bit

I use: — PC-DOS — MS-DOS — CP/M-80 — CP/M-86

My computer's name and model is: ______

The disk size I use is: ☐ 3½" ☐ 5¼" ☐ 8"

Name: ______

Shipping Address: ______

City: ______

State: ______ Zip: ______

Telephone: ______

Amount: (CA add 6% tax) ______

Payment: VISA MC Check Bank Draft

Credit Card Expiration Date: ___ / ___

Card # ______

NOTE: Turbo Editor Toolbox and Turbo GameWorks are available for the IBM PC and true-compatibles using Turbo Pascal 3.0 ONLY.

COD's and Purchase Orders will not be accepted by Borland International. California residents: add 6% sales tax. Outside USA: add $10 and make payment by bank draft, payable in U.S. dollars drawn on a U.S. bank. H8

BORLAND INTERNATIONAL

4585 Scotts Valley Drive, Scotts Valley CA 95066
Phone (408) 438-8400 Telex 172373

MICRO CORNUCOPIA
P.O. Box 223
Bend, Oregon 97709
503-382-5060 Orders Only
503-382-8048 Tech. 9 - Noon

MICRO CORNUCOPIA supports systems programming languages and builders of single board and S-100 systems.

Application to mail at second class postage rates is pending at Bend, OR 97701 and additional entries. Published bimonthly by Micro Cornucopia Inc., 155 NW Hawthorne, Bend, OR 97701. POSTMASTER: Send address changes to Micro Cornucopia, PO Box 223, Bend, OR 97709.

SUBSCRIPTION RATES:

1 yr. (2nd Class)	$16.00
1 yr. (Canada & Mexico)	$22.00
1 yr. (Other foreign)	$30.00

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

ADVERTISING RATES: Available on request (call Alice Holbrow).

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

SOFTWARE, HARDWARE, AND BOOK VENDORS: We would very much like to review your software, Single Board, and S-100 products. Send materials to the Review Department, Micro Cornucopia.

ISSN 0747-587X

MICRO CORNUCOPIA

Dec-Jan 1986 The Micro Technical Journal No. 27

We're Crazy (But It Helps)

Gary had just hung up the phone when he looked over at me and chuckled. The caller had asked about the DSI-32 board.

"How did we like it?" Gary told him that we loved it, that it's very easy to use, very fast, really neat.

"What are we most excited about?" the caller asked. Gary replied that right now we're really up on our new XT clones. I mean clones: 4.77MHz, 8088, two 360K drives, IBM look-alike cabinet, color and B&W graphics, sound, the works.

There was a pause at the other end (I'd pause, too). Then Gary added that last month we were just as excited about the DSI board.

Before you conclude that we've gone totally off the deep end (it's really not such a bad place) let me explain.

Really Cheap Vs. Sorta Cheap

I've been doing some nosing around after hearing rumors that the cheap IBM look-alike systems advertised in Byte and Computer Shopper, etc. are being put together out of very inexpensive Taiwanese boards. I found a number of small outfits (one or two-person garage operations) that were assembling these systems and then selling them for $800 to $900 to computer stores. These systems are currently being sold retail for $1200 to $1300.

Searching For A Source

I asked these garage operators how I could contact the importers or distributors. The silence was deafening. It was obvious that if they could limit access to this information, they could keep their market. Locate the sources, or at least someone willing to live with a reasonable markup, and I could put together a complete PC for $500 to $700.

I knew that the assembly was a simple matter of bolting down the compatible motherboard in the compatible cabinet, plugging in the compatible power supply connector and the compatible I/O and graphics cards, connecting up the compatible drives, and I'd be done. It should take less than an hour (it did). The most difficult part was figuring out which boards I needed (there are zillions to choose from).

I spent a number of hours on the phone poking around, trying to connect up to this inner circle. A number of builders I talked to were paying too much, $1000 to build a system. Not cheap enough.

(continued on page 86)

Regular Features

LETTERS

Last Computer

Your editorial in Issue 26 was something I could readily identify with. I, too, feel the need to grow beyond my humble Z80 beginnings. For months I have been scanning the magazines, preparing to buy "my last computer." I also share your apathy toward MS-DOS, the IBM PC, and its clone legions. No warm fuzzies whatsoever.

I'm unhappy about the dearth of information about other processors, operating systems, and languages. Where are the reviews and benchmarks of systems running UNIX, P-systems, or OS-9? PC clones are certainly cheap, but would a little more money buy a much more useful, powerful, longterm investment? There's loads of literature floating around, but where are the hard facts?

I'd like to share my opinions concerning various processors and operating systems touched on in your editorial.

I like the Definicon 32032 board, but I'm not keen about the fact that software development retains the essential personality of MS-DOS. The board provides a lot of raw power, but its fatal flaw is the lack of a real operating system on the 32032 side of the interface. As other 32032 boards are developed (as I am sure there are and will be) there will not be any clear way for software to be shared between a new board and the DSI-32. I think it unlikely that anyone besides Green Hills will provide significant software in Definicon/MS-DOS/32032 format. Software distribution for non-Intel processors is bad enough without introducing another kludge format.

Turning to the 68000, I find that operating systems and languages do exist, but they don't seem to pop up much in reviews or literature. In the search for My Last Computer, I have dismissed CP/M68K, since it offers me virtually no features that aren't present in CP/M 80.

I am intrigued by the Amiga. I especially like the multitasking nature of its software, but I am annoyed that they came out with their own non-standard DOS.

I want many of the features of UNIX: the multitasking, the pipes, the shell, and the directory structure, to name a few. I've had some experience with Coherent, a UNIX look-alike, and I liked it a lot for software development.

68000 UNIX is only available on very expensive S-100 systems and the AT&T Unix PC; even with a university discount the AT&T tops $4000. The literature that I've read on the AT&T also suggests that it is almost worthless unless you substitute the 20 megabyte hard disk for the standard 10 meg.

UNIX is not easy to update or customize: if you have the source you are generally expected to maintain it yourself, and if you don't have the source you can't change it.

I am seriously considering OS-9/68K. I like everything about it. It is UNIX-like, meaning that it has multitasking and hierarchical directories, as well as all the other essential Unix characteristics. It is modular (you can tinker with it).

It has better real-time capabilities than UNIX. OS-9 has well-defined disk formats across different hardware systems, and is compact enough to be run on systems with only floppy disks. It appears to be supported by a fair number of software vendors, and many of these have already had success supporting the 8-bit version of OS-9.

What are the drawbacks of OS-9/68K? It is a relatively new product, and while it is popular in VME bus systems, not too many low-cost 68000 systems exist to take advantage of it. It also lacks some of the more esoteric UNIX utilities: lex, yacc, etc.

I don't share your emphasis on sound and graphics. I think that the necessary graphics software standards exist, and that new computers and software should be adhering to the standards.

Kurt Liebezeit
505 W. Springfield Ave.
Champaign IL 61820

Editor's note:

If you're really serious about buying a last computer, then I'm not sure what to suggest except that you make sure it's expandable.

Trevor is madly working on UNIX for the 32032 — should be ready any time now. The Amiga will be fairly cheap (if it ever shows up) but probably won't be super cheap because the custom operating system and custom ICs will make it difficult to copy.

As for OS-9 68000, I agree that it's one of the more interesting UNIX type operating systems. However, it needs a very popular system to carry it to glory, and that system will have to have excellent graphics to be successful (just as MS-DOS rode the PC to fame and fortune).

Graphics and sound have become vital, in my opinion. Once users have them they don't want to be without them (especially if there is lots of software support). As for graphics standards, no generic style graphics standard has become popular.

Gem looked interesting because it defined the graphics environment and because it worked very much like the MAC. Unfortunately, it worked too much like the MAC, and Apple sued. DR is now rewriting Gem.

I still think that the Amiga's combination of power, graphics, sound, and speed make it the heir apparent to the PC. (Go ahead, Commodore, prove me wrong!)

Issue 25 Input

In Technical Tips, Turner Caldwell shows that he really believes in doing things the hard way. Maybe I'm off base, but it seems that removing the 6 or 8 screws to change jumper settings would be a whole lot easier than building or modifying a circuit. Since we configured our 8 keyboards, they give an absolutely beautiful upper case/lower case with 8 bit output. (Our keyboards require -12V.)

Jumpers	Setting
E1 E2 E3 O O—O	Enables both upper and lower case
E4 E5 E6 O O—O	Space bar is encoded 20H
E7 E8 E9 O—O O	Use domestic code on some keys (disables German codes)

Merely unscrew the board (they're sturdy), use a small screwdriver to remove jumpers (they're not sol-

LETTERS

dered), and relocate to this configuration.

The 113 key keyboard by MicroSwitch is available from Cascade Electronics.

Steve Wells
3211 NE 55th
Kansas City MO 64119
(816) 455-2598

Patching WordStar

I updated my Kaypro II (pre-83) to a Kaypro 4 and installed your Pro-8 monitor ROM. I also installed two Mitsubishi 4851 DDDS floppies. WOW, what a difference! Your instructions were great and everything works as advertised. The Mitsi floppies plugged right in. No modifications were necessary — even the screw holes fit. The screen print function on the Pro-8 is great. Now I can print the menus of my programs to study at a later time and it works great with WordStar.

I had one little problem. That was with WordStar 3.0. When WS was booted, a cursor would appear that was different than the CP/M cursor. It came up as a flashing "[", no matter what the cursor was set in CP/M using Config83.COM.

The solution I found was to patch WordStar 3.0 at location 0292 (Trmini) with 01 1A 00 00 00 00 00 00 00. Now I can change my WS cursor at will by using Config83.COM.

One more thing. I was worried about transferring my files from SSDD floppies to DSDD. I was very pleasantly surprised to find that the Pro-8 monitor chip will automatically read or write either SSDD or DSDD, depending on which disk is booted. I love it!

Bob Cabler
610 S. 6th Ave.
Yuma AZ 85364

Editor's note:

You wouldn't have an extra set of compatible screw holes, would you, Bob? We've got a couple of systems here we're going to have to drill, otherwise.

This 'n That

I've just finished the 5MHz, Pro-8 and Plus 4 upgrades, and I have a few comments that may help others. Buying drives from Payload Computer Services in Texas removed the anxiety from what I often call "the mail ordeal." Payload had Mitsubishi quads (4853) in stock, and had them to me in four days with minimal shipping charges. The price per quad of $140 is competitive, and Payload gives great service and phone support. They advertise in Computer Shopper.

I really appreciated the drilling diagram in Issue 21. Apparently the threads in the drive cases are softer than those on the 3 mm bolts included with mine, as trying to thread the bolt at a slight angle when the holes were a touch out of alignment resulted in stripping. The solution, of course, is to mount the drives with the bottom of the casing loose so there's some slop to let you get the holes aligned just right.

Another Computer Shopper discovery: The Arnold Co., also in Texas, sells Y-adapter kits for drive power chaining for $3.75 each. It's nice to have some wiring work that can be done in advance, leaving only a plug-in when you're finishing the installation. You need one per added drive.

Last but not least. I'm still amazed about this one. If any of your readers have trouble with the MOVCPM-63K process or normal sysgening, I recommend answering the source drive prompt in sysgen with a drive designation rather than carriage return. My version of Sysgen (from the updates that Kaypro released in mid-1983 with Perfect Writer 1.20) also includes the "Source drive name (or return to skip)" prompt, but I had never skipped before. Just as well, because this version apparently creates a track that sends the system south on a cold boot — and leaves it there — unless you answer the question with a drive letter. If a dealer hadn't mentioned it, I'd still be wondering whether the drives or my soldering were acting up.

Loren Marshall
1705 Bartlett Dr.
Anchorage AK 99507

Editor's note:

You always answer the "source drive name" question with a drive (usually A) when SYSGENing unless you have purposely placed the system in memory using MOVCPM or DDT immediately before running SYSGEN. (Otherwise, SYSGEN is putting whatever garbage is in memory onto the system tracks.)

Quiet Drive Access

Many thanks for sending me "Notes On 3 ms Step Rate." Before I made your suggested modifications, the stepper motors in my Shugart 465 quad density drives in my Kaypro 8-84 were rasping and squawking loudly as they accessed the disk, apparently at the speed of 6 ms. Lubricating the rail did nothing.

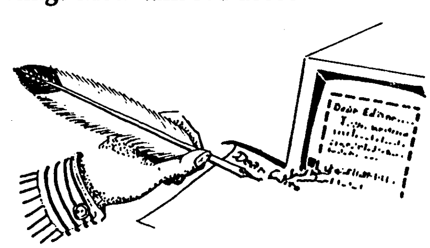
Piggybacking a 74LS157 to the U60 and following the rest of your instructions to the letter changed the stepper motor speed, quieted the drives to a whisper, and kept the drives — at least to this point — working perfectly.

Jim Ballard
10614 32nd Dr. SE
Everett WA 98204

Bulletin Board Notice

The Cleveland County bulletin board, located in Shelby, NC, is a general BBS on line 24 hours per day 7 days per week, and is free to callers. The 45 different sections serve S-100, CP/M, UNIX/ZENIX, etc., and users can chat with Willard (artificial intelligence). Call (704) 482-8012.

Dan Wise, SYSOP
Rt. 2 Box 914 Woodbridge
Kings Mountain NC 28086



The New NEC Microprocessors – 8080, 8086, Or 8088?

By Randy Davis

Rt 5 Box 107K
Greenville TX 75401
214-454-2426

Suppose you're a microcomputer chip design engineer working on a new family of microprocessors which need to be pin compatible with, but faster than, the 8086 family of microprocessors. And let's say that near the end of the design process you discover you still have quite a bit of space left in your microcode area. What do you do with the extra space?

First, you include the 186 extensions to the 8086 instruction set. But you still have room, so you add a few bit and BCD oriented instructions (since the absence of these was a complaint with the 8086's original instruction set). Unfortunately, you still have room, a little microcode space. So (as a joke maybe) you add the entire instruction set of another microprocessor! (Nothing fancy, just the 8080.)

Clever you and clever NEC (Nippon Electric Corporation) to solve the space problem just this way when they designed the first two members of their new line of microprocessors, the V-Series. (NEC intends to release the V40 and V50, 16/32 bit processors, and the V60, a super 32-bitter, in the future).

The V20 and V30, are pin compatible replacements for the 8088 and 8086, respectively, with the following improvements —

1. Fewer clock cycles for many instructions.
2. Inclusion of the 186 instruction extensions.
3. Addition of bit and BCD oriented instructions.
4. Addition of a new repeat prefix.
5. Support for NEC's 8087 replacement to be introduced later.
6. Inclusion of an 8080 mode, under which the chip executes the entire 8080 instruction set.

The Entire 8080 Instruction Set?

Yes, the entire 8080 instruction set, and here's how it works. Modern microprocessors are designed as microcoded machines. That means that their hardware understands a very tiny set of instructions. Microcode translates the standard object code instructions in a .COM or .EXE file into the simpler instructions that the hardware understands. Thus a simple hardware design plus some fancy microcode can understand any set of instructions.

Each microcode instruction executes in some base cycle (time). For instructions, the base unit is the CPU clock cycle. For memory fetches the base unit is the bus cycle, which is four cycles of the CPU clock.

Simple instructions such as move register to register and simple operations between registers execute in a scant two to four clock cycles. Instructions which involve external memory take a much larger number of clock cycles.

For example, moving register data to memory requires 12 to 23 clock cycles depending on addressing mode used and whether the processor is a 16-bit bussed 8086 or an 8-bit bussed 8088. The longest instructions, the multiply and divide, may take more than 100 cycles in both the 8086 and 8088.

Because of improvements in the art of chip manufacture the designers of the V-Series processors are able to include extra hardware which the 8086 family of processors doesn't have, such as an effective address generator circuit, extra internal registers, and a second internal data bus.

These internal architectural improvements make the microcoding of many of the instructions more efficient — most notably the multiply and divide. Thus, at the same clock speed, the V20 operates between 5 and 30 percent faster than the 8088 and the V30 is between 10 and 40 percent faster than the 8086. The increase depends on the instruction mix.

So how do you add a whole new instruction set without getting confused?

The V-Series processor uses two status register bits that were unused in the 8088/86. When one of these bits, the emulation flag, is set, the chip is in native mode and executes its "normal" 8086-like instructions. (In the original 8086, this flag bit is permanently set to 1.) But when the flag is cleared, a different set of microcode which processes 8080 instructions is given control and the chip enters "emulation mode."

Why 8080?

The 8086 processor family was designed as a replacement for the 8080/85 family. In order to ease the transition for programmers of the 8080/85, many of its structural details were made similar. Also, the 8080's instruction set is fairly simple, mostly owing to the hardware limitations of the day when the chip was designed. So it was easy for the NEC chip designers to add the 8080 instruction set to hardware originally designed for the 8086s.

Unfortunately, the V-Series microprocessors do not include the Z80 extensions. The Z80 was not designed by the same company as the 8080 and 8086 families and, therefore, the 8086 does not reflect the Z80 improvements to the basic 8080 instruction set. This probably increased the difficulty of implementing these additions on the V-Series hardware beyond the amount of free microcode space available.

(Editor's note: It turns out that the Z80 was designed at Intel, but the Z80 design team left when the company decided not to market the chip. Thus Zilog was formed. The Z80 has a much larger instruction set than the 8080 which is probably why the emulators don't support it yet.)

CP/M Under MS-DOS

Since the V-Series processors are pin compatible replacements, they can be inserted into any machine which uses one of the 8086 family processors, most notably the IBM PC and its many clones. Use of this chip adds the already noted modest advantages of speed and instruction set to the PC.

But to the owner of a CP/M computer who has resisted purchasing a PC clone (because it wouldn't run his large library of CP/M programs) and to the owner of a PC who would like access to this large base of CP/M programs, adding a V-Series processor, with its 8080 mode, opens the PC family of machines to the CP/M family of software.

Of course, even with the V20 or V30, MS-DOS machines need a little software help to run CP/M programs. Two such help packages are GFI's

EMULATOR and MicroSolutions' UNIDOS. These two products take very different approaches and bear closer examination; however, both CP/M emulators have the following properties in common.

Both emulators must present a system interface which looks like CP/M to the 8080 program. In practice this means the emulator must initialize a program's PSP area properly and properly handle BDOS and BIOS calls. Code which doesn't use system calls (i.e., talks directly to ROMs or ports) won't run.

BDOS Calls

The handling of the BDOS calls is straightforward. MS-DOS 1.0 was modeled directly after CP/M to minimize the difficulty software houses would have in translating their programs from CP/M over to the new environment. Although MS-DOS 2.x introduced a new set of Unix-like system calls which were easier to use, the originals were retained for compatibility. Thus, almost all CP/M BDOS calls can be executed by setting up the registers properly and calling the corresponding MS-DOS system calls.

BIOS Calls

Handling the BIOS calls is slightly more complicated, since IBM didn't feel obliged to pattern its BIOS calls after those of CP/M. However, the CP/M calls aren't difficult and can generally be mimicked by one or more of the ROM BIOS entry points.

One problem arises because of the slightly different philosophies of the two operating systems. For example, CP/M doesn't keep date and time stamps, while MS-DOS doesn't load a disk with Control-C. These functions can almost always be handled at the BDOS level by either ignoring the call and/or returning some default value (such as Jan 1, 1980 for the time/date stamp and logging all drives in permanently).

At the BIOS call level, it isn't quite as easy since these are more hardware oriented. Calls which cannot be emulated are trapped by the BIOS and BDOS handlers and generate a message and an immediate exit from the program.

Other differences cause no problems at all. Even though CP/M doesn't support path names, for example, CP/M programs can be run out of subdirectories since MS-DOS assumes the current path for filenames if none is provided. Most of the improvements of MS-DOS are available to the CP/M program when run under emulation. One exception is redirection: the original MS-DOS system calls do not support redirection and it is, therefore, not available to CP/M programs.

May We Interrupt?

No matter what mode the V20/V30 is in, it always reverts to native ('8086') mode to service a hardware interrupt. Since disk accessories, spoolers, and the like run off the interrupts, these work properly with CP/M programs running under emulation.

For example, in the middle of running MicroPro's CP/M WordStar from a RAM disk, you're free to bring up a SideKick window. Neither SideKick nor the RAM disk does anything to diminish your 63K TPA. You should be able to do all this as a task under a multi-tasker such as MicroSoft's Windows with 8086 programs running concurrently in the other grounds.

GFI Emulation

EMULATOR from GFI Electronics, 1800 Avalon, Olathe, KS 66062, 913-829-0157, retails for $49.95 and isn't copy protected. Besides several support programs for the V-Series processor, the EMULATOR disk contains a menu driven installation program.

You are left to your own devices for getting CPM .COM files onto an MS-DOS format disk. Once the programs are transferred to the MS-DOS formatted media, you run the installation program, which installs a header onto the beginning of each CP/M program. This process generates new .COM files which are executable on a V-Series equipped machine.

An MS-DOS loader loads both the CP/M program and the prefix code into memory. It then passes control to the prefix program. This preliminary code first checks to make sure the host processor is indeed a V-Series model; if not, an error message is generated and execution stops.

The code then calculates the beginning of the CP/M program area, sets up the BIOS jump table at the very top of the 64K CP/M area and a PSP in the first 100 bytes. Then it sets up jumps to the BDOS and BIOS code handlers at address 0 and 5. All of the BDOS and BIOS handlers point back to the prefix code, which handles the system calls in native mode. This allows for a TPA (transient program area) of 63K+.

UNIDOS Emulation

The CP/M package UNIDOS from MicroSolutions (125 S. Fourth St., DeKalb, IL 60115, 815-756-3411) has a somewhat different approach.

If you have Uniform, it lets you reconfigure one of your floppy drives to almost any CP/M format. (limited by the floppy controller).

For example, if I install drive B as an Osborne SSDD, then it looks like an MS-DOS drive when I call drive B, but it pretends to be an Osborne drive when I ask for drive D.

The $69.95 UNIDOS package (Uniform sells for another $69.95) is installed by invoking it from the command line or from the AUTOEXEC.BAT file before you intend to run the CP/M program. UNIDOS can work with or without Uniform, but without it it's like EMULATOR — you must figure out some way to get the .COM file onto an MS-DOS disk.

Once you've converted the file to MS-DOS format (via Uniform, Media Master, or a serial connection between your CP/M machine and the PC) you need to change the extension from .COM to .CPM before UNIDOS will run the program.

Then you can execute the program by entering its name, just as if it were a .COM file. After MS-DOS looks and doesn't find a .COM or .EXE file of the specified name, UNIDOS takes over and looks for the .CPM file, executing it as a CP/M program.

(continued next page)

(continued from page 5)

A Nice Mesh

UNIDOS meshes nicely with Uniform, if you install Uniform before installing UNIDOS. Together they form a powerful, albeit somewhat pricey, combination.

Using the previous Osborne example, if I access drive D, Uniform handles the format conversions so my MS-DOS operating system thinks the disk is one of its own.

Now, if I execute a .COM file off that Osborne disk, UNIDOS, seeing that drive D is an Osborne format disk, assumes that the .COM file is a CP/M executable file and executes it under emulation. Therefore, with both products installed I can run MS-DOS programs off an MS-DOS format disk in drive A and CP/M programs off a CP/M formatted disk in drive B (D) on the same machine.

Also, UNIDOS has the added capability of running Z80 programs. When told that the .CPM file contains Z80 instructions, it executes the CP/M program by running a software emulator program (it also runs all CP/M programs this way when executed on a non-V20/30 processor). Although this may sound like an advantage, the resulting execution is so slow it's almost worthless.

UNIDOS Terminal Emulation

UDRIVER converts screen output from that intended for a CP/M terminal to either the PC's monochrome or color graphics screen. The number of screens that UDRIVER can emulate is much smaller than the number of disk formats Uniform can handle, but the list includes many of the standards (such as VT100, Televideo, etc.).

Fast?

How fast do the V20 and V30 processors run in emulation mode?

First, note that since the 8080 is an 8-bit CPU, CP/M code is 8-bit oriented and the V30 doesn't have much of an advantage over the 8-bit bussed V20 (at least in emulation mode).

I compared the IBM PC running at a standard 4.77MHz to an Osborne 1 with a Z80 microprocessor running at 4MHz. Both were equipped with double density drives, although the Osborne's were single sided. I tried several programs typical of the ones I assumed an average user would run, using both GFI's EMULATOR and MicroSolutions' UNIDOS.

UNIDOS was run both by itself from MS-DOS formatted disks and with Uniform from Osborne formatted disks. To compare V20 performance with software CP/M emulators, I also ran UNIDOS in interpretive mode, where it doesn't make use of the V20's 8080 capabilities. The results of these benchmarks appear in Table 1.

Notice that in the empty loop and the Fibonnaci benchmark, where only CPU speed is important, all three emulation modes ran at the same speed and slightly faster than the Osborne. In the screen output benchmark, the Osborne outperformed UNIDOS slightly, with EMULATOR far behind. In the assemble and compile benchmark, where disk speed is important, UNIDOS came in first again, with EMULATOR only slightly behind, and the Osborne slightly behind it.

After running the benchmarks, I feel safe in drawing the following conclusions: the 4.77MHz V20 is very slightly faster than a 4MHz Z80 in CPU performance. The IBM's screen output seemed slower than the Osborne's. Since both are memory mapped, both should beat remote terminals handily. UNIDOS emulated CP/M system calls faster than EMULATOR could. And the PC's disk access was much faster than the Osborne's.

The above times do not include the time to load the application from disk; including that time the Osborne fares worse. And after each benchmark was run, CP/M had to reload from the boot disk, whereas MS-DOS did not (this time was also not included).

As far as a comparison between the two emulator packages, I preferred UNIDOS, but only when it was combined with Uniform. Being able to feed my PC the same disk I just pulled out of my CP/M machine was a great advantage. However, the combined package carries a much larger price tag than EMULATOR, which I would rate roughly equal with UNIDOS when Uniform wasn't present.

In Conclusion

Where can you get a V20 (replaces 8088) or V30 (replaces 8086) of your own?

Both GFI and MicroSolutions sell the V-Series processors (though MicroSolutions sells them only with their software). I've also seen several sources in the backs of microcomputer magazines (normally these ads stress the performance improvements and are often called "speed-up kits").

The price depends on make and speed and runs about \$20-\$25 for a 5MHz V20 and \$30-\$35 for an 8MHz V30.

Note that the speed improvements

Table 1 - Benchmarks

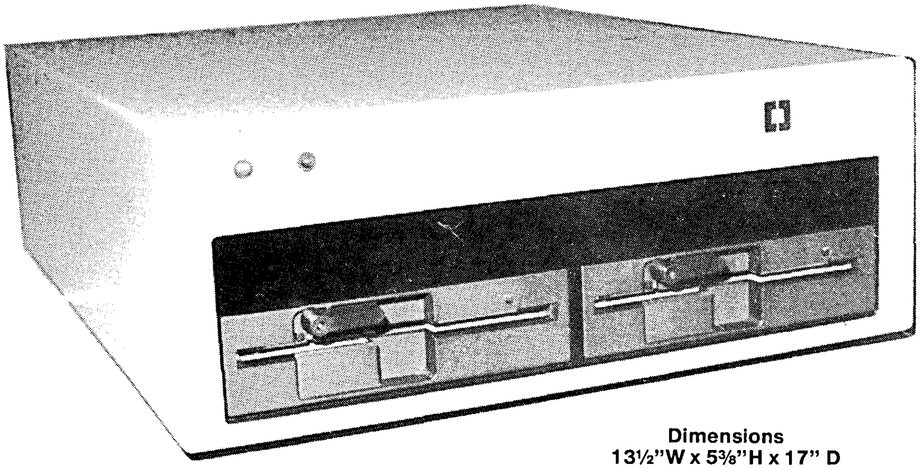
Benchmark	EMULATOR	UNIDOS w/ Uniform	UNIDOS alone	Osborne 1	UNIDOS (interpretive mode)
EBASIC empty FOR loop (10,000 iterations)	51.0	51.0	51.5	53.4	489
EBASIC Fibonacci number generator (10x100 Fibs)	12.8	12.8	12.8	13.6	117
EBASIC screen output loop (100 lines)	20.4	12.2	12.2	9.5	57.7
Assemble a .ASM file w/ AS.COM	33.4	28.9	30.7	34.8	—
Compile a .C file w/ Software Toolworks C compiler	27.3	24.5	22.1	32.9	—

(all units are in seconds and are accurate to roughly .3 sec.s) of the V20 cause problems in two MS-DOS machines (neither of which is an IBM clone). The TI Professional incorrectly times out its Programmable Interval Timer chip during power on diagnostics and then stubbornly refuses to boot up. The fix here is a simple patch to the boot up ROM (contact GFI).

The Victor 9000 will not format a disk properly with a V20 installed; there is no fix for this, although it should be no problem for an assembly programmer.

One goal of NEC's V-Series is to open up both the CP/M and the MS-DOS worlds without the expense of owning and maintaining two computers. Now if NEC would just add 68000 capability so we PC owners could also run Mac and Amiga programs (when they get here), they'd really have something.

■■■



Dimensions
13½"W x 5⅜"H x 17" D

FULL 90 DAY PARTS & LABOR WARRANTY

Colonial Data
SB-80-II

COMPLETE SYSTEMS!
4 MHZ Z80A CPU
64K RAM (EXPAND TO 128K)
DUAL 5¼" DISK DRIVES
2 SERIAL PORTS
1 PARALLEL PORT

MODEL IIA - 2/380K DSDD DRIVES	**$ 895.00**
MODEL IIB - 2/780K DSQD DRIVES	**$ 995.00**
MODEL IIC - 10 Meg H/Disk w/380K Floppy	**$1895.00**
MODEL IID - 10 Meg H/Disk w/780K Floppy	**$1995.00**

* **SB80-II** (Board Only) Wired and Tested **$275.00**

SB80-II SYSTEM OPTIONS	**PRICE**
All Models:	
Additional 2 Serial Ports (installed)	**$60.00**
Additional 64K Memory (installed)	**$50.00**
CP/M 2.2 Operating System w/Manual	**$75.00**
CP/M 3.0 Operating System w/Manual	**$250.00**
Models IIA and IIB only	
SASI Interface Option (installed)	**$35.00**
XEBEC Hard Disk Controller	**$250.00**

PICK YOUR SYSTEM! CHOOSE YOUR OPTIONS!

The Original "Better Board" SB80 from Colonial Data has just gotten BETTER YET! More on-board features like SASI interface circuitry, 4 Channel Counter Timer, Memory Expansion to 128K and MORE! All made to run the most popular programs under the CP/M Operating System.

Z80™ — ZILOG
CP/M™ — DIGITAL RESEARCH (Calif.)

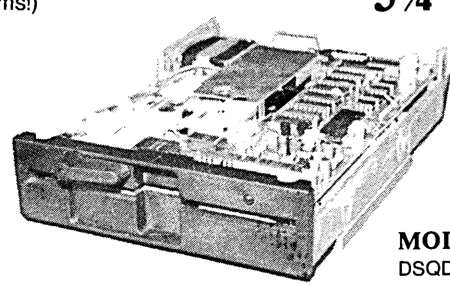
Colonial Data Cabinet Kits (Build Your Own Systems!)

MODEL CKP1
Metal Cabinet will accept Single Boards up to 12"x16" complete with fan, RFI filter with power switch, front panel and mounting for two 5¼" half-height floppies. Pre-punched for 4 DB25 and 1-50 pin. Includes 60 watt power supply with ±12V, + 5V. **$225.00**

MODEL CKP2
Same as Model CKP1, but includes 90 Watt Power Supply to power half-height Winchester 5¼" hard disk and single 5¼" half-height floppy. **$325.00**

MODEL CKX
Cabinet only with fan, RFI filter, power switch, prepunched connector holes. No Power Supply. **$139.00**

Shugart 5¼" Disk Drives



MODEL 455
DSDD Floppy **$195.00**
Over 380K Capacity 2 For **$370.00**

MODEL 465
DSQD Floppy **$225.00**
Over 780K Capacity 2 For **$430.00**

O.E.M. Quantity Inquiries Invited!
Call or Write for Details!

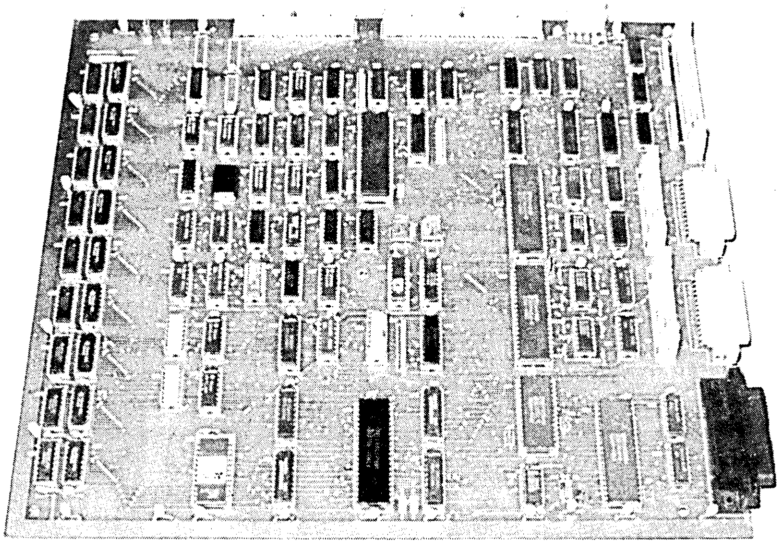
HOW TO ORDER:
Call or Write Colonial Data—Orders paid with bank card or cashier's check are shipped within 3 working days. Allow 3 weeks for Personal Checks.
Shipping: Add 2% (up to a maximum of $20.00) for UPS (Brown) shipping within the United States.

Colonial Data Services Corp.

80 Pickett District Road,
New Milford, Conn. 06776
Telephone (203) 355-3178

Colonial Data SB-80-II

The "Better Board" Gets BETTER!



$275.00
ASSEMBLED & TESTED
NOT A KIT!

Standard Features Include:

- 4MHz Z80A Processor
- 64K Memory (Expands to 128K)
- Supports 8" and 5¼" drives simultaneously
- Auto Density Selection (single/double/quad)
- 2 Serial Ports/1 Centronics Parallel Port

For the past 4 years, Colonial Data has been supplying thousands of its original SB80 "Better Board" computers to the O.E.M. market. Now, the "Better Board" is even BETTER with the introduction of the **SB80-II.** More Standard Features at a NEW LOWER PRICE!

SIZE: 12" x 13"
POWER: +12V, −12V, +5V
WARRANTY: 90 days Parts and Labor

■ 4MHz Z80A CPU WITH NO WAIT STATES
Enhanced **BIOS** uses mode 2 interrupts.

■ 64K MEMORY STANDARD (EXPAND TO 128K)
Parity checked RAM utilizes 4164
Up to 8K of EPROM (4K is standard)
Accepts Pin-compatible EPROMS from 2716 thru 2764

■ NUMEROUS FLOPPY DISK STORAGE OPTIONS
Uses the advanced NEC765A controller chip
Supports 8" and 5¼" drives simultaneously
Automatic Density Selection (Single/Double/Quad)

■ 4 SERIAL I/O PORTS (2 STANDARD)
Software selectable baud rates to 19,200
Uses Z80 SIO/0 and SMC 8116 baud rate generator.

■ Z80A PIO PARALLEL PRINTER INTERFACE
Centronics compatible printer interface is Standard!

■ SASI HARD DISK INTERFACE CIRCUITRY ON-BOARD!
SASI interface provides access to hard disk.
Compatible with XEBEC and other controllers.
Just populate with TTL and add connector.

■ 4 CHANNEL COUNTER TIMER-STANDARD!
Allows custom software applications.
User accessible 125Hz interrupt.
Provides system date/time clock.
4th Channel used for the NEC 765 interrupt.

■ CP/M 2.2 BIOS ENHANCEMENTS
Allows flexible device assignments, baud rate selection, automatic density detection, a system date and time clock and hard disk support.

■ OPTIONAL CP/M 3.0 (CP/M PLUS)
Extensive disk buffering speeds system throughout.

SB80-II SYSTEM OPTIONS:	PRICE
ADDITIONAL 2 SERIAL PORT (INSTALLED)	$60.00
ADDITIONAL 64K MEMORY (INSTALLED)	$50.00
SASI INTERFACE OPTION (INSTALLED)	$35.00
XEBEC HARD DISK CONTROLLER	$250.00
CP/M 2.2 OPERATING SYS W/MANUAL	$75.00
CP/M 3.0 OPERATING SYS W/MANUAL	$250.00

Z80™ — ZILOG   CP/M™ — DIGITAL RESEARCH (Calif.)

HOW TO ORDER:
Call or Write Colonial Data—Orders paid with bank card or cashier's check are shipped within 3 working days. Allow 3 weeks for Personal Checks.
Shipping: Add $5.00 for UPS Shipping (Brown) Within the United States.

Colonial Data Services Corp.

80 Pickett District Road,
New Milford, Conn. 06776
Telephone (203) 355-3178

The S-100 Bus

By Dave Hardy

736 Notre Dame
Grosse Pointe MI 48203

In this column Dave describes a 32-character type-ahead buffer for S-100 systems. It's the perfect addition for touch-typists with slow drives. (A Christmas present for the nimble fingered!)

Over the past few years I've received hundreds of novel S-100 add-on circuits for everything from infrared front panels to 16K core memory boards. Many of the circuits I see are of little use to the general population of S-100 users (like the adapter for running an S-100 machine in a car, or the humidity-controlled hardware breakpoint trap), but occasionally I see one that merits more than just a trip to the circular file.

S-100 Type-Ahead

The S-100 FIFO adapter shown in Figure 1 is one of the best add-on circuits I've ever seen. I'd like to give credit to the author of this work, but, unfortunately, the envelope that brought it and the letter describing it had no signature, name, or address!

NOTE: Figure 1 was not available at press time. Please write to Dave at the address above to get a copy.

What makes this circuit so handy is its ability to be added to just about any 8251-based serial I/O board to provide a 32-character (AM2812-based) type-ahead buffer. Although installing it is not a simple task, it is well worth the trouble if your machine suffers from slow keyboard response or misses characters typed in during disk accesses, or whatever.

ZZZZZzzzz

I have such a machine (SYSOP CBBS) that uses a pseudo-DMA disk controller without interrupts so it "goes to sleep" during disk accesses.

Of course, while it was sleeping it dropped characters (particularly while using WordStar or background spoolers). In fact, the disk delays were so long that even remote users complained about lost keyboard input.

Unable to solve this problem using software modifications (they won't let me run my chain saw anymore), I decided to give this circuit a try. So I wire-wrapped it onto a daughter board attached to my S-100's serial I/O card (it used an 8251, of course), and the results were impressive.

Evaluation

Even during the longest delays, I couldn't fill the buffer. Data coming into the machine from remote callers (even at 2400 BPS) was never lost. In fact, the 32-byte FIFO used in the add-on circuit never filled more than about half way. (I suspect that the 32-byte FIFO was chosen because it is commonly available, not because 32 bytes are necessary.)

Because the circuit is rather complex, it is not shown here in its entirety. If you are an engineer, E.T. (that's Electronic Technician, not Extra-Terrestrial, although some of the techs here at the lab may qualify for both), or a reasonably skilled hobbyist, you should have no trouble working from this figure (famous last words).

However, if you don't like to waste your time second-guessing the unknown, drop me a note, call me (313-846-1055), or leave me some EMAIL on Compuserve (72435,222 or 70150,102). I'll mail you a computer-generated schematic of a working (and already debugged) serial I/O card that uses this buffer.

How It Works

Here's a brief description of how the FIFO circuit in Figure 1 works: (Of course, the 8251 SIO U1 and S-100 data bus buffers U2 and U3 must already exist on the the old I/O board, but are shown here to avoid confusion.)

The FIFO circuit uses the 8251's RxRDY output to tell when the 8251 has received a character.

Then it uses the RXRDY signal to generate the RDSER* and LDFIFO signals which cause the 8251 to output its data onto the LOCAL data bus for the FIFO. Notice that this circuit uses the board's own data bus to transfer the incoming data from the UART to the FIFO, and does not ever send data out into the S-100 bus data lines (the key feature of this add-on) because the Data Output driver (U2) is not enabled by this action.

All characters received into the UART are, therefore, sent directly into the FIFO buffer through this modification. When the computer asks for the Received Data Available (RDA) status of the UART, multiplexer IC U4 sends the DATA AVAILABLE status of the FIFO rather than the UART's RDA line. (It switches only the RDA bit, of course — the rest of the UART status word is unchanged.)

Mainstreaming The Data

The remaining problem is to be sure the computer reads characters from the FIFO buffer instead of from the UART (when the computer asks for data).

IC U5 handles this by sending a READ request to the FIFO which then puts a character on the local data bus. The character is then passed through the data output buffer, U2, into the S-100 bus.

The UART's Read Request line (RD*) is now triggered when the FIFO wants data. The bus signal which used to control the UART now drives U5. This is a very clever circuit, although a bit difficult to understand...

Some things I've noticed about this circuit: If the FIFO overflows, then the first data received is lost. This shouldn't be a problem unless you type 30 cps.

If you are running CP/M version 1 or 2, avoid filling the FIFO with ctrl-C characters, since each ctrl-C will cause a warm boot, and 32 warm boots could take a while. Perhaps a good add-on for this add-on would be a circuit that would dump the FIFO when certain characters came in.

Feedback, Letters, & Requests

I've received a request for assistance from a small university in Texas. Specifically, they would like to get an S-100 frame running as an instructional tool for a course that includes installing ZCPR2 or ZCPR3, and probably for teaching some of the basics about small computer systems. (I wish there had been courses like that when I went to college.) If you know of an

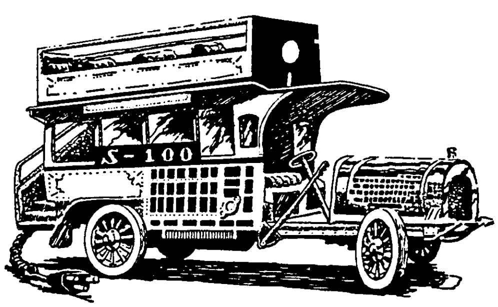
IEEE-696 machine without a home, contact me, and I will put you in touch with the right people.

About half of the readers who send me mail (I only open the packages that don't tick; the ones that do get returned) say that they would like to see more TurboDOS-related items in this column. I would like to dedicate a few issues to them, but don't want to leave other readers stranded.

But, as the pro-TD readers argue, TurboDOS can be used with several different processors, and requires some pretty sophisticated S-100 hardware, whose discussion here would benefit almost all S-100 users. So, unless I hear howls from the readership, I am going to yield to the more vocal pro-TD forces and delve into the peculiar methods of TurboDOS hardware.

Next Time

Some of the latest offerings for TurboDOS-based multi-processor S-100 machines, including some new slave processors and mixed CPUs.



■■■

EARTH
LAUNCHES NEW STARS

EARTH COMPUTERS launches two of the "Hottest" new stars in the S-100 Universe. Both the **TURBOMASTER 8**™ and **TURBOSLAVE I**™ are Star performers, featuring high speed Z-80H CPUs.

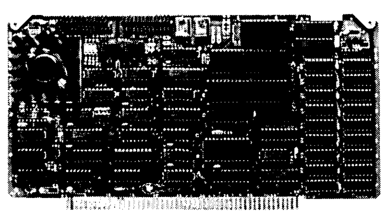
EARTH's new stars are fully compatible with the Multi-user TurboDOS operating system, and will operate in most S-100 systems, including pre-IEEE 696 systems.

TURBOMASTER 8

This outstanding new 8-Bit Single Board Computer offers features that are out of this world:

- On-board ST-506 Winchester Controller
- TurboDOS, CP/M, MP/M compatible
- 5-¼″ and 8″ Floppy Controller
- Up to 256KB of memory
- 8 MHz, Z-80H CPU
- 2 Serial ports
- 1 Parallel port

TURBOSLAVE I

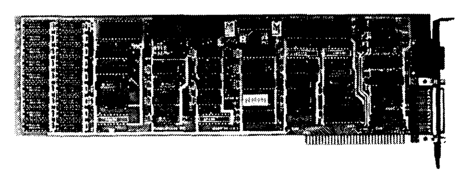


The perfect companion to the TURBOMASTER 8 or other 8/16-Bit Master processor. This high speed slave utilizes an 8 MHz Z-80H CPU and offers extensive on-board diagnostics... an industry exclusive.

- No paddle boards
- S-100, IEEE 696 compatible
- 128KB of RAM
- 2 RS-232 ports, 50–38.4K Baud
- FIFO communications
- Data transfers to 1 MB

EARTH COMPUTERS also manufacturers a growing line of PC compatible stellar performers such as:

- **TURBOSLAVE PC**™—an 8 MHz



Z-80 single board slave processor that runs CP/M applications on a PC and is compatible with the TurboDOS multi-user operating system.

- **TURBOACCEL 286**™—a high performance 80286 accelerator that boosts PC performance up to five times.
- **EARTHNET PC**™ and **EARTHNET S-100,** the low cost, ARCNET-compatible way to tie PC and S-100 systems together.

To put these stars to work for you, call or write EARTH COMPUTERS. **BE SURE TO ASK ABOUT HOW YOU CAN WIN A FREE Z-80 CO-PROCESSOR BOARD.**

EARTH COMPUTERS

"Building Blocks For The Super Micro"

P.O. Box 8067, Fountain Valley, CA 92728 • TELEX: 910 997 6120 EARTH FV • PHONE: (714)964-5784

In The Public Domain

By Stephen M. Leon

200 Winston Drive
Cliffside Park NJ 07010

Steve covers some of the history of public domain software in this column, featuring the more unusual programs. He also details the latest releases from the SIG/M and PC/Blue libraries.

Our teaser last issue was that next time we would cover the exotic aspects of public domain software. Not exotic in terms of X-Rated, but rather in terms of "why in heaven's name would anyone write such a program, much less release it!" Therefore, the topic this issue might be "software trivia" or, better yet, "trivia software."

Historically, as the early hackers solved problems with software, they used the medium of public domain as a way to provide others with their solutions. The first few volumes of the CP/M Users Group library had such practical programs as CASDSK.ASM, a cassette to disk transfer, ICOPY.ASM, to copy a file from an Isis diskette, TREAD.ASM, a tape reader to disk transfer, Laurence Livermore BASIC, Tiny BASIC, and lots of games in EBASIC.

The first volume of the SIG/M library was the infamous, original 350 Point Adventure in 8080 code. While this version was later superseded by SIG/M Volume 11 — Super Adventure, an expanded 550 point Adventure for either 8080 or Z80 code, the original Adventure was released again in June 1982 as Volume 13 of the PC/Blue library. To run it on the PC you need a Baby Blue type CP/M card. The Adventure programs are still active sellers in the SIG/M library as is the Dungeon game (SIG/M Volumes 45 to 47). I suspect that if the PC version of Adventure runs on the new NEC V-20 chip, we may see more copies in use by the non-appliance IBM set.

Eight Disks Of Stars?

Universality has never been a requirement for inclusion in a public domain library. The SIG/M library is, in fact, a kind of Smithsonian depository for obscure material. It amazes me every time I look at the distribution data, but we keep getting orders for SIG/M Volumes 31 to 38 — the Yale Catalog of Bright Stars. PC/Blue put out a two volume set of Census Data (100 and 101) and people ask for it! The point is rather simple: because the public domain libraries are not commercial ventures they can make available even the most limited use software.

Numbers, Policies, & Submits

Are you a random numbers freak? Mark Weiss, a graduate student at Syracuse, recently donated a whole series of Turbo Pascal random number generation programs to SIG/M. We showed them to one of our professorial friends who said they were the best thing he had ever seen. So we released them as SIG/M Volume 243.

But if neither random numbers nor bright stars are your cup of tea, how about music. The Musicraft Software System in on SIG/M Volumes 56 to 58. CPMUG Volume 39 is another music volume, as is PC/Blue Volume 60. On SIG/M Volume 120 we have a music composer for the H-89 or any GI AY38910 sound chip board.

Want to run SUBMIT multiple times? There is a 1K program and a 1K doc file on SIG/M 72 to tell you how. The program is called REPEAT.COM, and you include it in your submit file as REPEAT n. It then goes through the submit file n times. Simple and obvious, but handy!

How about an Econometric Model to test alternate money supply policies. Isn't that the height of obscurity? Not at all, if you understand economic theory. If you do, take a look at THE FED on SIG/M Volume 87. There are a number of statistical programs in the libraries. PC/Blue has the EPISTAT statistical package on Volume 59. SIG/M has them scattered through the library in too many places to list here, so check the catalog.

Typing, Railroads, & Energy Design

The library contains everything from learning to touch type (SIG/M 83) to calculating Canadian mortgage payments (115). There is a two volume Building Energy Design Analysis Program (SIG/M 137-138). WOOF.COM will check the frequency response of loudspeaker enclosures. While checking your loudspeaker you can also test your drives if you have a 1793 controller, or do gain and phase analysis of a ladder network (all three on SIG/M 152).

You can even attach your Atari to a CP/M machine (SIG/M 157 and 171). If not that, we have a Model Railroad Traffic System on SIG/M 182 and a way to install Atari joysticks on an Osborne on SIG/M 193. More! For those looking for curve fitting programs, we have the fluff minimax algorithm and the simplex algorithm on Volume 195, or multidisk formats for the Kaypro 2, 4, and 10 on 212. Even biochemical engineering games — on SIG/M 225!

Special Software For Special Needs

I still use my TRS-80 with Omikron Mapper and 8" drives to copy disks. Back on SIG/M Volume 86 we released a full volume of software just for that system. We have a Versafloppy II double density BIOS on Volume 26, CDOS support on Volume 41, SD Sales Hard Disk Support on 42, CP/M To Helios and vice versa on 44, DTC Hard Disk Bios Support on Volume 50, Big Board Utilities on 136 and 141, a Disk IO driver in CP/M+ for the Versafloppy 2, and a CP/M+ Disk IO module for the XCOMP Hard Disk on SIG/M Volume 172.

I hope I've made two points with this trivia. Lots of people solve rather specialized problems in software and allow the libraries to make these programs available to all. You stand a good chance of finding a solution by a detailed browse through either the printed or disk catalogs. On the other hand, if you've done such a job in software, how about putting it in the library? Why not help someone else avoid reinventing the wheel.

Still Rolling Along

Last time I looked, PC/Blue was getting near a 150 volume size and growing rapidly. A lot of it is from the dreaded freeware beggars with their hands out and their source code omitted. At a diner the other day, we were having coffee and battling over the

(continued on page 15)

(continued from page 13)

freeware issue. The basic argument is that freeware is necessary so the little guys can market software. Their contention is that it costs too much to put a product on the market; therefore, to promote software development, we should encourage freeware.

Sure, it takes big bucks to market a Lotus or a dBASE, or to bring a major new commercial package to market. That kind of package doesn't get to the public as freeware. Most freeware is cottage industry. I just wrote a 300+ K Clipper compiled program to run a friend's video store. A couple of outfits have asked me about marketing it, because it probably is the most comprehensive package around. They could sell it for about $250. But before you read the next paragraph, I suggest you sit down.

I may put VIDEO MANAGER out as freeware! The big difference is this — if freeware is designed to help cottage industry, then why not handle it like ASCAP. Why not put out the disks with a single request for payment to a centralized point, the not-for-profit public domain library. It would then distribute the payments to all the authors in the library according to an ASCAP type formula. It would also serve as a confidential repository for the source code. Let me know if you think it's an idea whose time has come.

Editor's note: The word I'm getting is that some of the original freeware people have started dropping out of that market. At first it looked good, but now it appears that few are getting more than occasional token payments and many are spending hours upon hours supporting (nonpaying) users.

Also, you face some special support requirements when you release a business oriented package.

Any video store which uses your program to handle its business is going to be on the phone to you the instant the program so much as twitches (be it hardware, software, or operator). A store is a livelihood for a number of people, and even if they've paid nothing, they're going need support if something happens.

A software outfit in Seattle wrote and released an accounting package a couple of years ago. It wasn't a fancy package, but it worked. About a year into the project they offered full copying rights to any dealer for the price of a single copy. The dealer could make as many copies as he wanted, as long as he wanted, and sell them for whatever he wanted. The only stipulation was that the dealer had to handle all the support. He couldn't tell anyone who had written the package, and he couldn't call the company for help.

Perhaps your experience will be different. I hope so.

New PC/Blue Releases

PC/Blue 140 and 143 contain terminal emulation and host communications programs. PC TALK III is also on 143. There is a thing called JUDY, a background desktop utility on 140, which Hank Kee raves about, but which I have not tried yet. Apparently Judy nags you until you do what you have to. PC/Blue Volume 142 has some handy utilities including VFILER for the PC. Rich Conn wrote VFILER, and SIG/M released it for CP/M 80 on Volume 145. Harry Van Tassell translated it for CP/M 86 on SIG/M Volume 145. It is the best screen oriented file manipulator around. The PC crowd has now translated it. Thank heavens they're not asking for a handout, but they did hold back the source code.

PC/Blue is very good at compiling volumes of how to do things. Figure 1 shows the articles contained on the latest information volume, PC/Blue 145.

123TODB3.TXT	convert 1-2-3 formats to dBase III
ASMTUTOR.TXT	Macro Assembler tutorial
DB3FIXPO.LQR	dBase III fixes
DB3TO123.TXT	converts dBase III format to 1-2-3
DOUBLEDO.UNP	extending DoubleDOS v1.0
MB640K .DOC	extending XT systemsboard to 640KB
MULTIMAT.FIX	Multimate fixes
TPTUTOR1.LBR	Turbo Pascal tutorial
TPTUTOR2.LBR	/
WSPATCHS.TXT	WordStar patches

The Ongoing Protocol Wars

Last issue we noted that SIG/M Volume 241 has on it the official newly revised XMODEM protocol which supports optional 1K blocks for those who use the new 2400 baud modems. On the same disk, Ron Fowler updated MEX to the same 1K standard. We warned you to beware because at least one member of the three-revision-a-day crowd was trying to sabotage the standard (because he can't count to 16 bits). The warning still holds. Stick with MEX for your modem program, and watch the version numbers on XMODEM. You may regret it otherwise.

Trenton Computer Festival

The big East Coast non-commercial computer event is the Trenton Computer Festival, sponsored by the Amateur Computer Group of New Jersey, Trenton State College, and just about all of the East Coast computer clubs. We get 15,000 people over a two day period, have acres of flea market and commercial exhibits, etc. This is its 11th year, making TCF the oldest of the festivals. (The only one that preceded it was the Altair World Festival, which no longer exists.)

Editor's note: Sorry for interrupting again, but the flea market alone is well worth the trip. It's a great event!

Speakers are needed for the Trenton Festival (April 19 and 20, 1986). If you would like to be on the program, please drop me a note and I will get the information to you. While April seems far off, the program will be finalized before the year is over.

Next issue we are going to cover printing and text formatting the public domain way.

SIG/M volumes are available on 8" SSSD disks for $6 each ($9 foreign) directly from SIG/M, Box 97, Iselin, NJ 08830. Printed catalogs are $3 each ($4 foreign). A new version of the catalog should be back from the printers in short order. Disks in a variety of formats may be obtained through the worldwide SIG/M distribution network. The distributor list is included with the printed catalog. A disk version of the catalog (Volume 00) is available for $6.

PC/Blue volumes are $7 each ($10 foreign). The printed catalog is $5. Both are available from the New York Amateur Computer Club, Box 100, Church Street Station, New York, NY 10008.

■ ■ ■

Private Domain, An Alternative To Public Domain

By David Thompson

I've always thought that software could be inexpensive. After all, how much does it cost for a disk and postage? A couple of dollars, maybe. Of course, there are also the fancy, printed manuals with fake leather bindings, boxes with gold embossed logos, and dealer markups.

Well, even without all the fancy trappings, software isn't cheap to sell (see this issue's "On Your Own"). After paying for advertising (Profiles, Morrow Owner's Review, Micro C), catalogs, disks, boxes, postage, people, space, phone, returned packages, testing and support (half of two people's time) we're barely breaking even on the public domain disks at $12 each.

In fact, there doesn't seem to be any way for an individual with a program to make it unless his program becomes very popular (e.g., Turbo Pascal) or he can charge a bundle for each copy.

So, for the past year I've been trying to come up with a way to centralize the distribution of inexpensive, commercial software so programmers can make money, we can make money, and the purchaser can get good software at a great price.

Private Domain

As I envision it Private Domain software will cost $19.95 per (single disk) package. A package could be one program or a disk full.

Of that $19.95, $3 goes to the author(s) for the program and documentation (plus an additional 50 cents per package if the author lets us distribute source). Another 50 cents per disk goes into a support fund. (There is a separate support fund for each package.)

Every six months the support funds are disbursed to those who did bug fixes or upgrades to the software or documentation. If there are no fixes or mods, or if the author makes all the changes, then the author receives the entire amount. Otherwise, those (including the author) who made significant contributions to the package during the six-month period share the accumulated amount.

If the author does not release source to the public (he must send a copy of the source to Micro C) and doesn't choose to do upgrades or fixes, then Micro C will use the accumulated funds to pay for fixes to the code and documentation.

The balance of the $19.95 goes to cover the expenses of marketing, distributing, and supporting the software. Maybe there'll even be some left for Micro C.

Manuals

The manual for most programs will be on disk, but we'll also print some initial instructions and a quick reference card for every product. We are aware that some packages (compilers, fancy editors, assemblers, etc.) require a formal, printed manual. When this is the case, we'll charge an additional $3 to $10 for the manual (depending on the cost of printing, postage, and maintenance).

Disclaimer

As far as we know, these numbers and procedures should be realistic and workable. But we may find out for some reason that they aren't, so we may have to change things as we go.

Upgrades

This is a real problem. I don't like charging to correct bugs because it doesn't seem fair for those who wait a few months before ordering to get a better product than those who order right away. On the other hand, a policy of free upgrades would make any company think a long time before updating a package (and going out of business in the process).

If a product has a debilitating bug, then we'll update for free. Otherwise, updates are half price for registered owners. (Registered owners are also the only ones who can receive cash for updates.)

What We're Looking For

Dave Pogue said it best. What we're looking for are things that are so good that you know immediately they are commercial software.

We're looking for assemblers, compilers, packages like REC and MEX, essential utilities (like unerase), games with graphics and sound, editors, really good NSWEEP type programs, and database languages (or Turbo applications). Plus, libraries (C & Pascal), training disks (languages), spreadsheet templates, business applications, scientific applications, statistical applications, etc.

We don't want something you've pulled off a bulletin board or gotten from a friend. That's public domain material.

This, however, is commercial, copyrighted material which we must receive from the author. It must be material that has never been distributed without a copyright, but it doesn't have to be a new, unmarketed product.

The obvious target systems are the PC, CP/M (maybe), and the Amiga.

Lots Of Packages

The key to being able to sell software at $19.95 per package is lots of packages. Twenty or thirty packages not only make the advertising more effective (people are more likely to find at least one thing they want) but it also encourages repeat orders. (More packages mean more orders. More orders mean we distribute more catalogs which lead to more orders.)

What You Can Do

If you have an idea or a program that you think would work well in the private domain, drop us a note or give us a call. Ask for Larry Fogg. He'll be heading up this project.

If your idea sounds good, we'll work out a course of action. But let me warn you, this is going to be a class act, commercial quality all the way (at LEAST commercial quality). If it isn't absolutely solid and absolutely right then it isn't absolutely ready.

Some Sales Projections (Guesses)

Here's where we get to make fools of ourselves. But based on our experience with $12 public domain disks, I'm projecting an order rate of between 50 and 300 units per month for each MS-DOS package and between 5 and 25 units per month for CP/M (and dropping rapidly). As we become known and as we advertise more, the number of MS-DOS products shipped could take off, but I'll stay with these figures for now.

Utilities, typing tutors, games, and modem software lead the way in the public domain sector (in that order). Spreadsheets, sidekick style programs, database handlers, and compilers, should also do very well, but really first rate commercial versions aren't available yet in this price range.

■ ■ ■

C'ing Clearly

By Ron Miller

1157 Ellison Drive
Pensacola FL 32503

Has programming left you out of sorts? Are you being bombarded by disorderly information? Don't get left out in the shuffle. Ron is here to help you get your sorted affairs in order.

An amateur programmer confronting a set of numbers to arrange or a herd of names to alphabetize faces an embarrassment of riches. There are shell sorts, insertion sorts, quicksorts, merge sorts, selection sorts, and who knows how many others. Which algorithm is the fastest? Which most compact? And perhaps most important of all: Which can be translated from BASIC (the sample at hand almost always turns out to be in pidgin BASIC) into an honest language?

It's One Sort Or Another

The speed and flexibility of C make it possible to get by with two sorts — one from each end of the spectrum, stretching from the simplest to the most sophisticated. If either bubble sort or quicksort won't do the job, you probably didn't need to sort it after all.

Bubble Sort

For quick and dirty efforts, I reach first for bubble sort because it's so straightforward that even the absent-minded among us don't have to look it up while hacking away. The idea is simplicity and inefficiency itself: Zero a flag and run through the array, exchanging adjacent pairs when they are out of order. Set the flag whenever an exchange takes place. Rezero the flag and keep looping until the flag isn't set. For an array of integers of LENGTH items beginning at BASE, bubble sort written in C is shown in Figure 1.

A swap routine with a temporary variable can be fashioned anew for the type variable being swapped. Or better, as shown here, you can create a more general swap function by passing the addresses and the length of variables.

Figure 4 contains a speedy Z80 assembly language routine that keeps the bytes on the registers and thus avoids all temporary storage areas. As you can see, the bubble sort algorithm is so brief that it can easily be retyped when the comparison is not a matter of integers but of strings. The first "if" statement would simply become:

```
{ if(strcmp(s1,s2)>0){ . . . }
```

Bubble sort remains my choice whenever I'm not playing fancy games with structures (see below) and not sorting more than about 50 items. I wanted to see just how long it takes to bubble sort a set of random integers, so I made repeated runs on my 5MHz Kaypro, using Software Toolworks' C/80. Sorting 10 items averaged 0.0123 seconds per sort. With 100 items, the average time rose to 1.54 seconds. I can live with that.

If I were using BASIC, the choice would be different. In MicroSoft BASIC at 5MHz the rates were 1.1 seconds (as opposed to .012 seconds) for sets of ten, and 106 seconds (as opposed to 1.5 seconds) for sets of 100. One minute and 46 seconds is quite a pause while waiting for somebody to shuffle. Interestingly enough, it doesn't seem to matter whether the integers are stored as ordinary numbers (i.e., floats) by default or as two-byte elements by invoking DEFINT. The speed remains the same.

Quicksort

Even with the speed of C, the situation becomes absurd when a bubble sort takes on an array of much more than a hundred. I tried sorting 800 hex (2048 decimal) random integers and found that an average of 11 minutes and 15 seconds elapsed before my timing circuit was triggered. If you think that's bad, consider the result for a 800H array of numbers in BASIC:

Figure 1 - Bubble Sort in C

```
main()
{
  bsort(BASE,LENGTH);
}
/********/
bsort(init,count)
  int *init,count;
{
  int  j;
  char flag;

  for(j=flag=0,count-=2;;){ /*keeping down subtractions: see test below */
    if(init[j]>init[j+1]){
      swap(&init[j],&init[j+1],2); /* see below on swap formats */
      flag=1;
      }/*if all pairs compared and flag not set, quit. Otherwise, repeat.*/
    if(j++ == count) if(!flag) break; else j=flag=0;
    }
}
```

Figure 2 - The Quicksort Version

```
qsort(init,count)
  int *init,count;
{
  int  j,k,pivot;

  if(count>2){    /* If three or more, must be subdivided. */
   j = 0;
   k = count - 1;
   pivot = init[(count-1)/2];  /* Choose the value of the midmost element.*/
   do{                         /* Any member would do, however. */
    while(init[j]<pivot) j =(j<count-1) ? ++j : count-1;
    while(init[k]>pivot) k =(k>0) ? --k : 0;  /* To prevent index overrun */
    if( j<=k ){
      swap(&init[k],&init[j],2);
      j =(j<count-1) ? ++j : count-1;
      k =(k>0)       ? --k : 0;
      }
    }while(j<=k);
   qsort(&init[0],k+1);          /* recursion */
   qsort(&init[k+1],count-(k+1));
   }
   else   /* If length=one, leave it alone.  If two, flip if necessary. */
     if( count==2 && (init[0]>init[1]) ) swap(&init[0],&init[1],2);
}
```

13 hours and 29 minutes. Over a DAY, if my Kaypro still ran at 2.5MHz! That's one result I didn't replicate: I suffered a mild case of computer withdrawal while waiting for BASIC.

With C and quicksort, that same 800H integers took 4.25 seconds.

The general idea of quicksort is slightly more complicated than that of bubble sort: Choose a "pivot" value and divide the array into an "upper" section in which all values are above the pivot, and a "lower" section in which everything is below. Then apply the process recursively upon the resultant sections. Continue until the portions are reduced to single-member cells.

Unfortunately, this process requires some complex looping and indexing which makes it difficult to write off the top of your head. You also need a recursive language, which ends our comparisons with BASIC. Figure 2 shows the quicksort version.

You can see that quicksort is faster than bubble sort for large arrays, not because the algorithm itself takes less time to execute, but because the binary divide-and-conquer method leads to far fewer calls. Thus, for small arrays, bubble sort is actually faster. But how small? For ten random integers, quicksort wins the race: 0.0084 seconds versus 0.012 seconds. But for five integers, quicksort averages 0.0032 seconds to bubble sort's 0.0025. With these times, convenience and code size are the only considerations.

Generalizing The Algorithm

The relative complexity of the quicksort algorithm suggests that you will want to summon the code from a library, rather than rewrite it each time. That, however, complicates matters further, because the code in Figure 2 works only for integers. You're going to add some extra code to handle string pivots and string comparisons. Complex data types require a total redesign.

In Pascal, these difficulties are crippling. Quicksort must be recoded each time it's employed. But not in C. The ability of C to pass function addresses to other functions, plus the ability to calculate pointers, permits the construction of an all-purpose algorithm that works with any type of element, from single characters to complex structures. All you have to know is: (a) how you wish to compare the items in the array and (b) where within the items the fields to be compared are located.

In the version below, "offset" is the offset of the comparison field within the array element, and "compare" is the address of a comparison function which takes two addresses for its arguments and returns -1 if the element at the first address is smaller than the element at the second, 0 if they are identical, and +1 if they are out of order. To compare integers, for instance, you would need to write:

```
intcomp(s1,s2)
  int *s1,*s2;
{
  if(*s1<*s2) return(-1);
  else if(*s1==*s2) return(0);
  else return(1);
/* or, more succinctly,
 if not more lucidly, return (a==b)
 ? 0 :  (  (a<b) ? -1 : 1 );
 If you want a descending order,
 just multiply by -1. */
}
```

Even if this couldn't be stored in a library, I'd rather set up a comparison function again than re-enter the entire quicksort routine. Fortunately, the standard library strcmp() function returns of the proper -1,0,+1 signals already. (Incidentally, in C, the name of a function by itself is treated as the address of the function — just as with arrays.) Anyhow, the function is shown in Figure 3.

The only new twist here besides the employment of the function pointer is the use of one of the array elements itself as the pivot. Thus, it isn't necessary to allocate space for a test element for each call. Nonetheless, things get tricky because the sort will consequently move the test element. So I test for movement with the extra 'u' and 'v' to keep track of the wandering pivot (see Figure 3).

Unfortunately the added indirection of this general function slows down the simple integer quicksort by a factor of four. 800H integers take 16.4 seconds to sort, but that still beats the BASIC routine's 13.5 hours.

The power of this algorithm shows up when you're sorting complex data structures. If you can stuff the database into memory, you can sort on any field in any way your little heart desires, and at speeds that could awaken dBASE programmers. Even if

Figure 3 - strcmp() function

```
qsort(init,offset,size,count,compare)
  char *init;      /*arbitrary: char makes pointer arithmetic simplest. */
  int size,count,offset,(*compare)();  /* function address */
{
  int  j,k,u,v;
  char *pivot;

 if(count>2){
  j = 0;
  k = count - 1;
  pivot = init+size*((count-1)/2);
           /* pivot item begins at size*( (count-1)/2)  */
  do{
    while( (u=(*compare)(init + offset + j*size,pivot + offset )) < 0)
      j =(j<count-1) ? ++j : count-1;
    while( (v=(*compare)(init + offset + k*size,pivot + offset)) > 0)
      k =(k>0)       ? --k : 0;
    if( j<=k ){
      swap(init + k*size, init + j*size,size);
      if(!u) pivot = init + k*size; /*moved the test item if u or v == 0*/
      if(!v) pivot = init + j*size;
      j =(j<count-1) ? ++j : count-1;
      k =(k>0)       ? --k : 0;
      }
   }while(j<=k);
  qsort(init,offset,size,k+1,compare);
  qsort(init + size*(k+1),offset,size,count-(k+1),compare);
  }
 else
  if( count==2 && (*compare)(init+offset,init+size+offset) > 0 )
           swap(init,init+size,size);
}
```

(continued next page)

(continued from page 19)

you can't put the whole thing into memory at once, parts can be sorted and the resultant files merged. But that's another story.

Appendix

Figure 4 shows a speedy generalized swap routine for assembly language junkies.

Given the way implementations of C are usually written, integers, which fit nicely on the HL register, are the most efficient objects to exchange with equals signs. Yet even with integers, the following routine outpaces the "temporary variable" technique. You know:

```
temp=item1;
item1=item2;
item2=temp;  . . .
```

For larger variable types, swap() really sings. The Z80 string transfer routines do have their uses. Moral? For speed, stay out of the run-time package.

Figure 4 - Generalized Swap Routine in Assembly Language

```
swap(s1,s2,d)
   char *s1,*s2;  /* pointers to the two variables being swapped */
   int  d;        /* length of variables in bytes */
{
#asm
        DB      08H     ;Z80   EX AF,AF': save flags for insurance
        POP     PSW     ;save return address
        POP     B       ;length in BC
        POP     D       ;one address into DE
        POP     H       ;other into HL
        PUSH    H
        PUSH    D
        PUSH    B
        PUSH    PSW     ;return address back to top of stack
        DB      08H     ;AF<->AF': flags restored
LOOP:   LDAX    D       ;(DE) into A
        DB      0EDH,0A0H ;Z80 LDI
        DCX     H       ;LDI increments HL as well as DE, so back up
        MOV     M,A     ;A into (HL)
        INX     H       ;now catch up
        JPE     LOOP    ;loop until BC=0
;I'm curious: Can anyone do this loop in fewer machine cycles? On the
;8088 you can use an extra index via "MOVSB" and the
;back-and-forth can be avoided.
        RET
#endasm
}
```

■ ■ ■

RETURN TO 19,200 BAUD . . .

The following "set–serial" routines for the Kaypro and COM1 on an IBM or compatible were inadvertently omitted from Ron's last article in Issue 26.

```
set_serial()    /* KAYPRO VERSION */
{
    outb(0x18,6);                /*reset control port--#6*/
    outb(0xf,0);                 /* 19200 baud-- baud port=#0 */
    outb(0x4,6);                 /*bit set control*/
    outb(0x44,6);                /*one stop bit, 16x clock*/
    outb(0x03,6);                /*receiver logic control*/
    outb(0x1|0xc0,6);            /* bits read, no auto*/
    outb(0x45,6);                /*sender logic control*/
    outb(0x8|0x60|0x80,6);       /*8 bits send, assert dtr*/
    outb(0x01,6);                /*interrupt control*/
    outb(0x00,6);                /*interrupts disabled*/
}
set_serial()       /*  IBM compatible version for COM1*/
{
    outw(0x3|0x80,1019);       /*8 bits, no parity, one stop bit, no shake */
    outw(0x6,1016);      /*19,200 baud, in spite of what Peter Norton says*/
    outw(0x0,1017);
    outw(0x3,1019);            /*dtr low */
    outw(0x0,1017);
    outw(0x0,1020);            /* no interrupts */
}



-----------------FIGURE XXX------------------------------------------------
#include (command.c)        /*C/80 handles command line expansion this way */
#define TRUE 1
#define FALSE 0
#define S_PORT 4            /*the serial port of Kaypro; 1016 for IBM */
#define BFSIZE 0xa000
#define SHAKE 0xbb           /* or some other unlikely character */
#define EOF -1               /* "Tradition---", as Teyve sings */
#define EOT 0x04             /* ditto */

char inb(),filename[15];
struct {                     /* To provide a global buffer without */
  unsigned index,limit;      /* having 40K worth of fill in the COM file */
  char cell[BFSIZE];
  } *buffer;
int fdes;
/****************************/
_putc(b)                      /* 40K version of standard putc()  */
  int b;
{
  buffer->cell[buffer->index++]=b;
  if(!(buffer->index & 0x7f))
      if(buffer->index & 0x100) puts("\r    \r");  else puts("\rXXX\r");
  /*  I like it to blink when it's running. Every sector flashes on or off. */
  if(buffer->index == BFSIZE){               /*if the buffer fills */
     write(fdes,&buffer->cell[0],BFSIZE);
     buffer->index=0;
     }
  return b;
}
/****************************/
_getc()                     /* 40 K version of standard getc() */
{
  int b;

  if(buffer->index != buffer->limit){
/* then it's neither EOF nor time to refresh buffer */
    if(!(buffer->index & 0x7f))
       if(buffer->index & 0x100) puts("\r    \r");  else puts("\rXXX\r");
    return(buffer->cell[buffer->index++] & 0x7f);     /* no sign extension */
    }
  if(buffer->index != BFSIZE-1) return(EOF);
  /* If it's not A000H, I've reached the end of file */
  b=buffer->cell[BFSIZE-1];
  buffer->limit=read(fdes,buffer->cell,BFSIZE)-1;
  buffer->index=0;
  return(b & 0x7f);        /* no sign extension */
}
/****************************/
kbhit(       /* boolean for character waiting at keyboard */
{           /* Used to get out of endless loop. It's interrupt 21H,
             service 11 in MSDOS.  */
return bdos(11,0);
}
/****************************/
set_serial()
{
  See listings above.
}
/****************************/
char byte_in()
{   /* 6 is the Kaypro control port: 1021 for IBM;
       1 is character-waiting on BOTH machines */
    while( !(inb(6) & 1) ) if(kbhit()) exit(0);
    return(inb(S_PORT) );
}
```

```
/****************************/
byte_out(b)
 char b;
{
  /* character-can-be-sent is 4 on Kaypro, 0x20 on IBM  */
   while( !(inb(6) & 4)) if(kbhit()) exit(0);
   outb(b,S_PORT);
   return b;
}
/****************************/
puts(s)    /* the poor man's printf() */
 char *s
{
 while(*s) *s++;
}
/****************************/
send()
{
  int byte;
  char *charptr;

  if( !(fdes=fopen(filename,"rb"))){
    puts("\nFILE ERROR.\n");
    exit();
    }
  buffer-)index=0;   /*reading up to 0A000H bytes into the buffer */
  buffer-)limit=read(fdes,buffer-)cell,BFSIZE)-1;
       /* Since arrays in C start with 0 */
  puts("\nSENDING -- ");  /* I didn't want to use printf and get */
  puts(filename);         /*all that extra code */
  puts(" ==) \n");
  charptr=filename;
  do{
     byte_in(byte_out(*charptr));  /*Here's the file name.*/
     }while(*charptr++);     /*until null */
  while( (byte=_getc()) != EOF) byte_in(byte_out(byte));
    /* See the handshaking? Exchange goes on until index goes past EOF */
  for(byte=0;byte(8;byte++) byte_in(byte_out(EOT));
    /*Eight EOT's sent out at EOF*/
  fclose(fdes);
  putchar('\n');
  puts(filename);
  puts(" CLOSED.\7\7\7\7\n");
}
/****************************/
receive()
{
 int byte,endct;
 char *charptr;

 puts("RECEIVING -- ");
 for(charptr=filename;;){    /* filling in file name */
  if( (*charptr = byte_out(byte_in()))==EOT) exit(0); /*EOT means over */
  if(!*charptr) break;     /*if null, end of name */
  if(*charptr)32 && *charptr(127) charptr++;  /* Just an extra precaution */
  }
 puts(filename);
 puts(" ==) \n");
 if( !(fdes=fopen(filename,"wb"))){
    puts("\nFILE ERROR.\n");
    exit(0);
    }
 buffer-)index=0;
 for(endct=0;endct(8;){  /*receive until 8 EOT's in a row */
  if( (  byte=_putc(byte_in())  )  == EOT) endct++;
  if(endct && (byte!=EOT) ) endct=0;
  byte_out(SHAKE);
  }
 buffer-)index -= 8; /* Those EOT's weren't part of the file */
 write(fdes,&buffer-)cell[0],
    0x80*(buffer-)index/0x80 + !!(buffer-)index%0x80)) );
 fclose(fdes);
 putchar('\n');
 puts(filename);
 puts(" CLOSED.\7\7\7\7\n");
}
/****************************/
main(argc,argv)
 int argc;
 char **argv;
{
 int i;

 command(&argc,&argv);
/*Expanding command line in C/80. MSDOS 2.0 does it for you. */
 set_serial();     /*initialize ports */
 buffer=alloc(BFSIZE);     /*allocate buffer */
 if(argc==1){       /* If no arguments, then go to receive mode */
  puts("READY...\n");
  while(byte_in() != SHAKE)  if(kbhit()) exit(0);
    /* waiting for the other computer to come on line */
  while(TRUE) receive();  /* program exit is from receive() routine */
  }
 byte_out(SHAKE);  /* "I'm here," the donor says */


 for(i=0;i(8000;i++);  /* A little delay to let receiver get there first */
 for(i=1;i(argc;i++){
    strcpy(filename,argv[i]);
    send();
    }
 byte_in(byte_out(EOT));  /* T-t-t-that's all, f-f-f-folks!! */
}
```

Inside The Small C Compiler

By F.A. Scacchitti

25 Glenview Lane
Rochester NY 14604

Small C has been bouncing around the public domain community for several years now. It has been the root of many of the commercial Cs, and there's no doubt that the public domain has been improved by its presence. This is truly a cook's tour of a very famous compiler.

Any programmer can tell you — compilers are a mystery fit for the talents of a Holmes, a Wolfe, or a Marlowe.

A programmer might wonder why the minimum program size is 12K or where the stack is located. If he's lucky, his questions regarding compiler operation are answered in the documentation. If he's not, he might wonder why he became a programmer.

In order to clear up some of the mystery of Small C, I'll discuss the steps which occur between the start of compilation through the production of the COM file. In the process we'll move from CP/M into the program and back to CP/M, outlining the sequences a compiler takes to translate a source file of ASCII text into something a processor will understand.

I hope this discussion of my version of the Small C compiler will be helpful, and you can use this general information for your specific application. (Small C is available in the public domain for the 8080, 8088, and the DSI-32.)

C Function Library

One of the most important components of the C compiler is the C function library and its runtime modules — CLIB.REL and the module ULINK.REL. Its structure and contents determine how the program will move from the CP/M environment, through the program activities, and back to CP/M.

Let's assume we're dealing with a source file called SOURCE.C. The distribution disks (Micro C K35 and K36) contain information on running the compiler and on instruction syntax. The disks also include example programs and submit files for producing .MAC, .REL, or .COM files.

(Editor's note: Disks K7 & K8 contain a version of the Small C compiler whose output can be assembled with the ASM assembler. If you can't afford MicroSoft's M80, then this is a good option, but the assembly times are longer and the resulting programs are larger. The version of Small C on K35 and K36 is much nicer to use and is the one discussed in this article. The 8088 and DSI-32 versions should be very similar.)

What The Compiler Does

The CP/M version of the Small C compiler produces an output file of 8080 assembly language mnemonics formatted for use with MicroSoft's M80 Macro Assembler.

The first statement generated by the compiler is JMP CC1 followed by static variable defines (DW, DB, DS). Next comes the program code which starts at label CC1: and continues until all code is generated. After the program code, external defines are generated for calls not found within the program. If the program contains a main(), ULINK is declared as an external define and the variable ZZZCCP is generated. Listing 1 shows a mixed listing output example of the compiler.

Compilation creates a .MAC file. This file is then assembled with M80 to produce a .REL file. The .REL file contains a table of undefined externals which will have to be satisfied by the loader for proper operation.

Finally, SOURCE.REL is loaded using L80, and the CLIB.REL library is searched for modules to satisfy the external defines. As they're found the code is added to the "almost .COM file" and the entry addresses established. If any entry point isn't satisfied, L80 displays the error. Since L80 searches only once and in the forward direction, the library must be constructed with all modules forward referencing. Check LIB80, the Small C DOC files, and the L80 manuals for more details.

Neither the compiler nor M80 produces any module with a program starting point recognized by L80. The only module in the library that has one is ULINK. If the C source file

(continued next page)

Listing 1 - Sample Mixed Program

```
;/*
;** test.c       Test C Program
;**
;**              Written in Small-C
;**              Version 2.10 or later
;**
;*/
        JMP CC1
;
;
;char data[10], dummy;
DATA::
        DS   10
DUMMY::
        DS   1
;int counter, sum;
COUNTER::
        DS   2
SUM::
        DS   2
;
;main(argc,argv) int argc, argv[]; {
CC1:
MAIN::
;
;   counter = 1000;
        LXI H,1000
        SHLD COUNTER
;   sum = counter * 10;
        LHLD COUNTER
        XCHG;;
        LXI H,10
        CALL CCMULT##
        SHLD SUM
;
;   printf("This is a test program\n");
        LXI H,CC2+0
        PUSH H
        MVI A,1
        CALL PRINTF
        POP B
;
;   delay(100);
        LXI H,100
        PUSH H
        MVI A,1
        CALL DELAY
        POP B
;
;   puts("It doesnt do much at all\n");
        LXI H,CC2+24
        PUSH H
        MVI A,1
        CALL PUTS
        POP B


;
;}
        RET
CC2:    DB 84,104,105,115,32,105,115
        DB 32,97,32,116,101,115,116
        DB 32,112,114,111,103,114
        DB 97,109,13,0,73,116,32,100
        DB 111,101,115,110,116,32,100
        DB 111,32,109,117,99,104,32,97
        DB 116,32,97,108,108,13,0
;
        EXTRN PRINTF
        EXTRN PUTS
        EXTRN DELAY
        EXTRN ULINK
        ZZZCCP::  DB  0
        END
```

End of Listing

(continued from page 23)

contains a main() function, ULINK will automatically be defined as an external entry point and loaded by L80 when searching the CLIB.REL library. This forces L80 to generate a jump to the start of the ULINK module as its first three bytes of code. ULINK defines three additional externals, MAIN, ZZBUF, and ZZZCCP.

The Three Externals

MAIN (entry point for the compiled program) is CALLed after ULINK performs all the preliminary operations. ZZBUF is physically and logically the last module in the library (therefore always loaded last), and consists simply of a NOP instruction that marks the beginning of memory storage outside of the program. ZZZCCP is a byte generated by the compiler and used to determine both the stack placement and return path to CP/M.

If ZZZCCP is true, the stack moves to the base of the CCP and returns to CP/M via a RET instruction (having left the CCP intact). This allows for much faster operation of programs at a cost of 800 bytes of stack space. If ZZZCCP is false, the stack moves to the base of the BDOS, the return is performed via a JMP 0, and CP/M is warm booted. Use the -n switch during compilation to make ZZZCCP true.

Listing #2 is the commented source for the module ULINK. It provides the details of program entry, CP/M environment retension, buffer establishment, argument passing, link to user code, and return to CP/M.

Compiled, ULINK is less than 256 bytes so COM files can be quite small. If redirectable I/O were added the minimum file size would increase by as much as 5K.

What ZZBUF Does

Although small, the variable ZZBUF serves an important purpose. It's the last module in the library, and is always loaded last. It's always loaded if ULINK is loaded (remember ULINK defines ZZBUF as external) and marks the physical end of the program file. ULINK uses this point to set up a new

(continued on page 27)

Listing 2 - Source for the Module ULINK

```
; ulink
;
;               Small-C  System Library  Version 1.0
;
;                            by
;
;                       Fred A. Scacchitti
;                       25 Glenview Lane
;                       Roch., NY 14609
;
;                         11 - 24 - 84
;
;       This module is a derivative of RUNTIME.MAC by Glen Fisher
;       and Bill Randle. It contains a minimal implementation of
;       CP/M hooks to allow proper linking of Small-C programs
;       compiled by Version 2.08 (and later) of the Small-C compiler.
;
;       This module contains the following routines:
;
;       -ULINK entry point just like J. E. Hendrix Vers. 2.1
;
;       -Initialization of stack, memory buffers, and argument
;        passing (argv, argc).
;
;
;        call to main()
;              links to main in user program
;
;       -Cleans house and returns to CP/M
;
;
;
;
; Now then here's the starting code
;
;
;
; 1st - Save CPM's stack pointer, establish file i/o constructs set
;       new default buffer and establish start of available memory.
;
; 2nd - Get the default disk and set stack at base of BDOS or CCP
;       depending on the status ZZZCCP. 0 = bdos  1 = ccp
;
; 3rd - Parse the CPM input line and modify it so that we can pass
;       the C program  in the argc, argv form that it expects.
;               HL = pointer to next argv entry
;               DE = pointer to next character in command line
;               B  = number of characters left in line
;               C  = argument count (argc)
;
; 4th - Call MAIN to commence operation of the user program
;
; 5th - Work, work, work and then return to here via exit(), abort()
;       or normal return from program.
;
; 6th - Close any open files (buffers are not flushed)
;
; 7th - Restore CP/M's stack pointer, select the disk you entered
;       with, reset the default buffer and return to CP/M either
;       via a JMP 0 (ZZZCCP = 0) or RET (ZZZCCP = 1).
;
;
CBDOS   EQU     5       ;/* bdos entry point */
CPMARG  EQU     128     ;/* CP/M command line */
MAXARG  EQU     24      ;/* Maximum number of input args */
STDIN   EQU     0
STDOUT  EQU     1
STDERR  EQU     2
STDLST  EQU     4
CBDOS   EQU     5
CLOSE   EQU     16
;
FCBSIZE EQU     36      ;size, in bytes, of an FCB
BUFFER  EQU     6       ;offset to disk sector buf. in I/O struct.
UNGOT   EQU     5       ;offset to char ungotten by ungetc()
FLAG    EQU     33      ;file-type flag byte (in unused part of FCB)
FREEFLG EQU     128     ;This I/O structure is available
BUFSIZ  EQU     1024    ;how long the sector buffer is
NBUFS   EQU     8       ;number of I/O buffers
TBUFSZ  EQU     128     ;size of default disk buffer
;
EXTRN   MAIN, ZZBUF, ZZZCCP
```

```
;
;
ULINK::
;
        LXI     H,0             ; get CPM's stack pointer
        DAD     SP
        SHLD    ZZSTAK          ; save it for later
;
        MVI     C,26
        LXI     D,ZZBUF
        CALL    CBDOS           ; Set the default buffer out there
;
SETIO:
        MVI     B,NBUFS
        LXI     H,ZZBUF+TBUFSZ+FLAG
        LXI     D,FCBSIZE+BUFFER+BUFSIZ
        MVI     A,FREEFLG
SETIO2: MOV     M,A                     ;set all buffers to free
        DAD     D                       ;on to next buffer
        DCR     B
        JNZ     SETIO2                  ;if there is one...
        SHLD    ZZMEM                   ;put it where it belongs
;
;
;       anchors
;               away [sic]
;                       my
;                               bo
;                                 y
;                                  y
;                                   y
;                                    y
;                                    y
;
EXIT::
        MVI     B,NBUFS         ; Prepare to scan for open files
        MVI     C,CLOSE
        LXI     H,ZZBUF+TBUFSZ+FLAG
        LXI     D,FCBSIZE+BUFFER+BUFSIZ
SCANEM:
        MOV     A,M             ; Get free flag
        CPI     FREEFLG         ; Is it free ?
        JZ      NOPEN           ; Yes - not open, go get next
        PUSH    B               ; No  - prepare to close
        PUSH    D
        PUSH    H
        LXI     D,-FLAG
        DAD     D
        XCHG                    ; DE --> Start of FCB
        CALL    CBDOS           ; Close the file
        POP     H               ; Restore scan setup
        POP     D
        POP     B
NOPEN:
        DAD     D               ; Point to next flag
        DCR     B               ; Last one ?
        JNZ     SCANEM          ; No - keep scanning until done
;
        MVI     C,26
        LXI     D,80H
        CALL    CBDOS           ; Reset the default buffer
;
        LHLD    ZZSTAK          ; Load stack pointer
        SPHL
        LDA     ZZZCCP          ; How do we get back ?
        ORA     A
        RNZ                     ; this way to CCP
;
;                                       or
;
;                               this way to warm boot
;
        LDA     ZZDFLT          ; Grab orig. logged-in disk
        MOV     E,A
        DCR     E               ; (cvt. back to 0-n)
        MVI     C,14            ; and log it in again
        CALL    CBDOS           ; (mod to cbdos (fas))
        MVI     C,25            ; get logged-in disk
        CALL    CBDOS
        INR     A
        STA     ZZDFLT          ; save it
;
        LDA     CBDOS+2         ; get base of BDOS
        MOV     H,A             ; save page in HL
        MVI     L,0             ; where do we put the stack
;
        LDA     ZZZCCP          ; let's check
        ORA     A               ;
        JZ      SETSTK          ; just below the BDOS
        MOV     A,H
        SUI     8               ; just below the CCP
        MOV     H,A
;
SETSTK:
        SPHL                    ; set stack pointer
;
        MVI     C,0     ; Init argc
        LXI     H,ARGV  ; Pointer to first entry of argv array
;
; CPM does not tell us what the first word of  the command
; line was (the name of pgm), so we fake  it by pointing it
; to an ascii string with '*' in it
;
        LXI     D,PGM   ; Pointer to 'pgmname' string
        CALL    SVARG   ; Save next argument
;
; Ok, now for the real stuff.  Set DE pair to point to
; CPM command line and start searching for arguments

        LXI     D,CPMARG ; Pointer to CPM arg line
        LDAX    D       ; Load # character in line
        MOV     B,A     ; Save it in B
NXTSP:  INX     D       ; Point to next character
        DCR     B       ; Decrement character count
        JM      ENDCMD  ; End of cmd line
        LDAX    D       ; Load next character in line
        CPI     ' '     ; Space?
        JZ      NXTSP   ; Yes...continue searching
        CALL    SVARG   ; Nope, save starting point of this arg

; Loop looking for either end of line of a space

NXTCH:  INX     D       ; Point to next character
        DCR     B       ; Decrement character count
        JM      ENDWRD  ; End of cmd line, but need to end arg
        LDAX    D       ; Load next character in line
        CPI     ' '     ; Space?
        JNZ     NXTCH   ; Nope...keep looking
        MVI     A,0     ; Yes, replace it with a zero byte
        STAX    D
        JMP     NXTSP   ; Look for start of next arg
ENDWRD: MVI     A,0
        STAX    D
ENDCMD: MVI     B,0     ; Zero B (BC now is 16 bit argc)
        PUSH    B       ; First arg to main procedure
        LXI     H,ARGV  ; Point to argv array
        PUSH    H       ; Second argument to main procedure
        MVI     A,2     ; Load up the argument count
;
        CALL    MAIN    ; Transfer to the C world....
;
        JMP     0
;
;
SVARG:  MOV     M,E     ; Save pointer to start of string
        INX     H
        MOV     M,D
        INX     H
        INR     C       ; Increment argc
        RET
PGM:            DB      '*',0
;
ARGV:           DS      MAXARG*2
;
ZZDFLT::        DB      0       ; default disk
;
ZZSTAK::        DW      0       ; CP/M's stack
;
ZZMEM::         DW      0       ; useable memory pointer
;
ZZTEMP::        DW      0       ; available to anyone
;
;
;
        END     ULINK
```

(continued from page 24)

Figure 1 - Memory Map

```
Address          Code

 100        JMP  ULINK:

 103        User program - Static Variable Storage
                   followed by
----        User program code
                   followed by
----        ULINK:
                   followed by
----        Library Modules
                   ending with ZZZBUF
----        ZZZBUF: marks the physical end of program

      What follows is generated by the runtime module
      and exists outside of program code

----        128 byte new CPM default buffer

----        FCB #1             36 bytes
            File Construct      6 bytes
            File Buffer #1 1024 bytes

----        FCB #2             36 bytes
            File Construct      6 bytes
            File Buffer #2 1024 bytes
            .
            .
            .
----        FCB #8             36 bytes
            File Construct      6 bytes
            File Buffer #8 1024 bytes

----        Allocatable Memory
            .
            .
D400        CCP
            Stack at base of CCP if ZZZCCP = 1

DC06        BDOS
            Stack at base of BDOS if ZZZCCP = 0

EA03        BIOS

F000        Monitor

FFFF        End of Memory
```

default CP/M file buffer. (We want to preserve the original which may contain arguments passed to the program.)

The new default buffer is followed by eight 1K file buffers along with their FCBs and table info. Since none of this is set up until runtime, the program doesn't contain a lot of excess buffer space, which not only occupies valuable disk space, but also increases program loading time. I chose the number of file buffers and their size to suit my needs.

Following the file buffers lies a pointer used by the dynamic memory allocation/deallocation functions — calloc(), malloc(), free(), cfree(). The scheme employed for memory management must be handled with extreme caution. As blocks of memory are allocated, stack collision is checked, and if safe, a pointer is placed at the end of the block. When a block is deallocated, its memory (and all memory above it) is returned to the system.

A word of caution: be sure to use a first in last out scheme with these functions. A more sophisticated scheme could track all allocated and free memory, but usually at a significant code overhead.

The Memory Map (Figure 1) depicts how a typical C program would go together for a Xerox 820-II. Actual addresses for CCP, BDOS, BIOS, and Monitor vary with different systems.

One final comment: if you are compiling a function which doesn't contain a main(), none of the runtime code or variables are defined, and no external buffers are established.

Editor's note: If you're interested in learning more about the insides of Small C (or are interested in creating your own compiler) check out "The Small-C Handbook" by James E. Hendrix (we had trouble finding copies because it's almost out of print). It's clearly written and fun to read as well as a real education.

■ ■ ■

The Xerox 820 Column

By Mitchell Mlinar

The Z80 SIO is smarter than the Z80, and in many ways it's more difficult to deal with. In this issue Mitch programs the SIO to send and receive just about any type of asynchronous character ever devised. Next issue Mitch looks at the 8088-based PC clones that Xerox has begun unloading.

Xerox has been very quiet lately. The only interesting thing I've heard since the last column is that Xerox has expanded its support of 16-bit machines; there are several IBM clones that now offer full Xerox Americare service.

Service contracts on computer equipment are offered on all systems sold by Xerox and many that are not. My RQP/M (an 820-I) has suffered through some troubled hardware times with its 8" disk drives. I had them in Xerox service a few times, but neither Xerox nor I could find the problem. Finally, a replacement logic board did the trick. Considering what it could have cost me without the service contract, and the excellent turn-around (within 48 hours every time), I did not hesitate to renew my contract.

Z80-SIO

In previous segments, I examined the Z80 interrupt mechanism and CTC chip. This time, I will plunge into the most complicated support chip of the set — the Z80 SIO.

The SIO (Serial Input/Output) chip is a two-channel peripheral which is almost too versatile; the manual for this chip is a mere 48 pages long. Not only does the SIO support "simple" asynchronous communication, it also handles synchronous protocols such as the IBM bi-synchronous mode, HDLC, and IBM SDLC. (If you are versed in the last three terms, skip the rest of the article as you know more than I do.)

Most serial communication is asynchronous, although a notable exception are the packet radio users who employ HDLC. (I would encourage anyone who uses the 820 in packet radio to write an article about it.) I will only discuss asynchronous in this column.

Figure 1 - Connecting the Signal Set to COMM and PRINTER

Signal	Name	Z80 Direction	Pin Number for Port COMM	PRINTER
Tx	Transmit	Output	2	3
Rx	Receive	Input	3	2
CTS	Clear-to-send	Input	5	4
RTS	Request-to-send	Output	4	5
DTR	Data-Terminal-Ready	Output	20	8
DCD	Data-Carrier-Detect	Input	8	20
GND	Ground		7	7

Figure 2 - Three Write Registers

```
                      Write Register 3 (Receive)
Bit 7 6                   5               4 3 2 1 0
    0 1  7-bit Rx     x set to 1 for     0 0 0 0 1
    1 1  8-bit Rx       auto-enables

                       Write Register 4 (Mode)
Bit 7 6                 5 4   3 2               1 0
    0 1  x16 clock      0 0   0 1  1-stop bit   x 0 no parity
                              1 1  2-stop bits  0 1 odd parity
                                                1 1 even parity

                      Write Register 5 (Transmit)
Bit 7                  6 5            4 3 2   1                0
     x set to 1 to     0 1  7-bit Tx  0 1 0   x set to 1 to    0
       enable DTR      1 1  8-bit Tx            enable RTS
```

The architecture of the SIO offers a nearly full asynchronous signal set (often called Modem control) which are connected to the COMM (port A of the SIO) and PRINTER (port B) as shown in Figure 1.

DTE Vs. DCE

The COMMunications port is configured for DTE (Data Terminal Equipment) whereas the PRINTER port is configured as DCE (Data Communications Equipment). Notice the duality between the pins (2-3, 4-5, 8-20), which is why a straight-through cable to your modem (which is DCE) works fine from your COMM port but not from your PRINTER port. DTE must always connect to DCE (unless you swap wires in the cable). It is possible to change jumpers on the 820 board to make the COMM port DCE (the PRINTER port is fixed).

Part of the Z80's task upon power-up in the 820 is to initialize the serial port's baud rate, frame type, and mode. It is not complicated, merely confusing.

BAUD RATE — the speed of transfer in PBS (Bits-Per-Second) ranging from 110 to 19200. The baud rate is determined by a clock signal supplied by the COM 8116 chip; I'll cover that later.

FRAME TYPE — bit width (5-8), stop bit (1, 1.5, or 2), and parity (none, even, odd). Nearly all applications use either 7- or 8-bit transfer, 1 stop bit, and no or odd parity. Two types used on the the Xerox are:

8-bit 1-stop no-parity — nearly always used with modems (COMM). 7-bit 1-stop odd-parity — Diablo printer default (PRINTER).

MODE — the clock slice rate, set to x16 for all asynchronous applications.

Programming the Z80 SIO can be confusing as there are (gulp!) ten separate control registers inside the chip (3 input, 7 output) done through two ports. However, only a few need concern us. Z80 programming ports (in decimal) are as follows:

	COMM	PRINTER
Data	4	5
Control	6	7
Baud	0	12

By now, you might be wondering how 10 registers can be accessed using a single control port (I did). The SIO sneaks around this by requiring 2-bytes to program each register: the first is the register number, and the second the programming byte. After the second byte is sent, register 0 is automatically selected. (Hence, ye be properly warn'd to Disable Interrupts while initializing the SIO.) To change the mode and frame type you must write to three registers (a 6-byte transfer to the SIO control port). These registers are partially described in Figure 2 with bit 7 the high bit (or MSB) and bit 0 the low bit (or LSB).

(continued next page)

XEROX 820 COLUMN

(continued from page 29)

See, I said it was confusing! For example, you may notice that it is possible to set different bit widths for transmit and receive. In a word, DON'T! Two items need further explanation:

PARITY — the number of 1s (or 0s) in the byte. Even parity, for instance, means that the total number of 1s in the character (plus parity) will be even. If the character contains an odd number of 1s then the parity bit is set to 1 (to make the total even). If the character contains an even number of 1s then the parity bit is set to 0. If an error changes one bit in a character then the other end will detect the problem. In an error changes two bits then the problem won't be detected.

AUTO-ENABLE — a nice feature of the SIO which is often ignored, but can handle hardware handshake automatically. On transmit, the SIO checks the CTS line and will not transmit the character until CTS is true. Similarly, the SIO will not accept any character until DCD is also active. Thus, the usual (and simple) method of checking xmit/recv buffer status can still be used instead of having to check DCD and CTS in software.

Figure 3 - Programming the COMM and PRINTER Ports

```
COMM port (6): 8-bit  1-stop bit  no-parity

Byte #   Value (hex/binary)          Description
  1        05 / 00000101       Write register 5
  2        EA / 11101010       8-bit Tx with DTR and RTS set
  3        04 / 00000100       Write register 4
  4        44 / 01000100       x16 clock, 1-stop bit, no parity
  5        03 / 00000011       Write register 3
  6        C1 / 11000001       8-bit Rx

PRINTER port (7): 7-bit 1-stop bit odd-parity

Byte #   Value (hex/binary)          Description
  1        05 / 00000101       Write register 5
  2        AA / 10101010       7-bit Tx with DTR and RTS set
  3        04 / 00000100       Write register 4
  4        45 / 01000101       x16 clock, 1-stop bit, odd parity
  5        03 / 00000011       Write register 3
  6        61 / 01100001       8-bit Rx auto-enables
```

Figure 3 gives a quick example (or two) which may help — namely programming the COMM and PRINTER ports for the default configurations in the Xerox 820. Each group of six bytes is sent in succession to the designated port.

Next Time

As you can see, I ran out of room again, so I'll finish the SIO next time. Looking ahead, I see several hardware/software projects for the 820 including a CHEAP parallel port EPROM programmer (software will be free off the RQP/M) and a hard disk adaptor.

If you have any suggestions for columns, comments, gripes, or some useful information, please write or call the RQP/M at (213) 320-9309.

■ ■ ■

HiSpeed C Library

- **A complete and accurate REFERENCE MANUAL for your desktop. Matching library FUNCTIONS for Software Toolworks C/80 in Microsoft M80/L80 REL format. Disk SOURCE CODE to all functions and utilities. Works under Digital Research CP/M 2.2.**
- **Develop portable applications using Bell Labs Unix V7 conventions. Designed from Kernighan & Ritchie, Harbison & Steele, Banahan & Rutter, Sidebottom & Wortman and Donald Knuth's three-volume set.**
- **Settable dynamic buffers. Full stream I/O. Transcendentals. Strings. Math. Searching & sorting. Timing. Compact code (K&R HELLO. COM is 254 bytes). Up to 255 files. Direct-disk. Directory. Memory management. High-speed seek. Fastest block reads. System access. Speed and code size optimization. Automated compile, assemble and link. Ideal incremental development.**
- **Useful C Utilities: Xref, Beauty, Calls, BoldList, Archive, Portable Source Generator, Jump-on-error Monitor.**
- **Sample Programs: Learn or teach C style using source code from textbooks, scholarly periodicals and popular magazines.**
- **No royalties. No copy protection. No limitations on educational use. No extra charges.**
- **Reference Manual with library object code on disk..... $29.95**
- **Source code archives and utility sources on disk....... $19.95**
- **Phone orders shipped immediately. Write or Call today for free brochure.**

Viking C Systems
2243 South Belaire Drive
Salt Lake City, UT 84109
1-801-466-6820/Grant

VIKING SOFTWARE

OUTSTANDING PROGRAMS
For Your CP/M 2.2 System. . .

AutoSpool/80 — $39.95

Bring the power of a mainframe spooler to your CP/M 2.2 system. AutoSpool/80 features:

- Nearly unlimited storage capacity
- Multiple copies
- Printer controls, including pause for next page, resume print from middle of job, and presettable configuration of pitch and lines per inch.
- Smart file handling, including control of printing order, wild cards in file names, and individual cancelling

CHAIN2 — $15.00

At last! Run *all* your .COM files from *menus,* arranged any way you like, nested to any level! No modifications to .COM files required. CHAIN2 chains to your .COM file, then causes it to chain back to the menu when it exits. Menu program supplied.

DUMPREL — $15.00

Displays the contents of your .REL relocatable files. These files are the output of many assemblers and compilers, used for linking into .COM files.

BANNER — $15.00

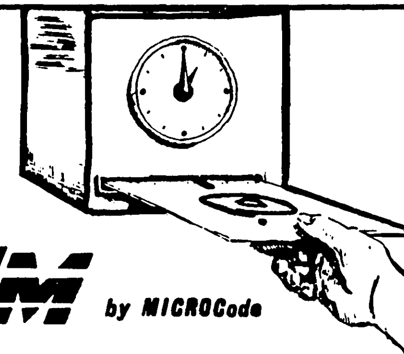
Prints your message sideways in big block letters. Variable size characters. Requires no special printer. Composes characters with a second message.

ALL OF THE ABOVE: $59.95
30-DAY MONEY BACK GUARANTEE

VISA® MasterCard

ANAHEIM SOFTWARE
13814 Hawes St., Dept. M
Whittier, CA 90605
(714) 863-9044

AUTOMATIC TIME/DATE STAMPING OF FILES WITH



QP/M by MICROCode

ONE PACKAGE FOR ALL CP/M SYSTEMS!

- 100% compatible with CP/M 2.2 and resides in same space
- Uses your computer's hardware/software clock
- 10-15% faster disk read/write
- User-selectable default drive/user area
- Automatic disk re-logging
- Full Access to ALL 31 user areas
- Completely replaces BDOS and CCP
- Efficient backup utility
- Requires QBIOS or CP/M 2.x
- 9 new transient commands, 5 new system calls

AUTOMATIC INSTALLATION TO ANY CP/M SYSTEM

QP/M is automatically written to your system tracks, using QINSTALL and your SYSGEN program.

Kaypro/Xerox/BB-I BIOS Available

MICROCode offers a custom BIOS for Kaypro and Xerox/Big Board owners on a bootable QP/M disk.

COMPLETE QP/M PACKAGE with Documentation Includes

D	Sorted DIRectory program. Displays time/date, system files, and .LBR directories.
QBACKUP	Copies and verifies only new or updated files
QINSTALL	Installs QP/M on system tracks.
QPIP	All PIP functions, plus copies files with current or existing time/date.
QSTAT	All STAT functions, plus shows or changes file date(s), archive bit, and MORE.
QSUB	Replaces SUBMIT and XSUB. Has nesting capability, internal command set (including conditionals), embedded XSUB, and more.
TDCNFG	Creates time/date initialization module (supports software clock, Z-Time or Optronics clock).

QBIOS: Available for the Kaypro, Xerox 820-II, Xerox 820-I and BigBoard-I with Emerald Microware or SWP Dual Density (820-I/BB-I). QP/M bootable disk will be shipped if QBIOS is ordered with QP/M. (No installation required!)

*****NEW PRODUCTS FROM MICROCODE*****

KEYMAP: Not just another keyboard configuration program! Written exclusively for the Xerox 820/820-II. Full ON-SCREEN keyboard image while editing. Create your own function keys! Multiple custom keyboards can be created, and any 2 can be instantly toggled.

SMARTROMS: Add 820-II features to your 820-I/BigBoard-I. Provides real-time clock, screen dump, screen/program pause, dumb terminal mode, reset with CTRL/ESC, CRT blanking after 11minutes of idle time, 7 or 8-bit keyboard mode, 820-II (4.0x) compatible vectors, and MUCH MORE! Replaces your socketed ROMs (no soldering!).

PRICES

QP/M..$60.00
QBIOS FOR QP/M...............................$20.00
KEYMAP..$20.00
SMARTROM SET..........(single density)..........$30.00
For SWP.............$40.00............PLUS 2 (Emerald).........$50.00

When ordering, please specify your computer system and media preference (8" IBM 3740 or any popular 5.25" format).

Please add $3.00 shipping/handling (U.S. & Canada) for UPS shipment, or $8.00 for foreign air mail. CA residents please add local sales tax (6% or 6.5%). VISA and MasterCard are accepted (add 3%). **FREE information packet available upon request.**

MICROCode Consulting
Department M5
Box 9001
Torrance, CA 90508-9001
(213) MICROCD (24- hour recorder)

VISA MasterCard

Speech Generation With Radio Shack's SPO256

By David C. Alford

111 Glenbrook Rd.
Rochester NY 14616

This is a simple, inexpensive way to cure any computer's laryngitis.

Can your computer talk? Well, it can if you add Radio Shack's SPO256 Speech Processor. The SPO256 uses parts of words strung together to form complete words. The words aren't great, but still it's speech, and this is an easy hardware/software project.

Hardware

The information provided by Radio Shack with the SPO256 is not totally correct. For example, their description of the LRQ-NOT signal is backwards, so I inverted the signal before using it as a strobe for my Centronics port. You can use any IC containing four inverting OR gates (e.g., 74LS02) for this inverter as well as for the one-shot we discuss shortly.

ALD-NOT is the input strobe that tells the talker that valid data is available from the computer. A1 through A6 (pins 18 - 13) supply the data.

We'll use A7 to fire a one-shot that resets the speech processor after we've finished talking. This insures that it will remain silent after each speech. See Figure 1 (schematic). Radio Shack's schematic shows a rather complex reset circuit but mine is simpler because we're driving the talker with a TTL signal.

The op-amps form an active filter which makes the speech more intelligible. If you don't want to build such a complex filter, you can build the simple filter shown on Radio Shack's data sheet. In either case, the filter connects to the 256's speech output, pin 24.

The LM386 power op-amp doesn't really like to run at 5 volts, so I highly recommend using 12 volts to run it (but no more!).

The recommended 3.12MHz crystal is hard to find, so I use the standard color burst crystal at a frequency of 3.58MHz. I've heard both and it doesn't make much difference.

I added a pullup resistor to data lines D0-D6. (Editor's note: 1K ohms 1/4 watt connected between +5V and each data line should be fine.) The pullups absorb stray signals that might otherwise cause the 256 to speak in tongues.

I mounted my completed circuit board inside my computer box (a modified Xerox) with a small speaker in one of the air holes (there are plenty to spare). Using the 386 power op-amp at 12V, I get a pretty good sound. If you need more volume you could run a connector out the back for audio. The op-amp filter should have no trouble driving an amplifier.

Interrupts

When the Z80 sees an interrupt it takes the contents of its I register (initialized to FF in our system — DON'T change this), and uses this as the high byte for the interrupt table address. It takes the lower byte of the address from the PIO. The result contains a jump to the interrupt service routine for the PIO.

The service routine must end with a RETI (NOT RET) opcode, or you'll leave the other peripherals (like disk drives) shut down.

I changed the definition of WB to fool the RETI instruction into popping 0000 off the stack and into the program counter instead of popping off the actual point the interrupt happened (sending us back into the program we're trying to leave). This took care of both problems. I got the warm boot back to the system, and the system got its RETI code. See Figure 2.

(continued next page)

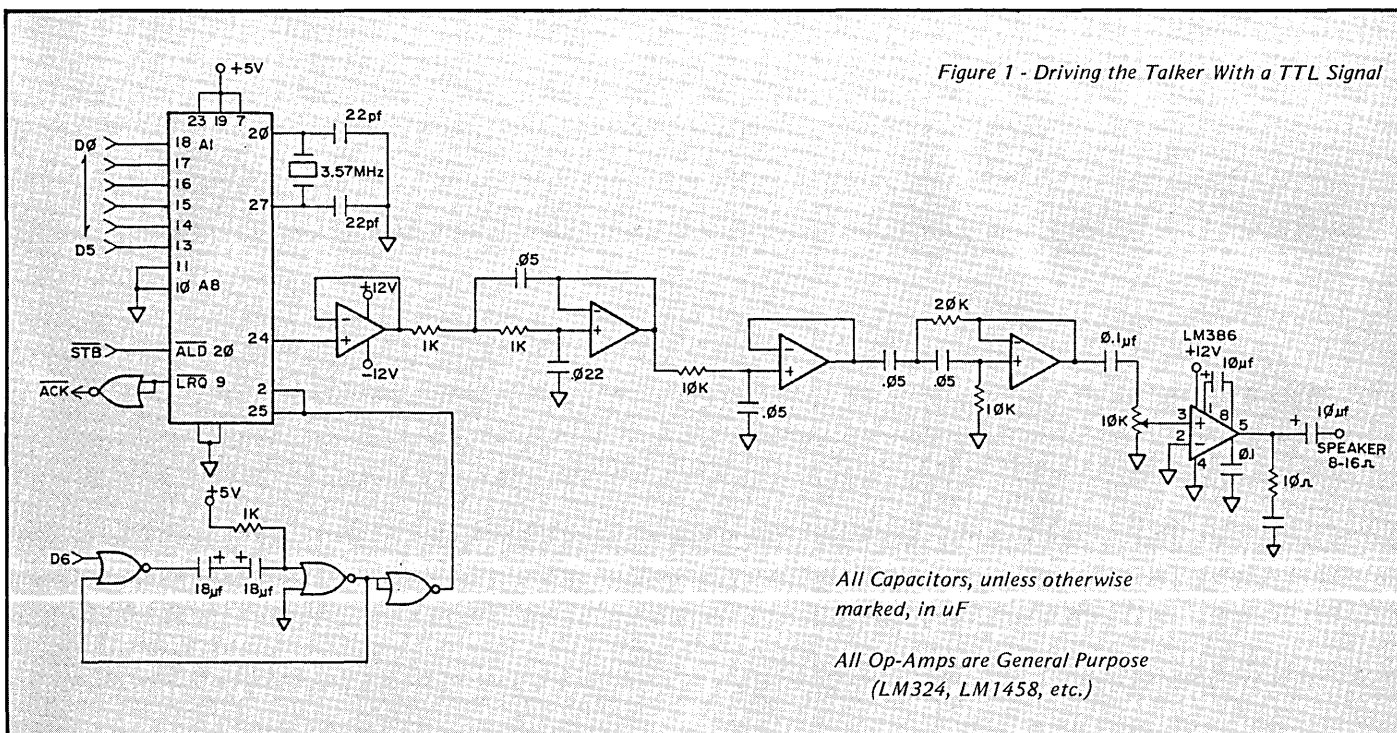


Figure 1 - Driving the Talker With a TTL Signal

(continued from page 33)

The equates at the beginning are set for port B, but note the comments. You need to change these only if you want port A.

GENVEC is the location in the interrupt table for the general purpose port. INVEC is the lower byte address code contained in the port control section. PPIO is the data location in the port. PCTL is the control register.

RETI Macro

After the ORG statement, we initialize the port with the mode 0 control word and then give it a unique vector. Next we prepare the interrupt table and then set the port to send interrupts.

Using Your Assembler

Typing in all those allophones is tedious, but after you're finished you don't have to translate each new speech. You can let the assembler do the work! Just make sure that when you make up a new word you use the allophones as listed.

Radio Shack's data sheet has a small dictionary that doesn't always jive with the allophone listing in the back. Also, MAC won't let you use some

Figure 2 - Service Routine Using RETI Code

```
GENVEC  EQU     0FF1EH  ;0FF1CH FOR PORT A
INVEC   EQU     01EH    ;01CH FOR PORT A
PPIO    EQU     0AH     ;08 FOR PORT A
PCTL    EQU     PPIO + 1        ;PIO CONTRL REG
@       EQU     040H    ;CODE USED TO RESET CHIP
STOP    EQU     0BFH    ;CODE USED TO STOP TALKING
;
;
$-MACRO                 ;TURN OFF EXPANSIONS (WHATEVER THAT DOES)
RETI    MACRO
        DB      0EDH,04DH
        ENDM
;
        ORG     100H
;
;       INIT PIO
;
        MVI A,0FH       ;PIO TO MODE 0
        OUT PCTL
        MVI A,INVEC     ;PIO INTERRUPT VECTOR
        OUT PCTL
;
;       INTERRUPT VECTOR INIT
;
        LXI H,PPOUT     ;HL CONTAINS INTERRUPT ROUTINE ADDRESS
        SHLD GENVEC     ;STORE IN INTERRUPT TABLE
        MVI A,83H       ;OK TO INTERRUPT US NOW PIO
        OUT PCTL
;
;       ALLOPHONE EQU
;
PA1     EQU     00H
PA2     EQU     01H
PA3     EQU     02H
PA4     EQU     03H
PA5     EQU     04H
OY      EQU     05H
AY      EQU     06H
EH      EQU     07H
KK3     EQU     08H
PP      EQU     09H
JH      EQU     0AH
NN1     EQU     0BH
IH      EQU     0CH
TT2     EQU     0DH
RR1     EQU     0EH
AX      EQU     0FH
MM      EQU     10H
TT1     EQU     11H
DH1     EQU     12H
IY      EQU     13H
EY      EQU     14H
DD1     EQU     15H
UW1     EQU     16H
AO      EQU     17H
AA      EQU     18H
YY2     EQU     19H
AE      EQU     1AH
HH1     EQU     1BH
BB1     EQU     1CH
TH      EQU     1DH
UH      EQU     1EH
UW2     EQU     1FH
AW      EQU     20H
DD2     EQU     21H
GG3     EQU     22H
VV      EQU     23H
GG1     EQU     24H
SH      EQU     25H
ZH      EQU     26H
RR2     EQU     27H
FF      EQU     28H
KK2     EQU     29H
KK1     EQU     2AH
ZZ      EQU     2BH
NG      EQU     2CH
LL      EQU     2DH
WW      EQU     2EH
XR      EQU     2FH
WH      EQU     30H
YY1     EQU     31H
CH      EQU     32H
ER1     EQU     33H
ER2     EQU     34H
OW      EQU     35H
DH2     EQU     36H
SS      EQU     37H
NN2     EQU     38H
HH2     EQU     39H
OR1     EQU     3AH
AR      EQU     3BH
YR      EQU     3CH
GG2     EQU     3DH
EL      EQU     3EH
BB2     EQU     3FH
;
;       PROGRAM BODY
;
        LXI H,SAY1      ;HL POINTS TO FIRST 'PHONE'
        CALL PPOUT      ;SAY FIRST 'PHONE' AND FORCE
                        ;FIRST INTERRUPT
WAIT:   EI              ;BOOK SAYS WE SHOULD ENABLE
                        ;INTERRUPTS EVERY TIME
        JMP WAIT
;
;
PPOUT:  MOV A,M         ;GET 'PHONE' IN A
        CPI STOP                ;AFFECT FLAGS
        JZ WB           ;LEAVE IF STOP CHARACTER
        OUT PPIO                ;ELSE OUTPUT CHARACTER
        INX H           ;POINT TO NEXT CHARACTER
        RETI            ;SEE TEXT
;
WB:     LXI H,00                ;HL TO 0000
        XTHL            ;EXCHANGE HL WITH TOP OF STACK
        RETI            ;SEE TEXT
;
SAY1    DB      PA5,PA5,PA5,PA5
        DB      HH1,EH,LL               ;ADD ALLOPHONES HERE
        DB      AX,OW,PA5,PA5,PA5               ;DON'T BE SHY
                                                ;WITH SPACES
        DB      PA5,PA5,PA5,@,STOP
;
        END
```

names like "OR", so I called it "OR1".

Expansions

This program was written to be expanded. The WAIT routine doesn't do much other than loop a lot. You could expand it so your computer does something else while it talks (like print what it's saying). This is what this type of interrupt structure was intended to do.

To add your own speech to the program simply add the necessary DB statements followed by the "phones" (limit five to the line). Don't be afraid to add pauses liberally throughout (particularly at the beginning and end). They help intelligibility.

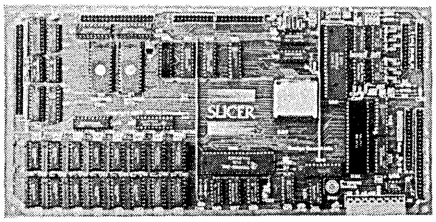
Start Talking

Just assemble your creation with MAC, LOAD it to get the .COM file, and off you go!!

■ ■ ■

SLICER. . . Growing to meet your needs!

The Slicer

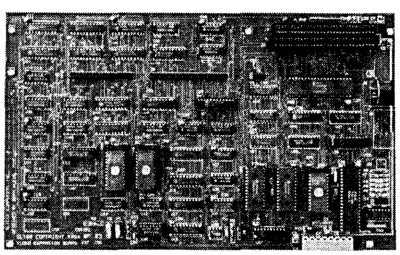


REAL 16-BIT POWER ON A SINGLE BOARD
Featuring the Intel 80186 (C) Step CPU

- Complete 8 MHz 16-bit microprocessor on 6"x12" board
- 256K RAM, plus up to 64K EPROM
- SASI port for hard disk controller
- Two full function RS232C serial ports with individually programmed transmission rates 50 to 38.4K baud
- Software compatibility with the 8086 and 8088
- 8K of EPROM contains drivers for peripherals, commands for hardware checkout and software testing
- Software supports most types and sizes of disk drives
- Source for monitor included on disk
- Bios supports Xebec 1410 and Western Digital WD 1002 SHD controller for hard disks

Fully assembled and tested only .... ~~$995~~ $945

The Slicer PC Expansion Board

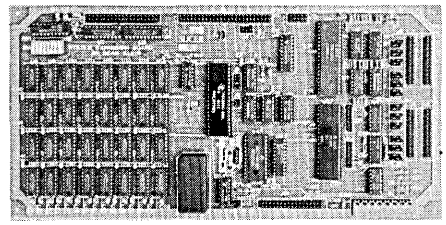


GIVES YOUR SLICER HIGH PERFORMANCE VIDEO CAPABILITY

- IBM compatible monochrome video
- Video memory provides 4 pages of text or special graphics capability
- 2 IBM type card slots for color video, I/O expansion, etc.
- IBM type keyboard port

Fully assembled and tested only $600

The New 1-Megabyte System Expansion Board!



FOR EXPANDED MEMORY, ADDITIONAL PORTS, AND REAL TIME CLOCK

- 1-MB additional dynamic RAM
- 2 RS232C asynchronous ports with baud rates to 38.4K for serial communication
- 2 additional serial ports for asynchronous RS232C or synchronous communication (Zilog 8530 SCC)
- Real Time Clock (with battery backup) for continuous timekeeping
- Centronics type parallel printer port

Fully assembled and tested only $800

SLICER/1 MEG EXPANSION COMBO
The Slicer (without RAM or RAM controller parts) with full 1MB system expansion
Kit form only $1395

CONCURRENT CP/M— with manuals $250
without manuals $200

The SLICER DOS Kit $295
includes Bare Board, CPU and choice of MS-DOS or CP/M-86

The SLICER Enclosure System
alone $125
with 135W Power Supply $245
with 135W Power Supply and 2-80 track 5-1/4" Disk Drives............... $695

PC Compatible Keyboard $150

PC Compatible Amber Monitor $175

10MB (formatted) 5-1/4" Half-Height Hard Disk $700

Western Digital 1002-SHD Hard Disk Controller $200
assembled and tested

8087 Math Co-Processor Board................ $300
bare kit form

8 MHz Intel 80186 (C) Step CPU $100

Check the SLICER Bulletin Board (300/1200 baud) at 612/788-5909 for complete set of manuals and access to Slicer's design team.

SLICER™
SLICER COMPUTERS INC.
2543 Marshall Street N.E.
Minneapolis, MN 55418
(612) 788-9481

All products are available in several kit forms:

Operating Systems are:
CP/M-86 by Digital Research, Inc. $85
MS-DOS by Microsoft Corporation $175
Runs MS-DOS generic software; PC-DOS program operation not guaranteed

MasterCard, Visa, Check, Money Order or C.O.D. Allow four weeks for delivery. Prices subject to change without notice.

By Laine Stump

Development Foundation of Turkey
Tunali Hilmi Cad. 22
Ankara Turkey

If you like a good story told by a very good teller of stories then sit back, kick off your shoes, and read on. Laine is on the kind of adventure that few will ever have, and he's sharing it with us. This is a "don't miss" part of Micro C (and you might even pick up something about Concurrent CP/M and 8088 assembly language interfaces to MSDOS in the process).

Merhaba Efendim! (Hello Everyone!) Welcome to the Micro C Turkish Bureau. I arrived in Turkey on August 24 and spent the next month learning Turkish, wandering around Istanbul (where the Turkish language classes were), traveling to Ankara and back on the train several times, and waiting anxiously for news of my hapless Slicer (he was imprisoned at Turkish customs when I arrived at the Istanbul airport).

I am now at work in Ankara (at The Development Foundation of Turkey, for those of you who just tuned in) and I finally got my beloved out of "the pound" just two days ago. You may scoff, but getting electronic equipment out of customs is a very complicated process, especially when you speak no Turkish.

To get my Slicer back by my side I had to travel to Ankara by train (only $4 each way, but 400 miles and 9 hours!!), sign multiple copies of several forms, and have another man from DFT travel to Istanbul as my representative to give the customs officials all the forms with my passport and residence papers.

Worst of all though, I had to live for an ENTIRE MONTH (that's 30 days, 720 hours, 43,200 minutes...) without touching a computer. It is all over now, though, and it was a good experience to deal with a government bureaucracy as an outsider; now I know what foreigners entering the U.S. feel like.

Power Struggle

My problems were not over when the boxes containing the Slicer arrived at DFT's offices. All of the power supplies were configured for 110 volts, 60 Hz. Turkey, like the rest of Europe (and most of the rest of the world), runs on 220 volts, 50 Hz. I had checked into this before I left and found that my main power supply (an IBM PC-type) had a little jumper for using 110 or 220. "No problem," I thought to myself as I changed the jumper over and plugged in the AC cord. I flicked the switch and was greeted with a sickening metallic "click click" sound from the winchester. Not good (iyi degil).

I immediately turned the switch back off and disconnected the drive, then tried again while monitoring the power supply output voltages. Nothing. 5, 12 and -12 all registered a fluttering 0. My first reaction was anger, then disappointment, and, a bit later, disgust. Finally I started looking at the supply and found two adjustable pots. With a couple of tweaks I had everything going just fine. (Yayy!!, Cok guzel!)

The Wini Still Works

The most wonderful moment of all was when I put the "ol' binder" to the ultimate test — booting from the winchester. I just knew that traveling 12,000 miles by airplane and taxi and baggage handler would ruin it, but it worked first try. I ran FINDBAD (User disk #1) and it did not report a single bad sector out of 611 tracks! Wini has passed the final examination; may it live forever.

Warranty Restrictions

Before you all rush out to buy boxes for taking your computers with you on vacation to the Riviera, I should warn you about some complications. The first has to do with 8" disk drives. Although 5" floppies, winchesters, and new half-height 8" drives are totally DC, most older 8" drives spin the disk with synchronous AC motors. This means that you must purchase a different motor AND a different pulley for the drive. If you catch Cascade Electronics at the proper moment, they can supply you with both (they receive many 220V, 50Hz drives and convert them to 110).

Another thing you should realize is that not all power supplies have the 110-220 jumper like mine did. If you have a switching supply it probably will have the jumper; otherwise it probably won't.

The last major point to beware of is that some video monitors use the line frequency to determine their scan rate. Unless they are very adjustable, these monitors will not work when changing to 50Hz (mine is fortunately all DC).

The major advice I can give is to look before you leave. It is easy to purchase new equipment in the States, but in many countries you would have to order from somewhere else and wait for weeks or even months for a simple transformer, or even an IC or connector. I brought along a box of spare ICs and another box of connectors and ribbon cable. I am already very grateful for my foresight (pronounced: luck).

Computing In Turkey

The state of the art of computing in Turkey is at about the stage it was in the U.S. 8 or 9 years ago. Most people don't know what a computer is. And most of those who do, think that a computer is a very large, expensive, magic machine (mostly expensive).

While I was at language school in Istanbul I saw a proposal from a Turkish computer dealer for an 8 user networked system of Acorn BBC computers (2MHz 6502, 8K of memory upgradeable to 32K) for primary schools. The system, including two 400 Kbyte floppy drives, was about 8 million Turkish Lira ($15,000 US).

It was sold as a network instead of as eight independent users because they wanted to save money by using only two disk drives for the entire system; apparently they don't realize that it would cost only $600 more to have a disk drive for each user. And 8K of memory!!! 64K is now selling for well under $10.

While there are some more advanced systems here (IBM and Wang), I am almost certain that I have the first Slicer in the country (unless someone at one of the NATO bases has one). I feel like a carrier of good tidings and a

(continued next page)

(continued from page 39)

side show freak at the same time. Before my system arrived in Ankara, the people at work were referring to it as "your jalopy" and questioning whether it would work. It is working, I am happy to say, and others will soon be converted.

At any rate, technology is swarming into the country at an alarming rate. The magnitude of the current changes in Turkish computer usage could be compared to that of changes in the U.S. during the '60s and '70s.

Within a few years all the banks and most of the larger companies will be computerized. It may take a while for home computers to become extremely popular because the foreign exchange rate and tariffs, coupled with a low median income, make a small computer a BIG investment for a Turkish family.

Back To Business

Now that I've gotten a few things off my mind, I'll talk about the kinds of things I'm supposed to talk about here: 8086 based computers and software.

First is an announcement of new hardware, not the kind that most of you would run right out and put on your Visa card, but still interesting. Wang computers has just announced the Wang APC, a new multiuser system based on the 80286 processor running Xenix or Unix (also MS-DOS in single user mode). Its price is comparable to the IBM-AT, but its performance is much better (as Wang tells it, anyway).

The processor is running at 8MHz instead of 6MHz, while the memory is operating with 0 wait states instead of 1. Some tests run twice as fast on the Wang APC as they do on the AT.

The big problem, as always, is the price. A four-user APC system logs in at just about $18,000 while a similar system based on a Slicer is under $6000 (and nearly as powerful).

CCP/M

Speaking of multi-using the Slicer, I have been seriously playing with CCP/M for the last week and have had a delightful time. We are currently examining options for putting several four-user accounting systems in DFT's field offices and have been experimenting to see if a Slicer with CCP/M could do the job.

One of the things necessary for a customized job like this is that the hardware and software be extremely configurable. Slicer's hardware flexibility is obvious, but what about the software? First, Slicer CCP/M comes with an EH Productions program called 'SU' that allows you to change almost all of the normal system parameters without even running GENCCPM. Second, if you really want to get esoteric and use some strange custom hardware or have special functions built into the operating system, Slicer has included all the source code to the XIOS (eXtended IO System). Wow!

Even if you don't absolutely need a multiuser system at home, you should seriously consider getting CCP/M just so you can learn about file locking, mutual exclusion, queues, etc., and study a very good implementation of the CCP/M XIOS. This is code written the way it should be written, not just some kludge put together by a terminal-eyed freak who eats only the heads of chocolate bunnies and sleeps in his clothes. (Editor's note: See what I mean.)

MS-DOS Assembly Language

I have been promising this for several months now, but other important things kept coming up. Since I gave an example of CP/M-86 assembly language a few issues ago, I feel it is only fair that I give some time to MS-DOS, especially since it is more sophisticated in some ways (and much more popular). I would still rather use CCP/M where I can exploit multitasking capabilities, but MS-DOS does have a few niceties that bear mentioning.

I will explain some function calls in the text and include a short program at the end to show how to fit all this

```
;***********************************************************************
;*      MORE.ASM - print a file to the console 1 page at a time       *
;*                  similar to UNIX's more.                            *
;*                                                                     *
;*      Assembled with AZTEC macro assembler (inc. with Aztec C)       *
;*                                                                     *
;*                      A>as more                                      *
;*                      A>ln more.o                                    *
;*                                                                     *
;*                              MSDOS version   Laine Stump 9/30/85    *
;***********************************************************************
;
LINES   equ     24      ;lines per page
CR      equ     'M'-64  ;carriage return
LF      equ     'J'-64  ;linefeed
EOF     equ     'Z'-64  ;end of file character
STDIN   equ     0
STDOUT  equ     1
STDERR  equ     2

OPENHANDLE      equ     3Dh     ;open file or device
    READ    equ 0       ;read access only
    WRITE   equ 1       ;write access only
    RandW   equ 2       ;read/write access
READHANDLE      equ     3Fh     ;read from file or device
WRITEHANDLE     equ     40h     ;write to file or device
ENDF            equ     4Ch     ;exit to DOS
;
;*********************************************************
;       a 'DOS' instruction
;
DOS     MACRO   FTNNUM
        MOV     AH,FTNNUM
        INT     21h     ;DOS interrupt
        ENDM
;
;       INPUT and OUTPUT macros to make life easier
;
INPUT   MACRO   HANDLE, ADDRESS, BYTES
        MOV     BX,HANDLE
        MOV     CX,BYTES
        MOV     DX,offset ADDRESS
        DOS     READHANDLE
        ENDM

        OUTPUT  MACRO   HANDLE, ADDRESS, BYTES
                MOV     BX,HANDLE
```

gibberish together into something you can run.

I will warn you before I start that I didn't use MicroSoft MASM to assemble the sample program. I used AS, the assembler included with Aztec C. I don't have MASM yet, and now that I'm nine time zones away I can't just call an 800 number and have it shipped next day air. Anyway, it will probably assemble with MASM, but I can't say for sure.

Getting A Handle On MS-DOS

MS-DOS's file system is much nicer than standard single user CP/M. The first difference you will notice when programming in assembly language is that MS-DOS allows reading and writing any number of bytes to a file or device. This means you no longer have to construct silly little loops to write large buffers to disk, or keep track of "partial records" when you don't have exactly 128 bytes to write. With a single DOS call you can write a single character, or you can write 64K.

You also do not have to worry about parsing the filename into an FCB or making sure you are logged into the correct user number. To open a file with MS-DOS, just put the address of a '0' terminated string containing the filename into the DX register, access code (read/write/read & write) into the AL register, and give the "open handle" function call:

```
FILE    DB  "d:\sub\filename.ext",0
FILEHANDLE DW   ?


MOV     DX, offset FILE
MOV     AL,READ
DOS     OPENHANDLE
MOV     FILEHANDLE,AX
```

DOS is a macro defined in the listing at the end of the article, READ is just the number 0 (read access only), and OPENHANDLE is the function number 3Eh.

When control returns to your program from the openhandle call, a file "handle number" will be in the AX register. The handle number will be used instead of an FCB address when you read or write to the file. To read from the file you have opened, just put the handle number in BX, the address to read to in DX, and the number of bytes to read in CX, then issue the "Read Handle" DOS call:

```
MOV     BX,FILEHANDLE
MOV     DX, offset BUFFER
MOV     CX, BYTES
DOS     READHANDLE
```

On return, if the Carry flag is set, AX contains an error code; otherwise AX contains the number of bytes actually read. Use this information to tell when you have reached the end of the file (bytes read will be less than bytes requested). See the DOS programmer's manual (if you can decipher it) for details on specific error codes.

Writing to a file is similar, except the file must be opened for write or read/write access.

Getting A Handle On Devices

MS-DOS, like UNIX, treats I/O devices just as it treats files. Any device can be opened as a file if you know the device's name, but there are also five pre-opened handles to help make life a little easier:

```
0 - standard input (stdin)
1 - standard output (stdout)
2 - standard error (stderr)
3 - Auxiliary
4 - Printer
```

The beauty of treating devices as files is that you can develop general purpose programs that get their input either from the console or from a file with no change in structure of the code. High level languages usually do I/O this way, but MS-DOS does it even in assembly language. To read a character from the console, use the following:

```
MOV     BX, stdin
MOV     DX, offset INCHAR
MOV     CX, 1
DOS     READHANDLE
```

The character will be returned in memory at the address INCHAR. The only problem with this is that input through function 3F (Read Handle) buffers up an entire line before it

(continued next page)

```
        MOV     CX,BYTES
        MOV     DX,offset ADDRESS
        DOS     WRITEHANDLE
        ENDM
;*********************************************************
        CODESEG SEGMENT PARA PUBLIC 'CODE'
        ASSUME  CS:CODESEG, DS:CODESEG

MAIN    PROC    NEAR
        PUSH    CS          ;fix segment register
        POP     DS          ; for data in code segment
        JMP     MORE
;*********************************************************
;       see text for explanation of why this is here
;
LINECT  DB      LINES-1
DMA     DB      ?,?
MORMSG  DB      CR,'-- More --'
MORLEN  equ     11
;*********************************************************

MORE:   INPUT   STDIN, DMA, 1   ;read 1 char from standard input
        CMP     DMA,EOF         ;see if EOF yet (0 chars read)
        JZ      DONE
        CALL    OUTCH           ;IF not, THEN type this char.
        JMP     MORE            ;and go get another

DONE:   DOS     ENDF            ;finished, return to shell
        MAIN    ENDP
;*********************************************************
;       send 1 character @DMA to console, keeping track of lines
;
OUTCH   PROC    NEAR
        OUTPUT  STDOUT, DMA, 1  ;character to console
        CMP     DMA,LF
        JNZ     OUTEND

        DEC     LINECT                  ;end of line
        JNZ     OUTEND
        OUTPUT  STDERR, MORMSG, MORLEN  ;end of page
        INPUT   STDERR, DMA, 2
        MOV     LINECT, LINES-1
OUTEND: RET
OUTCH   ENDP

        CODESEG ENDS
        END
```

(continued from page 41)

returns a character. If you want to edit the input you'll have to use the normal console input functions (1, 6, 7, and 8).

A Direction In Life

Another nice feature of MS-DOS is I/O redirection. Placing a less than symbol (<) in front of a file name on the command line when calling a program causes the program to get its console input from the named file instead of the screen, totally transparent to the program. If we take advantage of this, we don't have to open files. Just read a character from stdin and write characters to stdout.

Misguided Direction

Redirection can cause problems, though. In my sample program I wanted to print 24 lines of standard input, wait for a key to be typed at the keyboard, and print 24 lines more.

True to the documentation, the keyboard input requested at the end of each page was read not from the keyboard, but from the file being used as stdin. When input is redirected, it is all redirected.

I tried all of the console input commands (there are at least five DOS functions for console input: 1, 6, 7, 8, as well as 3Fh) and all of them were redirected. I began to think that I would have to resort to using a monitor call (naughty practice), but then I remembered the "standard error" device.

Stderr cannot be redirected; its output always goes to the console. "So," I thought to myself, "why shouldn't I be able to input from it too?" I tried and was delighted to see that it worked. Even this had a problem, though. When reading from devices using READHANDLE, DOS buffers the input until you type a carriage return, then it returns the first character.

I was hoping to be able to do different things depending on a single keystroke, but this was not possible. I think that I may be able to solve this problem by setting a different I/O mode or something. I'll get back to you.

Another problem with redirection: When I discovered that the regular console routines (0, 1, 6, 7, 8, 9, etc.) were also redirected, I thought I would save some space and speed things up by using function 6 instead of 3Fh to get characters from stdin. That worked fine until I got to the end of the file. Then the system hung up waiting for a control-Z and I had to hit the reset button to get out.

DOS does not put a control-Z at the end of a redirected file for you unless you are using function 3Fh, so if you are planning on using redirection with a program, don't use the low numbered console functions. Matter of fact, use as few of the functions numbered below 2Ah as possible. Most of the low numbered functions have more flexible counterparts in the new functions. For best results use functions 39h-4Dh as these are the only ones compatible with MicroSoft's XENIX.

Handling Redirection

Using Redirection for this simple program turns it into a useful tool. Besides paging a file to the screen with this:

```
more <file
```

you can do many other useful things. For instance, how often have you needed to create a simple batch file or type a short note and you didn't want to spend all the time calling up a fullscale text editor? MORE can be used as a simple text editor:

```
more >my.bat
```

will let you enter lines of text into a file called "my.bat",

```
more >>my.bat
```

will tack new lines onto the end of the already existing "my.bat",

```
more <thisfile >thatfile
```

will copy the contents of "thisfile" to "thatfile" (with a pause every 24 lines, but it works). AND:

```
more <thisfile >>thatfile
```

appends the contents of "thisfile" to the end of "thatfile"! Also, you can do something like the following:

```
dir | sort | more
```

This command gets a directory of the current drive, sorts it, and sends it to the screen 24 lines at a time. The " symbol means "send stdout of the previous program to stdin of the next program." This is the infamous "piping" that UNIX freaks talk about all the time.

AS Bugs

You'll notice a few strange things about my MORE listing. The strangest is the "PUSH CS POP DS" at the beginning of the program. I had to do that because the linker kept setting the DS register to the wrong value, and I was getting my data from never-never-land.

At that time I was putting the data in a different segment with the SEGMENT directive. Because I could not figure out exactly how the linker was deciding where DS was, I just put the data in the code segment. I hoped the program would then be entered with DS--CS, but I was sorely disappointed, so I just did it myself ("If you want something done right...").

Also, I have the data up in the middle of the code. I did that because MASM (and presumably AS) cannot figure out whether to put in a segment override byte when a variable is referenced unless the variable has already appeared in the source stream.

If you put the data at the end (as I am accustomed to doing) an extra byte will be put in front of EVERY memory reference to make sure there will be room for a segment override instruction if it is necessary! To avoid this you must declare data before it is referenced (similar to Pascal).

The Right Way

ASM86 and RASM (from Digital Research) handle the segment override problem by taking three passes of the source code: 1 to determine what segment each variable is in, 2 to determine code length and offset addresses, and 3 to generate code and listings. It may take a little longer to assemble, but it generates much more compact code (and faster executing if you are running an 8088 or 8086).

Nothing is ever perfect. What I really want is an assembler that generates code and handles segments and link modules the way RASM does and has macros like MASM. It should have built-in support of the 80186, 80286, and 80386, along with the 8087, and be sold in both CP/M and MS-DOS versions. For $49.95.

The Left Way

It's about time I left, so I'll say gulegule for now. Next issue I hope to have a good example of using queues in CCP/M, and maybe more specific information on modifying the XIOS. Sonra gurusuruz!

P.S. Thanks to those who sent info on the MIDI interface (especially the magazine article written in Japanese!). Every little bit gets me closer to understanding. We'll break the industry's monopoly on those specs yet!

■ ■ ■

The Kaypro Column

By Alan Barlow

Box 3634
Seattle WA 98124

We've received a number of calls which start out like this: "Help! My new drives are tearing themselves up. Hear them grinding?"

Actually, they aren't grinding. They're just fast-stepping drives running on a slow-stepping machine. If you want to speed up your data accesses and quiet down those new half-height drives, then you're in the right spot.

Drives too noisy? Want faster disk access? Here are two ways to increase the maximum step rate of quad disk drives, from 6 ms to 3 ms, when used with Micro Cornucopia's Pro-8 ROM, version 2 or 3. Both involve minor hardware modifications, including soldering.

The job requires about the same degree of dexterity as the 2 to 4 upgrade or the 5MHz speed-up (both described in issue #21). In other words, it'll be a cinch for some, but out of the question for others. If you're not sure of your abilities, have a technician do it.

Although the fast step mod is independent of the speed-up mod, you might want to tackle the speed-up first, since increasing the CPU clock from 2.5MHz to 4 or 5MHz results in a more dramatic improvement in overall data processing speed than increasing the drive step rate only.

Step Rate

What is "step rate" anyway? And why change it?

The tracks on a floppy disk are in concentric circles, each a measured distance from the center. A stepper motor positions the read/write head(s) accurately at a given track. This motor doesn't rotate continuously, but at a fraction of a revolution for each step pulse received from the disk controller chip on the main board. One step moves the heads radially from one track to the next over the surface of the disk.

For example — the stepper in my Mitsubishi drives rotates exactly 1.8 degrees (1/200 revolution) per step. Since there are 80 tracks per side, the stepper never completes even half a revolution. The quad drives are configured at 96 tracks per inch (tpi); therefore, each step moves the heads 1/96 of an inch. (The direction of movement, inward or outward, depends on whether the "direction" signal is high or low.)

The original Tandon drives were designed to step from one track to the next in 6 milliseconds. The newer quad drives, including my Mitsis, all seem to be capable of stepping twice as fast, requiring only 3 ms per step. Thus every seek operation (stepping the heads from wherever they happen to be to the desired track) should take much less time at the faster rate.

The time won't be reduced by half because a few milliseconds of settling time is required to allow the heads to stabilize over the new track. This time is fixed, regardless of the step rate.

Speed

I ran CRC.COM on a disk loaded with 780K in 95 files (628K if measured in 1K blocks). It took 223 seconds at 6 ms/step and 198 seconds at 3 ms/step (at 5MHz). Time saved — 11 percent.

^KS (save file to disk and continue editing) on a 52K file in WordStar required 74 seconds at 6 ms/step and 68 seconds at 3 ms/step. Time saved — 8 percent.

I performed this test at 2.5MHz as well. The times were 79 seconds at 6 ms/step (the stock Kaypro II), and 74 seconds at 3 ms/step.

Curiously, the 5MHz speed-up and the 3 ms step rate mod deliver exactly the same improvement in time, about 5 seconds. Both mods together result in a reduction of 11 seconds, a 14 percent improvement over the unmodified Kaypro.

Most of this operation (^KS) is writing data to the disk, which isn't improved by either modification (data transfer to the drives is always 250K bits per second, regardless of CPU clock speed or drive step rate). (See Table 1 for a complete comparison.)

Hardware Or Software

If you've installed one or more quad density (96 tpi) drives in your Kaypro, and read the ad for Micro C's Pro-8 monitor ROM, version 2 or 3, you might think you just need to buy the new Pro-8 and plug it in to get the 3 ms step. The ad does say you can select a slow or fast step rate for each drive. But here, fast means as fast as the hardware can go (6 ms), and slow is slower.

Given the 1MHz clock input, the controller produces a step pulse every six milliseconds. Through software, the controller can be programmed to produce step pulses less frequently, but the fastest step rate is still 6 ms.

Why not simply double the disk controller clock speed from 1MHz to 2MHz?

There is a 2MHz clock signal available on the CPU board. However, the controller uses its clock to determine not only the step rate but also the data transfer rate to and from the drives.

The controller must have a 1MHz

(continued next page)

Table 1 - Benchmarks

CRC	6	3	
5 MHZ	2:55	2:38	(75 files/688K (560K in 1K blocks)
	3:43	3:18	(95 files/780K (628K in 1K blocks)
2.5	5:04	4:47	
^KS (Save file to disk and continue editing in WordStar)			
5 MHz	1:14	1:08	(52K file)
2.5	1:19	1:14	(52K file)
$$ea (save file and continue editing in Vedit)			
5 MHz	0.45	0.44	(56K file)
2.5	0.48	0.47	(56K file)
Assembly (submit file -- load M80, L80)			
5 MHz	0.23	0.20	
2.5	0.25	0.22	

(continued from page 45)

clock when when reading or writing standard double or quad density 5.25" disks. It needs a 2MHz clock for reading and writing eight inch double density disks and some high density (1.2 Mbyte) 5.25" drives (the data transfer rate doubles to 500K bits per second). When it has a 2MHz clock, the controller can generate step pulses at a 3 ms rate. Now if the controller received a 1MHz clock when it was reading and writing data and a 2MHz clock when it was stepping...

I called Micro C and discovered that Dana had already solved the problem by adding a multiplexer to the disk controller circuit.

A multiplexer is an electronic switch that can select one of several inputs and gate that signal to a given terminal. In this case we are selecting either the 1MHz or 2MHz clock signal, depending on the operation in progress, and gating that signal to the clock input of the disk controller chip. Two circuits accomplish this task.

Two Methods, Two Circuits

Through software control, the head load signal can remain off during track seek or restore operations, but a read or write operation always turns it on. The circuit is merely a selector between 1 or 2MHz controlled by the head load output on the 1793 disk controller.

The first method has the advantage of being non-destructive and easily reversible if U87 is socketed.

A multiplexer is piggy backed on U87, and the resulting hybrid chip is plugged back into the U87 socket. Two jumpers run to the 1793 to complete the modification. All modifications can be done from the top of the board, so the main board doesn't even have to be removed.

The second method has the advantage of using spare gates on the board to create the multiplexer, so no extra hardware is needed.

Method #1: Add A Chip

Dana added a 74LS157 multiplexer chip to his CPU board. (This chip actually has four multiplexers on it, but only one is used here.) His circuit is shown in Figure 1.

If you clock yourself down to your favorite five-and-ten-volt store you can pick up a 74LS157 for about forty cents. If U87 is socketed, it's a good idea to pick up a 74LS390 also so you'll have the original chip, just in case you want to reverse the procedure.

On some boards, U87 will have a couple of jumpers on it and a pin removed. If this is the case, prepare the 74LS390 you purchased to match the chip you're removing. Then cut all the pins on the 74LS157 except 8 and 16 at the place where they get wider.

Then solder pins 8 and 16 to pins 8 and 16 on the 74LS390, simultaneously providing power to the 74LS157 and holding it atop the 74LS390. When you reinsert the piggy back chip set, have pin 13 of the 390 bent out slightly so it doesn't go into the socket.

Wiring:

Add a wire from the bent-out pin 13 of the 390 to pin 3 of the 157 (the 1MHz input to the multiplexer).

Add a wire from U82-28 to pin 1 of the 157 (the select control for the multiplexer).

Add a wire from the 390 to pin 2 of the 157 (the 2MHz input to the multiplexer).

Add a wire from U82-24 to pin 4 of the 157 (the output of the multiplexer).

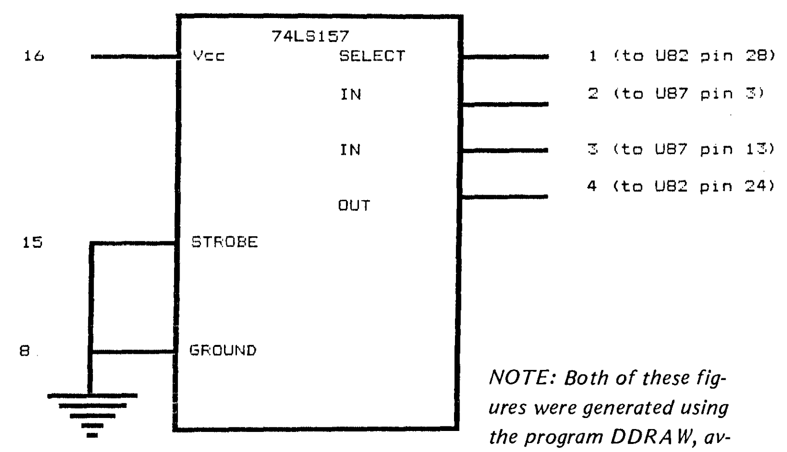
Add a wire from pin 15 of the 157 to pin 8 of the 157 (the multiplexer enable).

This completes the wiring for Method #1. See below for setting the step rates.

Method #2: Use Existing Hardware

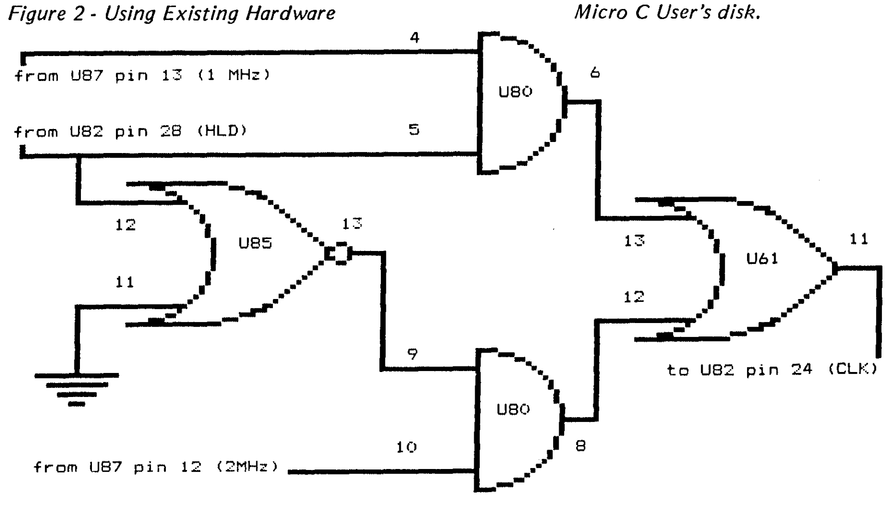
If you don't want to wire in an extra chip, you can easily roll your own

Figure 1 - The Add-a-Chip Method



NOTE: Both of these figures were generated using the program DDRAW, available in the future on a Micro C User's disk.

Figure 2 - Using Existing Hardware



multiplexer, using leftover gates on your Kaypro circuit board. You'll need to cut six traces and then add eight wires. You won't need to bend out any with this method. The trace cutting requires precision and dexterity; read before attempting. See Figure 2.

You might want to correct your Micro C schematic diagram (Kaypro II schematic, 1983, dated 5/23/83).

Look for the U80 gate at coordinates A-7. Pins 1, 2, and 3 are used. Now find the lower of the two U80 gates at coordinates A-3 1/2. Same pins shown, right? (Forget that the first gate is drawn as an OR function and the second as an AND — they are both correct presentations of the same piece of hardware.) The pins of the second gate should be labeled 9, 10, and 8 instead of 1, 2, and 3 respectively. I.e., 9 and 10 are inputs; 8 is the output.

Ready, everybody? It's time for a trace-cutting party! (Hold the beer until after you've finished.) "U87-13" means pin 13 of U87. (Refer to Micro C issue 21, The Kaypro Column, for hints on pin counting.)

After each trace is cut, use an ohmmeter to verify that you've indeed broken the continuity. Traces will be cut on the foil (bottom) side except the first two below:

Cut trace at U80-10. This is the only tricky item. The trace is on the component side, covered by part of the socket. Remove U80 from its socket. With your smallest needle-nosed pliers, break out part of the bridge between the two rows of pins, near the right end of the socket. (The plastic is soft and breaks without difficulty.) Locate two parallel traces near the right end of the socket. Cut the one to pin 10, nearest the center of the socket. Replace U80 in its socket.

Cut trace from U87-13, on the component side. It runs next to the letter "R" of "R34", between U84 and U87.

Cut trace at U61 between pins 12 and 13.

Cut trace at U61 between pins 13 and 14.

Cut trace at U85 between pins 11 and 12. (U85-11 will remain grounded.)

Cut trace at U80-5.

That's all the cutting. Now add these wires (Figure 2):

Add wire from U87-13 to U80-4.
Add wire from U82-28 to U80-5.
Add wire from U82-28 to U85-12.
Add wire from U85-13 to U80-9.
Add wire from U87-12 to U80-10.
Add wire from U80-6 to U61-13.
Add wire from U80-8 to U61-12.
Add wire from U61-11 to U82-24.

NOTE: U85-11 to ground already exists. Leave it.

This completes the wiring for Method #2.

Whichever method you followed, check your work carefully. Use an ohmmeter to check for short circuits between adjacent terminals where you soldered connections. (If you have any doubt at all about your ability, and U87 is socketed, purchase two chips and use the first method. You can always get back to square one by plugging in the original chip.) Note: If you screw up the clock to your floppy controller, it's awfully hard to boot up.

Setting Drive Step Rates

Run PRO82SET.COM (it came on the disk with the Pro-8 ROM, version 2) or CONFIG83.COM (for version 3). All step rate values shown in the menu will now be cut in half. Select the "Use Slower Step Rate on Selected Drives" option. When the next menu appears, if you select item #5, "No Slow Drives; Exit this Function," you are setting all drives to the default value, which is now 3 ms/step.

Now return to the first menu and write the changes to the disk in drive A:. Finally, you must perform a cold boot (RESET) in order to re-read the system tracks and enter the new step rates into RAM. You can use SYSGEN to copy the modified step rate to other boot disks.

I discovered that my one remaining Tandon drive seems to step reliably at 3 ms. (Nevermind that it sounds like a rusty cement mixer gargling pea gravel.) Not all Tandon drives can handle this. If you cannot log onto your Tandons or you start getting errors, use a slower step rate. The second fastest step rate (it says 12 ms, but it's really 6 ms after the modification) will do. There may also be a danger with drives that have a loose band drum. They can knock themselves out of alignment with the faster step rate.

Most quad density disk drives are designed to step at 3 ms. The TEAC and Mitsubishi quad drives which buzz at the 6 ms step rate will become nearly silent at 3 ms. Some Shugart 465s were designed for 3 ms and others for 6 ms. If your Shugart quad is quiet at the 6 ms rate, chances are you have the 6 ms version of the drive.

■ ■ ■

Home Care For The Invalid Computer (Part 1)

By David Thompson, RN (Register Nanny)

You learn a lot of things in this life because you have to, not because you want to. Servicing a recalcitrant system is one of those necessities.

So, I'm setting up this series as a reference work. You don't have to read it now (though if you like misery, you'll love this), and maybe you'll never even have to use it. But, if the technical calls we take are any indication, this series could well be the most requested set of issues we've printed.

It was only a couple of weeks ago. I was in my office trying to make some sense out of the editorial (definitely not a trivial pursuit) when I heard a groan from the other room.

"I've had it with this #&%'#Z@ system!"

I detected a note of distress in Bruce's voice.

His favorite system (modified to the point of anonymity) had been twitchy lately, occasionally refusing to accept characters from the keyboard (especially after idling for an hour or two), sometimes having trouble booting up.

But we were busy and the problems didn't occur very often so we let it slide. This day, however, the system refused to run more than a few minutes at a time and Bruce was getting a bit aggravated. (I'd be more descriptive, but this is a G rated publication.)

Diagnostic Options

The reason we let things slide was that troubleshooting a problem down to the chip level can be a royal pain. You can put in a lot of time and a lot of money and still have nothing to show for it. It's much easier for me to write about diagnostic procedures than to actually diagnose some problems.

So on that optimistic note, let's look at the options in order: from the easiest to the most difficult.

Board Swapping

The easiest diagnostic method is the most common method — board swapping. If the system is made up of three boards, five cables, power supply, keyboard, and monitor, then all you need are a matching (and running) set of three boards, five cables, power supply, keyboard, and monitor, and you can find the problem. You just substitute the good for the bad until the system works.

The reason board swapping is so common is that dealers have found it to be much easier and more profitable than servicing to the component level.

A new motherboard for an IBM XT retails for $720 (though the defective part on the original board might retail for only 25 cents). Dealers can make more by charging $720 for the board plus $50 for service time (30 minutes max) than they can by charging 25 cents for the part and $300 for service time (maybe a day or two of putzing). The customer would also be more likely to complain about the $300 service charge than he would about the $770 total for the new board and installation.

You don't have to know much to swap boards, just how to open the case and how to avoid static-zapping the units. Plus, board swapping practically guarantees success. There is no such guarantee when you get down to the component level. (What caused the component to fail? Will the same problem destroy the new part?)

About the only thing you have to watch for when swapping boards is a bad power supply. Bad power supplies can destroy some or all of the boards in the system. It's best to check the power outputs for correct voltages before trying new boards in a dead system.

Temperature And Stress

"Temperature and stress" (T&S) is the second easiest method of diagnosing problems. Unlike board swapping, it's not a good strategy for all types of problems, and unlike board swapping, the diagnosis and the fix are two separate steps.

However, T&S is an ideal way to start if you have a system that's running at least part of the time and you don't have spare boards.

The T&S method is just like it sounds. You use temperature (heating and cooling) and stress (pressure) to locate heat or position sensitive parts or connections. T&S requires little or no knowledge of the system and very little equipment (a couple of small insulated probes such as a toothpick and a small screwdriver handle, a plastic bag containing a couple of ice cubes, and a small blow dryer). This is the procedure many experienced technicians use first when they receive an intermittent system.

Wiggling Lines

Unlike the previous two methods, "wiggling lines" (WL) requires some knowledge of the system you're servicing. This kind of diagnostic is particularly good at locating dead parts or open signal paths, but you have to know which lines should be high, low, or moving up and down (wiggling). You'll also need to know which pins on an IC are the supply pins, and you'll need to know the proper supply voltages.

For this kind of diagnostics you'll need: a schematic diagram of the system, a voltmeter/ohmmeter (preferably digital), and a logic probe (available from Radio Shack). It would also help to have a good selection of ICs (at least the ones that are socketed) and access to an oscilloscope.

Wiggling Lines is a good way to dig into a dead system, or a system that you can force into oblivion (not stack crashes or other software lock-ups — this is real oblivion). It is not a particularly good method for bringing up a new design or trying a new modification where subtle timing problems can send a system to lunch.

Subtle Timing Detection

This is the most sophisticated type of servicing and is used to detect those problems that absolutely defy other methods.

It requires a thorough understanding of the timing relationships between signals and the ICs they are controlling (often called setup and hold times). You need to be able to read the timing diagrams published in IC books, understand where propagation delay (the time it takes for a signal to get through an IC) is important, and how signals can mysteriously appear where they aren't wanted (through

(continued next page)

(continued from page 49)

capacitive coupling). You also need a thorough understanding of the circuit under test.

I was on the design team for Tektronix's signature analyzer, and we went bats trying to detect significant changes in signal patterns without being thrown off by insignificant shifts.

A signature analyzer is supposed to display a number (signature) based on the precise timing of a signal. If a pin goes high and low at precisely the right times (with respect to some standard signal such as the system clock) then the analyzer should display the correct (the same) number each time you place your probe on the pin.

Take a working system, check each pin on each IC, and write down the number. Then if the system dies, you just check each pin on each IC until you find a signature that has changed. The change points you to the problem.

Unfortunately, if a signature analyzer is too sensitive to slight timing changes, then simple aging of parts (or replacement of parts) will change the signature (though the signal is correct). Conversely, if the analyzer is not sensitive enough to changes, then significant timing changes aren't caught.

Anyway, this whole problem of timing is very serious but very subtle. It takes experience (and usually an EE degree) plus a deep understanding of the system you're diagnosing before this method works.

It even takes a fair amount of knowledge to use the test equipment required for this kind of servicing. You'll need a two-channel, 60MHz (minimum) oscilloscope with X10 probes, plus schematics and IC data books. An in-circuit emulator (ICE) and a logic analyzer are also very, very handy (and sometimes very necessary).

Fortunately, the "Subtle Timing Detection" (STD) is not often required when diagnosing a once-working system. Sometimes heat and age will slow down a part to the point that it refuses to work in a circuit, but that kind of defect will usually show itself if you use the T&S (temperature and stress) method while the system is still relatively functional.

History Of Problem Important

We had been watching Bruce's system slowly get flakier and flakier. Resets had always brought the system back to life and there wasn't a predictable time frame for the lock ups, so we guessed that the problem wasn't heat related.

How To Tell If It's Heat

ICs slow down as they warm up, so timing between parts changes when the system is turned on. Thus, a change in the input level on the input of a simple inverter (turns a high on its input into a low on its output and vice versa) takes longer to show up as a change in the output. Sometimes the slower reaction improves the system timing and you'll see problems when the system is first turned on, but within 30 minutes, it becomes as solid as a rock.

Usually, though, the slow-down of parts creates problems rather than solves them. Often within 30 minutes (sometimes it's as long as three or four hours and is most likely to happen when the room is warm) the system gets strange, and it happens nearly every time you turn on the computer. It's this predictability that makes heat related problems easy to spot.

Sometimes the heating of the parts can cause a socket contact to become intermittent or a cold solder joint to open up, but usually heating problems are IC related.

Is It Mechanical?

We suspected that Bruce's system had a mechanical problem. It's about three years old, all the parts are socketed, and there didn't seem to be a relationship between the failures and the length of time the system had been on (or the room temperature).

In fact, once Bruce found that he could kill the system by lifting slightly on one corner of the processor board we knew we could probably pressure the system into revealing the problem.

I took a tiny screwdriver and used its plastic handle to press on the circuit board. Firm pressure anywhere on the board would send the system into oblivion. Hitting reset always brought it back, so I figured I was dealing with the movement between two parts (probably legs in a socket or a plug-in connector).

Even while I held the pressure that had zapped the system, Bruce was able to bring the system back. I wasn't opening up a circuit (at least not completely) with my pressure (darn it).

Once I knew that firm pressure anywhere on the board would shut it down, I reduced the pressure. I usually pressed on soldered-in parts or on the board itself because I didn't want to fix the problem by reseating a chip and then not know what I had done.

Pretty soon I found that the system was most sensitive right around the processor socket. In fact, just the weight of that tiny screwdriver on one end of the processor chip was enough to send the system out for daisies. The other end of the processor was much less sensitive.

Close inspection revealed that the sensitive end of the processor wasn't completely down in its socket. A very firm push and it popped solidly into place. We haven't had trouble with the system since.

Board Swapping

Just two weeks after we figured out the problem with Bruce's system, another system went down. This time, the system was working perfectly until a pair of pliers accidentally shorted across the contacts on the AC power switch. Sparks flew, the fan in the power supply began to turn, then quit. A puff of white smoke drifted upward from the system. The system had been off, but was connected to the wall (obviously).

I disconnected the system from the power supply and then turned on the supply. No smoke, but the supply's fan didn't budge. I checked the output pins for +5V. No luck.

I opened up the supply, looking for a fuse. There was a fuse all right, but it wasn't blown. A number of systems manufacturers hide fuses inside their power supplies and don't even tell their dealers about them. The service procedure for any dead power supply is replacement. When the manufacturer gets the supply back he can usually just replace the fuse and ship it out as a replacement unit.

I probed around inside the supply to see where the problem was. It turned out that the oscillator and the big drivers were no longer turning the 200 Vdc into nice, high frequency square waves for the little torroid transformer. (If this sounds like Greek, it's okay; I don't recommend that anyone try to fix a switching power supply if the fix isn't a simple fuse replacement.) Unplug the supply before getting into it and don't plug it back in until it's buttoned up. Period. 200 Vdc isn't fun.

Okay, I Need A Power Supply

I dug up another supply and hooked it up to the main board (this is an XT clone). Before turning on the power I removed all the other boards, I/O, video, floppy driver, winchester driver. Then I turned on the power, waiting to hear the little "beep beep" that says the board is happy. No "beep beep."

I turned off the supply. As I turned it on again I watched to be sure the fan started to turn the instant the switch clicked. If it hadn't, I would shut it off immediately, suspecting a short on the main board. I also checked the voltmeter I had connected between the 5V supply and ground. It was 5V.

Visible Damage

I looked closely at the main board. No visible damage, but on the multi I/O board, one of the heavy supply line traces had burned through. The trace had obviously acted as a fuse. I looked closely at the ICs.

The ICs at one end of the board weren't as shiny at their centers as at their edges. Obviously a power supply spike had cooked these parts. One, in fact, had a tiny pin hole in its top, probably the source of that white (death) wisp of smoke.

Hidden Damage

It's not unusual for the parts closest to the supply to take the brunt of a voltage surge. They'll short out and reduce the bus voltage before parts farther into the board are damaged. Unfortunately, even parts that work fine after such an episode may fail a short time later. I've heard tales of people connecting 110Vac to the 5V line on a system, and then after replacing a chip or two have the system running again.

"Of course it isn't very dependable," an owner of one such system mentioned to me. "It runs okay, but it takes 8 or 10 resets to get it started in the morning and I have to reset it every hour or so as it locks up."

Anyway, we wound up replacing the processor board and the I/O board. That was all. I did a little "wiggling lines" checking around the dead processor and found that it was receiving clock and power, but it wasn't doing anything else. (We'll discuss "wiggling lines" procedures in a future issue.)

Gary even pulled the RAM off the defective processor board and stuck it in his system. It worked fine. At least so far.

The rest of the boards (winchester controller and video graphics) look unscathed and work fine. The floppy drives are happy, also.

Next Time

This series will take at least one more installment. We've got to look a little more closely at using temperature as a tool, and we'll at least start on the wiggling lines procedures.

"Wiggling lines" will require some theory of operation discussions, so if you don't have a schematic and theory of operation of your system try to get them. The material we'll cover will be Greek otherwise.

■ ■ ■

Build A PC For Less Than $800

By Gary Entsminger

Here's how, and I promise it won't strain your brain, wilt your wallet, or break your fingernails.

Let's get down to basics.

What's A PC, Anyway?

Since August 12, 1981, "PC" hasn't stood for "personal computer"; it's stood for "IBM personal computer." And lately, PC's definition has expanded again — to "IBM compatible personal computer." I think that's too many words for two letters, so let's say that "PC" stands for "personal clone." It's a reasonable image, and it won't date us.

A basic PC (not to be confused with a PC running BASIC) consists of —

Cabinet
Power Supply
Mainboard
2 Floppy Disk Drives & Controller
Serial & Parallel Port
Keyboard
Video Board
Monitor
Miscellaneous Cables and Control Cards

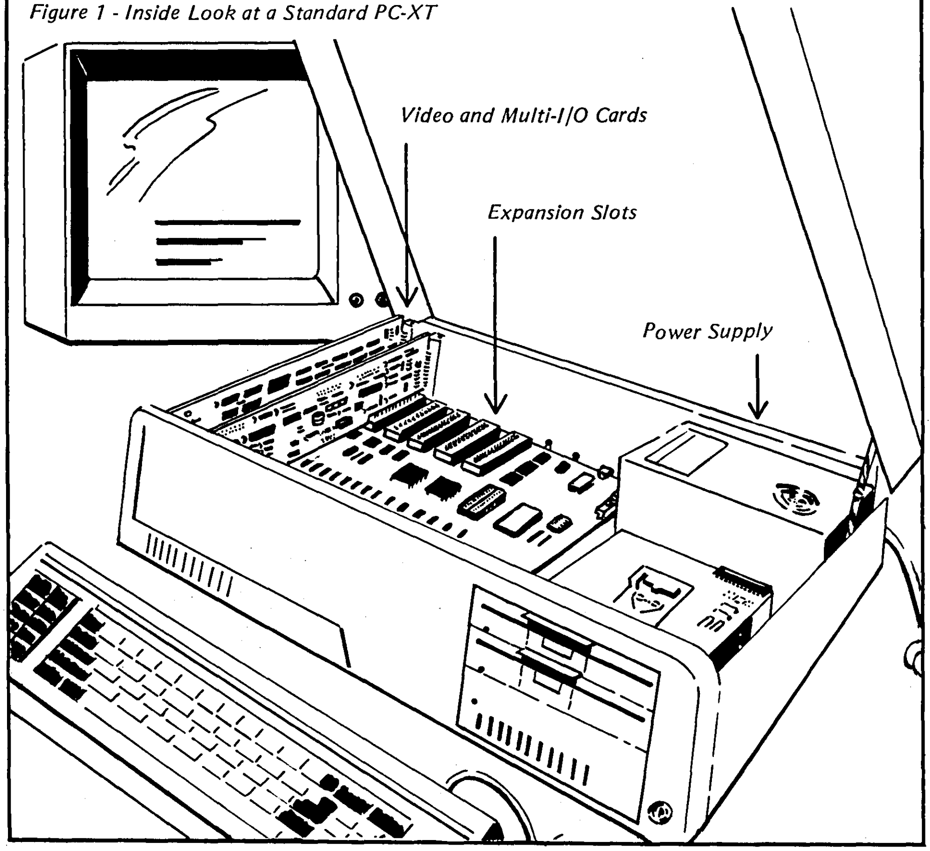
The keyboard is accessed from the main board, but we'll need controllers for the floppies, printer, clock, modem, joystick, and monitor.

To simplify matters, one multi I/O card can handle 2 floppies, 1 parallel printer port, an RS-232, a real-time clock, and a game port. A video card will control the monitor, allowing us to run RGB or composite color, or composite B&W. In short, we take a main board, add two plug-in boards, power and peripherals and we're on our way.

To see how these parts fit into a PC, study Figure 1, a standard PC-XT case opened to reveal its insides. Notice the roominess — 8 expansion slots, 2 taken by the multi I/O card and the graphics card. One more is used by the (short) winchester controller.

The beauty of the PC is in its willingness to be expanded. If you want to add something, just plug it in. If you want to speed it up, just — but I'm getting ahead of myself. First, let's get a basic PC together.

Figure 1 - Inside Look at a Standard PC-XT



How Much Is It Going To Cost?

I went shopping, letting my fingers do the walking, so to speak, and here's the bottom line this week.

Mainboard — a MEGA/XT 4-layer with 4.77MHz 8088 CPU, 8 slots, and 256K RAM $183 ($165 + $18 for RAM). There is a 2-layer board for $120, but they recommend the 4-layer and so do I.

135 watt power supply and case — $125. (The IBM PC comes with a 63 watt power supply, which just isn't enough to handle a hard disk and expansion. So if you think you might add a hard disk, buy the larger power supply. I'll include the larger supply in the personal clone, assuming I'll be adding things.)

IBM style keyboard — $55 (not great, too spongy for me, but it works).

Two Mitsubishi 5.25" DSDD 360K drives — $100 each. You can cut costs here by buying TEACs or Shugarts (some as low as $85 each), but the Mitsubishis are so nice, I'd pay the extra. (By now you should be getting the idea that this cheap PC isn't cutting all the corners. At these prices, let's be extravagant.)

Amber monochrome monitor — $80.
Multi I/O card — $125.
Video (graphics) card — $65.

Grand total — $803 (including cables). A less extravagant system, if you really cut corners (cheaper drives, 65 watt power supply, 2-layer board) goes as low as $677, but the $803 system is XT compatible, expandable, rarin' to go, and so far, dependable.

If you can't wait to find where you can purchase the parts to build your own PC, see Figure 2. But come back after you've ordered, and I'll talk about faster boards, more memory, and putting the system together.

Careful — System Under Construction

The PC's forte isn't speed (in the beginning, anyway), it's the ease with which everything fits together. And the roominess lets all those warm circuits breathe easily. So let's lay parts on the table and get started.

First, stuff RAM into the main board. It might have arrived with the RAM already in place.

If so, congratulations; you can skip this part unless you're planning on adding more memory. If you need to stuff, look over Figure 3 first. The RAM sockets on the MEGA/XT board are in the opposite corner from the 8088 and power supply connector. There are 64 RAM sockets, so you can't miss them.

The best way to stuff 'em is to set a RAM chip over the socket with one set of legs slightly started down into the socket. Then (using your thumbs) slightly press the set of legs nearest you toward the other until it, too, just starts down into the socket. Then, using your thumb on top, firmly press straight down on the chip. It should slide in easily and snugly. The only "gotcha" is BENT PIN, but you can avoid it by making sure each pin is started into the socket before you press in. Look carefully at each leg before you press down.

Figure 2 - Where to Buy Parts

```
Mega/XT main boards 4-layer — $165.
                    2-layer — $120.
Multi I/O card              — $ 95.
Case/135 Watt Power Supply  — $125.
Video card (color/graphics) — $ 65.
Mitsubishi Drives           — $100.
Samsung 12'' Amber Monitor  — $ 80.
Seagate 20 Meg Hd disk/contr— $550.

Sky High Co
1461 University Ave.
Berkeley CA 94702
415-549-3472

Turbo Board (7.5MHz 8088)   — $365.
Samsung 12'' Monitors       — $ 75.

Lolir Lectronics
13933 N. Central Expressway
Suite 212
Dallas TX 75243
214-234-8032

AT Mainboard              — $1250.

C.J. Computers
2424 W. Ball Rd. Suite B
Anaheim CA 92804
714-821-8922

Turbo PC (6.67MHz 8088)   —  $795.
(includes mainboard, case,
640K RAM, 1 drive, keyboard,
power supply)

PC'S Limited
7801 N. Lamar, #E-200
Austin TX 78752
800-426-5150
```

Mounting The Boards

After you've carefully installed RAM (it'll take a few minutes), screw down the mainboard. It goes in the left half of the box on the nine pegs. You'll need to buy nine screws (Phillips head) if your board didn't come with them.

The next steps can be taken in any order, and each has an advantage. If you put the power supply in first, you can check out the mainboard before you get too far along (worried?). If you put the drives in before the power supply, it's easier to connect the drive cables (but it's not really bad either way).

If you're going to install a hard disk you can do that before the power supply goes in. Either way you go, however, it's easy.

Power

Let's go power supply first and fumble with the drives later. The supply sits in the right rear of the box, cables toward the mainboard, fan up. It screws in from the back (the screw holes are already there for you), and you can reach the holes easily if you lower the lid. It's a snap!

If you want to test your mainboard, now's the time. Plug the cable with 11 colored wires coming from the power supply into the mainboard at the rear right, near the 8088 (see Figure 4). Then plug the small, 2-wire cable from the speaker into the front right of the board, (the speaker cable fits over the 4-pin socket labeled "speaker"). Hook up the speaker so the board can talk to you.

Reset & Speaker

While you're at it, connect the reset cable (my reset button came on the front of the case) to the mainboard. It fits in the R8 socket (near the speaker socket on my system and looks just like it). If your reset button is somewhere else, you'll need to figure out where R8 is. Once you find it, the cable just snaps in.

To check out the mainboard, plug in your power supply and fire up. If the beeper beeps, it's working. Congratulations. You've got a mainboard. If it doesn't beep, check your connections and make sure all the chips are firmly in their sockets. If everything seems secure and still no beep, something's not working. It's probably best to call the vendor.

If you can, try to buy as many parts as possible from one distributor; it'll make solving a malfunction problem a lot easier. But don't worry — chances are your board will come up just fine. If you don't have a speaker, then just proceed along; we'll test the whole system shortly.

(continued next page)

Figure 4 - Block Diagram of Power Supply

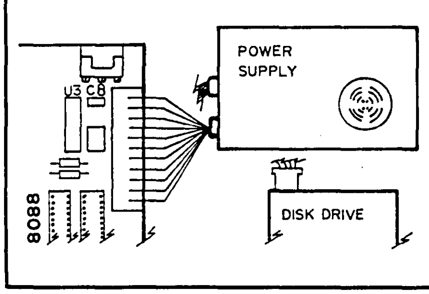
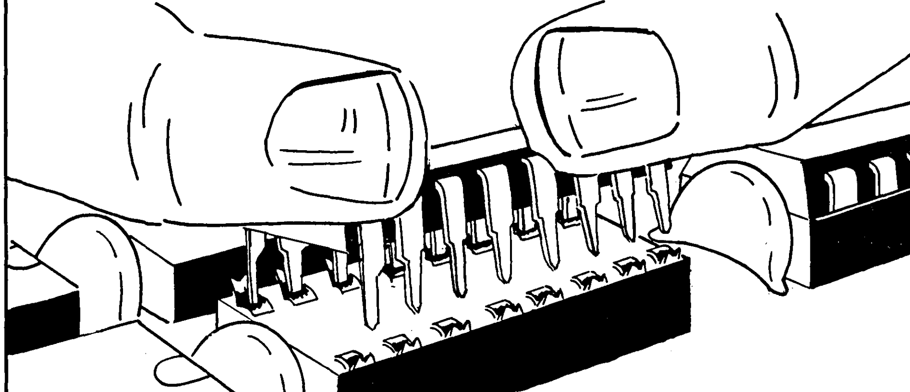


Figure 3 - Stuffing the RAM into the Main Board



(continued from page 53)

Drive Installation

Next, install the drives. They slide into the cutaways up front and screw into the metal plates inside the cabinet. Screwing is a little awkward — you come in from the sides — so work up from the bottom.

If you're adding a hard disk now, put it in first, and connect cables as you go.

Connecting the cables correctly is a little tricky. You'll need to connect two at the rear of each drive — one from the power supply and one from the multi I/O card (or whatever your disk controller card is).

The power cables connect only one way. The drive data cable has two connectors for two drives. In the IBM world, you jumper both drives as drive 1. The data cable determines which one is A and which is B. Also, the red edge of the ribbon cable is pin 1. Just look for pin 1 (or pin 2) marking on the data connector at the back of the drive.

Now you're just two cards and a few more connections from a clone.

I/O

Next, plug in the I/O card. It fits into any of the slots on your mainboard. Take your pick of any except slot 1. Slot 1 (the closest to the 8088) is reserved for a ROM board (usually BASIC), so we don't use it. Then remove one of the metal card plugs from the back of the cabinet, and slide the card down into the slot, making certain the pins on the card and the slot line up. Next, screw the card down (one screw at the rear of the cabinet).

Then connect the other end of the cable you just connected to the floppies to the 34 pins marked "disk" on the I/O card, using pin 1 again for reference. It's generally marked on the card (on the board we used, pin 1 was toward the front of the computer).

The printer port is mounted at the rear of the card, so you won't need cables. But the serial port (J22 on the MEGA/XT) needs a cable and a connector. There's a punchout for a db25 at the rear of the cabinet — you'll need two screws to attach it.

Next, plug in the video card. It plugs into any empty slot (except #1); its sockets are mounted on the back panel.

Now, close the box and plug the keyboard into its connector at the back of the mother board. Also plug the monitor into the video board.

Now, ladies and gentlemen, start your engines.

Faster PCs, XTs, X16s, And ATs

Once you've got a PC and you want to jazz it up, there are several routes you can take, depending on your emphasis — more speed, number crunching, more storage, more dynamic memory, or co-processing. What did you want a PC for, anyway?

The cheapest way to add a little more speed is to swap your 8088 CPU for an NEC V-Series 8088 compatible processor. It's about a $20 investment and will speed things up a little (see Randy Davis' article this issue).

The next cheapest route, from 4.77MHz to 7.5MHz, is to buy a Turbo XT board instead of an XT board. That'll up our no-corner-cutting PC's price from $803 to $1003. The Turbo board will act just like the XT board, only a little faster. I don't have any benchmarks to show you, but I'll guess about 50 percent faster. Noticeable.

A more elegant speed-up would be an X16, 80186 board. Buy it instead of the PC board. It's spendier, but it's at least 3 times faster than the PC (one benchmarking organization says 4.2 times). That switch would up the system cost to $1470.

The best price I could find for an AT mainboard was $1250, which set that system at about $1900. See Figure 5 for a table comparing systems.

If number crunching is your bag, an 8087 on any of these systems will quicken your calculations significantly. The cheapest route again starts at the PC and raises your investment approximately $150. Still, you're under a thousand dollars for very fast crunching.

If you want faster screen output and generally faster computing in addition to number crunching, the X16 will run very fast (with the 8087) for $1620.

Since the AT running an 80287 coprocessor is slower (or at least no faster than the 8087), you'd probably forego the AT. Use what you save for a printer, a hard disk, or a co-processor board.

If you need more memory you can add up to a megabyte by changing the 64K RAM to a 256K RAM (costs about $2.70 per chip, or under $90 for a Meg) and adding a jumper at E2. See Figure 6. If you need storage, skip the second floppy (save a hundred) and buy a 20 Meg hard disk (with controller, it'll cost you $550).

Figure 6 - Adding a Jumper at E2

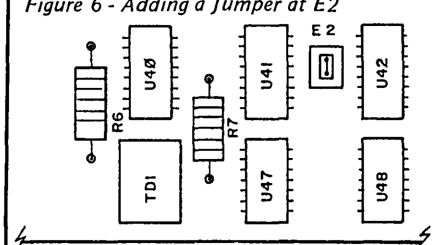


Figure 5 - Cost and Memory Comparisons of Different Systems

SYSTEM	COST (with 2 floppies)	MEMORY	COST (1 floppy, & 20 Meg Harddisk)	COST (1 floppy, 20 Meg Harddisk, & DSI-32)
PC (4.77MHz 8088)	803	256K	1253	2403
TURBO(7.5MHz 8088)	1003	256K	1453	2603
TURBO(6.67MHz 8088)	1115	640K	1565	2715
X16(8MHz 80186)	1470	512K	1920	3070
LEADING EDGE(8MHz 8088)	1599	256K	2049	3199
COMPAQ (8MHz 8086)	1875	256K	2369*	3514
AT(6MHz 80286)	1900	256K	2350	3500
286i(6MHz 80286)	...	640K	2795	3945
COMPAQ AT(8MHz 80286)	4499	256K**		
		640K***	6254	7409

*(factory equiped hard disk)
**(list with 1 1.2 Meg floppy and color monitor)
***(list with 30 Meg Hard disk and tape backup)

If you need multi-tasking on top of everything else, add the Definicon DSI-32032 board to any of the above for another $995 with 256K RAM ($1150 for a megabyte if you stuff your own RAM) and Concurrent DOS. The X16/DSI-32/multi-tasking work station equipped with a 20 Meg hard disk would run $3070 sans operating system and would be very powerful.

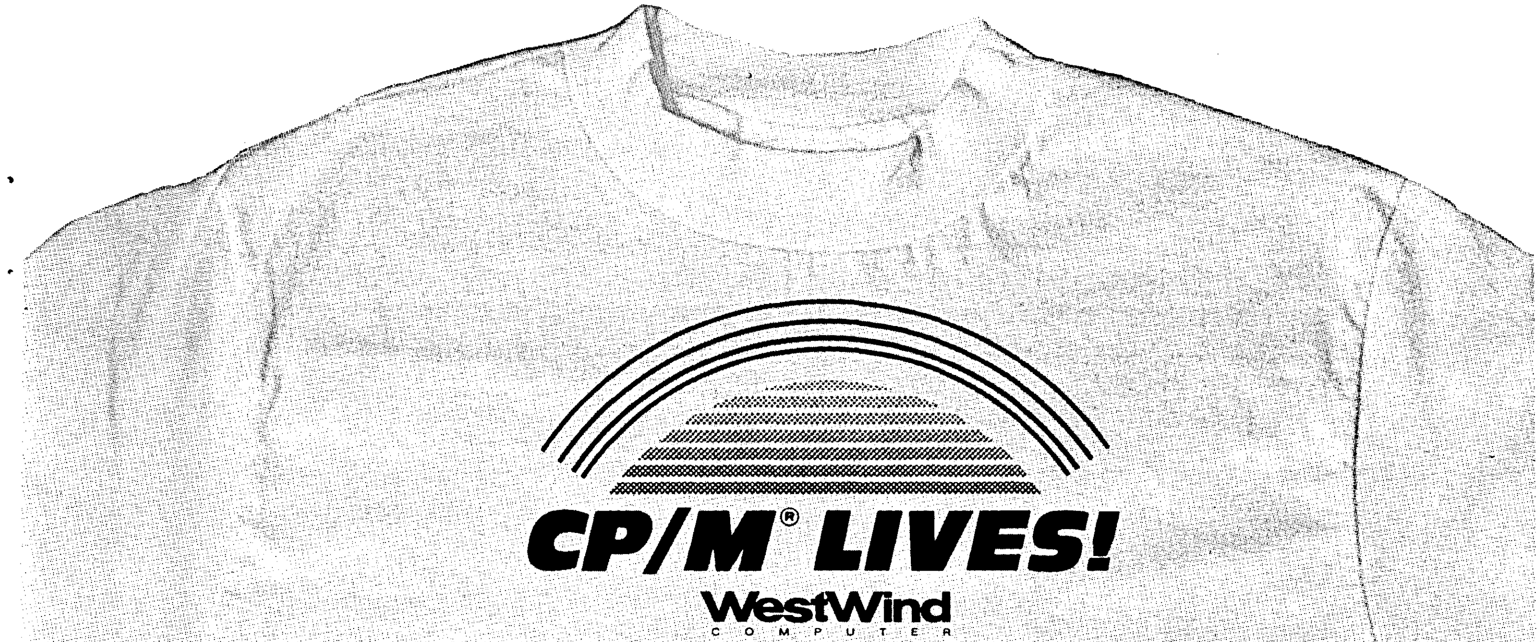
Movin' On

Obviously, the strength of the Intel processors/IBM PC standard is upward compatibility and flexibility. You can have just about what you want, and at a reasonable price, if you're willing to shop around, stuff a little RAM, turn a few screws, slide in your drives, and plug in a few cables.

Our basic PC isn't all that fast (I still like my Kaypro), but it's supported by neat and useful graphics, a wide variety of software, and most importantly, it's expandable.

If you're waiting for the perfect sound and graphics computer and you want the eloquence of a 68000 CPU, then by all means wait for the Amiga or one of its compatibles. But if you're looking for the quick fix now, build yourself a personal clone. It's compatible. (And who knows, with co-processor boards popping up all over, you could put together a system that would make a Cyber flinch.)

■ ■ ■



CP/M® LIVES!
WestWind
COMPUTER

Intel's Speed Trip — Or Why The New Processors?

By Dean Klein

PC Tech
904 N. 6th St.
Lake City MN 55041

Only Apple owners seem to be unaware that processor speed affects how quickly things get done. (Otherwise why would they own Apples?) Dean definitely knows how processors can hold us back or make us go. Herein he discusses the going.

The semiconductor manufacturers are making a lot of claims, each touting the speed of their microprocessors. They even show benchmarks supporting their claims. But even among similar chips, such as the 8086/8088, manufacturers are often blowing smoke. What does, in fact, make processors faster? Let's examine one family (the 8086) and see.

Inside The 8086

Intel introduced its first 16-bit CPU, the 8086, in 1978. In some ways the 8086 is really an expanded 8-bit microprocessor (Motorola fans would definitely agree!), incorporating several innovative concepts —

1. Separation of instruction fetch and instruction execution, thus dividing the chip into two parts: the Bus Interface Unit (BIU) and the Execution Unit (EU)
2. Queues
3. Wider Internal Data Bus
4. Increased Clock Speed

BIU And EU

As you might expect, the BIU is responsible for operations on the bus, including fetching instructions, reading or writing to memory, and reading or writing to I/O.

The EU decodes and executes instructions, and some instructions take longer to execute than others. For instance:

```
AND     AX,BX    ; Logical operation
                   on two registers

ROL     AX,CL    ; Rotate AX left
                   CL times
```

Each of these instructions is coded in two bytes, but the execution times can be much greater for the second. In the 8086 the first instruction requires three clock cycles to execute while the second instruction could take 2056 clock cycles. We'll return to these instructions later when we examine some newer processors.

Queues

The designers at Intel (and Motorola, National Semiconductor, Zilog, and God knows who else) figured all that silicon shouldn't sit idling while the EU was working, so they added an instruction queue to keep the BIU busy.

The 8086 BIU will fetch up to 6 bytes of instructions to fill its queue and will fetch another instruction whenever there are two empty bytes in the queue. This allows Intel to use slower memories without significantly degrading processor performance.

For Instance

A normal memory access cycle takes four clock cycles with no wait states. If a typical instruction takes five cycles, adding a wait state (access takes 5 cycles) won't slow the processor unless it gets a string of short instructions. And if the current instruction asks for a memory read or write, the processor doesn't have to wait to fetch another instruction. So it's faster, but not without drawbacks.

So What's Wrong Now?

Imagine you're trying to debug a new system by watching the instructions as they come down the bus. Since the processor may branch on an instruction (and dump the rest of the instructions already in the queue) some of the instructions in the queue won't be executed. A waste!

Also, you have to be careful about modifying instructions with Debug. After all, if the next instruction takes fewer than six bytes, it will already be in the queue and won't be modified. (Of course, this little drawback encourages good programming techniques; maybe CPU manufacturers should design a 1 megabyte queue!)

Data Bus Width

You might assume that a 16-bit microprocessor would require a 16-bit-wide data bus. It ought to, but it doesn't.

Following the introduction of the 8086, Intel released the 8088, an 8-bit version of the 8086. The 8088 has the same EU as the 8086 and a different BIU and is, of course, the microprocessor used in the PC.

Bus width affects microprocessor performance most in applications that move or evaluate large amounts of data, such as a database program. Since the data bus width of the 8088 BIU is only 8 bits, it takes twice as long as the 8086 to move 16 bits. If you look at STRING MOVE, STRING SCAN, and other instructions, you can see the difference.

Of course, if the processor EU is executing a series of short instructions, the BIU may not be able to keep the queue filled, which also hurts performance.

Now, you might shout, "Increase the queue size!" But it won't work. Remember the EU often uses the bus for operations such as reading and writing memory. It turns out that the queue must be shortened to keep the BIU from monopolizing the bus with instruction fetches. So the queue length is only 4 bytes in the 8088. (Note: this means you could write self-modifying code that would run on an 8088 but not on an 8086. Don't do it!).

Address Generation

Today's processors offer a variety of addressing modes, some quite involved. Base indexed addressing, for example, adds the contents of a base register to a pointer register and an offset in order to compute the data address. This address, in turn, must be added to a segment register to locate the final physical address. Whew! Of course, this takes a bit of time.

The times for many 8086 (and 8088) instructions have "+EA" appended. The total is the time an instruction takes for "effective address" calculation. In the 8086 and 8088 these address calculations are performed by the same hardware that does arithmetic instructions.

The newer processors — 80186, 80188, 80286, and the new NEC parts — have dedicated addressing hardware so there is no penalty for effective address calculations.

Execution Unit Speedup

One obvious way to improve processor performance is to reduce the number of clock cycles required for each instruction. Usually, a designer reduces this number by adding hardware to the chip, thus increasing processor complexity.

The 80186, for example, uses this approach to greatly improve performance on some instructions. For example —

```
ROL     AX.CL    ; Rotate AX
                   left CL times

DIV     BX       ; Divide AX by BX
```

In the 8086 and 8088, the "ROL..." took eight clock cycles plus four additional clock cycles per bit rotated. The 80186, 80188, V23, and V30 require only five clock cycles plus one additional cycle per bit rotated.

In addition, these newer processors limit the maximum value of the CL register to 31. This eliminates the possibility of tying up the processor with a bunch of meaningless shifts.

The newer processors do even better with the "DIV..." instruction. In the 8086 "DIV" takes from 144 to 162 clock cycles. In the 80186 it takes 38, thanks to an improved shifter. Another device, a barrel shifter, can shift multiple bits in a single clock cycle; it's included in Intel's new 80386.

Since other instructions such as multiplies and divides use a lot of shifts, the increase in shift speed improves the overall speed of the processor.

Clock Speed

Clock speed is also significant in processor performance and some of the new ICs are running significantly faster than their earlier cousins. Of course there are also limits imposed by memory and I/O devices.

In Sum

All the features I've mentioned can affect speed, and many manufacturers are working diligently to improve their devices.

It's easy to design machines that run faster, but productivity improvements don't always keep up with increased processor speed. I/O device speed and software quality are major bottlenecks. Now if we could just get faster software writers...

■ ■ ■

Pascal Procedures

By John P. Jones

6245 Columbia Ave.
St. Louis MO 63139
(314) 645-1596

Even the cheapest dot matrix printer is capable of creating very high quality characters. If you understand the graphics modes for these printers and a little Pascal you can create your own character sets and type faces. In this issue John covers the theory and practice of making a great impression.

Many of the newer dot matrix printers provide a Near Letter Quality (NLQ) print mode in addition to the normal draft quality. Since I have an occasional need for higher quality printing, but can't justify a new printer, I decided to emulate NLQ printing with a Pascal program. The idea is certainly not an original one — there are several fancy print programs on the market. Written for a Star Micronics Delta 10 printer, the program can be easily modified for any printer which has bit mapped graphics capabilities.

Printer Problems

The problem with draft mode printing on inexpensive printers is that they print a minimum number of dots per character. The standard arrangement for a 9 wire print head is a character 9 dots high by 12 half dots wide. In practice, however, characters are usually only 7 dots high (either the top 7 wires in the printhead or the bottom 7). Also, horizontal dots are spaced quite far apart.

Bit graphics are handled quite differently. Every dot must be specified. The standard method for this involves a command to enter the graphics mode followed by the number of dot columns to be printed in the current pass, and finally the string of dot data bytes. Normally 8 dots can be defined for each dot column. It will depend on the model of the printer whether the top dot of the print head is the high or low order bit of the data byte.

The Delta 10 has three horizontal densities for bit graphics. I chose 120 DPI because this is the same dot density as standard 10 pitch print — 12 half-dot widths per character. In graphics mode adjacent dots can be printed at half-dot spacing. The printer can also do fractional line feeds, as small as 1/144 inch (approximately half a dot).

On the Delta, two print passes is sufficient. A half-dot line feed between the passes interleaves the dots vertically. The net result is a character formed from a matrix 12 half-dots wide by 16 half-dots high with no restriction on adjacent dots. This roughly quadruples the total number of dots that can be printed per character.

Designing The Characters

Here's the real work in this project. Designing the characters is tedious, and it's easy to make mistakes converting dot patterns to numeric values. Plan on using lots of graph paper. I designed my characters manually (actually I borrowed most of them from another printer) and used a quick and dirty program to enter the data into a file. A good project would be to write a screen oriented input program that would show the dot patterns as they were entered.

Listing 1 is the program for printing a text file in NLQ mode, and Figure 1 shows normal draft mode, print pass 1, print pass 2, and full NLQ print.

The program is straightforward and explained in the comments. Modifying it for other printers should not be too difficult as long as you use the same basic method for bit image graphics. The places where changes may be needed are in the procedures PREFIX, HALFDOTLF, and in the body of procedure PRINTNLQ.

Expect the printer to take at least 3 or 4 times as long to print your text. I have a parallel interface to my printer so data transfer doesn't take long, but with a serial printer (and no buffer) you'll notice a big delay between passes of the print head. The way the program is set up, as many as 964 bytes have to be sent for each print pass. At 9600 baud that's about a second just to get the data to the printer.

(continued next page)

Figure 1 - Examples of Printing

```
This is an example of draft mode print, 10 CPI.
This is one pass of NLQ print, pass # 1.
This is one pass of NLQ print, pass # 2.
This is an example of full NLQ print, both passes.
```

Listing 1 - Printing a Text File in NLQ Mode

```
program print_near_letter_quality;
{
  Written for Turbo Pascal V3.00 for CP/M-80 and designed to
  be run as a COM file.  To run in memory mode, make the changes
  as commented for running with Turbo version 2.  Uses printer
  codes for Star Micronics Delta 10 printer but can be modified
  for use with other bit mode graphics printers.
}

type
  pass = array [0..11] of byte; { dot columns for 1 pass of 1 char }

  chardesc = record           { file storage record of char dot data }
    ch : char;
    pass1 : pass;
    pass2 : pass;
  end;

  passes = record             { array element for memory dot patterns }
    pass1 : pass;
    pass2 : pass;
  end;

  anystr = string[255];

var
  descfile : file of chardesc;  { dot pattern file }
  infile : text;                { text to print }
  infilename : string[16];      { text file name }
  inpdesc : chardesc;           { temporary for reading dot data file }
  passdat : array[' '..'~'] of passes;  { memory dot patterns }
  line : anystr;                { text line to be printed }
```

(continued from page 61)

Augmenting Your Program

There are many ways this concept can be expanded as has been done in the commercial programs. You could develop many different type styles (Gothic, futuristic, etc.), set it up for different character pitches, set up for more than 2 print passes to extend the characters vertically, or use more than 12 columns to define a character and thus extend it horizontally. The font data file could also be expanded to define as many as 256 unique characters. (Actually, if the input is a text file you could never print the character associated with ctrl-Z, since that would be seen as end of file.)

Without the font file, this program is not of much value, so I have sent that file and associated maintenance programs to Micro C's bulletin board. Briefly, the files are:

HOW2NLQ.DOC — BRIEF documentation for this set of files.

MAKENLQ.PAS — creates an empty font file for data entry/update.

EDITNLQ.PAS — font data entry/update program.

DUMPNLQ.PAS — prints the entire font file, each char as a 12 X 16 char matrix. Pass 1 dots are printed as '*' and pass 2 dots as '+'. Simple modification to dump to screen.

PRNTNLQ.PAS — the same program as Listing 1.

ASCII2.NLQ — my current ASCII font file.

All are pretty much "bare bones," but they get the job done and provide a starting point for the real fancy stuff.

For those of you without a modem, if you send me a blank, formatted disk (either 5 1/4" Kaypro CP/M-80 format or 8" SSSD CP/M-80) in a REUSEABLE mailer and $5 to cover return postage and refreshments, I'll copy the files for you.

■ ■ ■

Listing continued from page 61

```
PROCEDURE printnlq(s:anystr);

{ This procedure prints each line of input text as two passes of
  graphics dot patterns with a half dot line feed between each. }

var
  i : integer;


PROCEDURE prefix(nchars:integer);

{ Send graphics mode command string to printer.  For Delta 10
  <ESC> 'L' selects 120 DPI then the two binary bytes that are
  the total number of dot columns to be printed, low byte first }

begin
  nchars := nchars * 12;   { 12 half dot columns / char }
  write(lst,^[,'L',chr(lo(nchars)),chr(hi(nchars)));
end;

PROCEDURE printpass(p:pass);

{ Send data for one pass of one character to the printer.  The
  calling routine has done the table lookup and passes the
  data as a parameter }

{ At this point you could prompt for alternate font file
  or pull an alternate font filename from the command line.
  There should probably be a check for font file present here. }

  assign(descfile, 'ascii2.nlq');
  reset(descfile);

  repeat       { Read dot patterns from font file, store in data array }
    read(descfile,inpdesc);
    passdat[inpdesc.ch].pass1 := inpdesc.pass1;
    passdat[inpdesc.ch].pass2 := inpdesc.pass2;
  until eof(descfile);
  close(descfile);

  {$i- turn off I/O checking so don't get runtime error if no text file}

  infilename := paramstr(1);  { comment out this line for version 2 }

  assign(infile,infilename);  { try to open input file }
  reset(infile);

  if ioresult <> 0 then       { if file open unsuccessful, scream }
  begin
    writeln('Input file empty!',^G);
    halt;
  end;

  {$i+ turn I/O checking back on }

  repeat                    { read and print each line from input file }
    readln(infile,line);
    printnlq(line);
  until eof(infile);
end.

var i : integer;
begin
  for i := 0 to 11 do write(lst,chr(p[i]));
end;



PROCEDURE halfdotlf;

{ Tell printer to advance paper 1/144 ".  For Delta 10 the
  command string is <ESC> 'J' followed by the binary # of
  144ths to advance }

begin
  write(lst,^[,'J',^a);
end;



begin                     { PROCEDURE printnlq }

  if length(s) > 0 then   { anything to print? }
  begin
    s := copy(s,1,80);    { not real fancy, truncate at max allowed chars }
```

```
    for i := 1 to length(s) do  { remove all char codes for which no patterns}
      if not(s[i] in [' '..'~']) then s[i] := ' ';

    halfdotlf;              { vertical registration is better if you do this }
    prefix(length(s));      { 120 DPI graphics mode command }
                            { now print the first pass of dots }
    for i := 1 to length(s) do printpass(passdat[s[i]].pass1);
    halfdotlf;              { advance paper for second pass }

    prefix(length(s));      { print the second pass of dots }
    for i := 1 to length(s) do printpass(passdat[s[i]].pass2);

    write(lst,^[,'J',chr(22)); { finally, do 11 half dot line feed
                                 to prepare for next line }
  end

  else writeln(lst);        { null string, just do line feed }

end;                        { PROCEDURE printnlq }

begin     { Main program }

{ PARAMCOUNT and PARAMSTR are features of Turbo V3.  If you are using
  version 2, comment out the following 5 lines and use the marked code
  instead }

  if paramcount < 1 then
  begin
    writeln('No text file!',^G);
    halt;
  end;

{ ******* Use these lines for version 2 **********
  write('Text file: ');
  readln(infilename);
                         }
```

End of Listing

NSWEEP And The 32032/MS-DOS Interface

Excerpts From Talks By Dave Rand At SOG IV

Definicon Systems, Inc.
21042 Vintage St.
Chatsworth CA 91311
818-341-5654

Dave Rand is well known to the public domain community for creating NSWEEP, a file transferring and handling utility that is at least as famous as Turbo Pascal. Dave's work on the DSI-32 has kept him in the computer community's spotlight, so we were doubly delighted to have him speak at this summer's Get-together.

I developed NSWEEP because I wasn't satisfied with SWEEP. The original SWEEP was written in PL1, the different research compiler, and it was really good. In fact, I thought it was one of the best programs I'd ever seen. The only problem was it took up 38K on my disk, and I couldn't afford that much space. But since I really liked its features, I decided to build an assembly language equivalent. That was the origin of NSWEEP.

NSWEEP In The Public Domain

The first public domain release of NSWEEP was version 1.41, and thank goodness no one still has it. Version 2 was the first one with the SQUEEZE algorithm added.

People have asked me why I didn't sell it. Good question and easy answer.

Because NSWEEP is in the public domain, I've received a lot of positive feedback from people who use the program. If it were commercial, I don't think I would have had that kind of response. I would have been a lot richer, but I'm not in it for the money.

The other reason I work through the public domain is that when I first got into CP/M programming I was lost. I had no idea what CP/M was all about. I started to fumble my way through CP/M, and it was pretty tough. When I found out about public domain and bulletin boards, I thought, "This guy is obviously insane. He's put up his computer as a remote access system. People are just calling in and taking his software."

I couldn't understand the concept, but as I used it I found I was actually learning from other people and from the way they were doing things. The original SWEEP program was a perfect example. I learned enough from that program to enable me to develop the NSWEEP program. But more than that, I learned how to deal with CP/M. So that was the other reason I contributed NSWEEP back to the public domain.

Q: What's the history of the 4KWASH program?

A: The 4KWASH program was a precursor of SWEEP. The very first of these file programs was called CLEAN, and was developed by someone at Cal Tech. It ran only on Z80s, not on 8080s. Its sole purpose was to scan down files and collectively erase them. Hence the name CLEAN, for keeping the disk clean.

The author lost the source code. I gave a copy of the COM file to Mike Karras, who thought it was a great idea, but terrible implementation. At that point, he only had an 8080 so he wrote the first version of WASH. In the process of doing that he decided that deleting files was real nice, but there were a few other things he wanted, too. So he added the LIST option.

He needed a name for the program and since the original name was CLEAN, he called his WASH. SWEEP came next, because if you're not doing your wash, you're sweeping it.

Disk 77 came just before NSWEEP. I had originally intended to modify Disk 77 to have all the features of SWEEP. I went through the source code (typical assembly language programming) and I said, "I can do a better job than this." So I started from a blank screen and added the file delete, then the login, then the space, etc., and built up the command set from that point.

Inside NSWEEP

Q: What are the undocumented flags in NSWEEP?

A: The most common question I get. I was called at 4 in the morning by a person who had set his files to system in read only, and he couldn't unset them.

The Y command is an extension of the Mass commands, (M and the Q). First you tag the files (with the wild card W or the T), then you hit Y, and it asks which flags you want — either 1 through 4, R, S, or A (R is read only; S is system; A is archive).

A lot of people have asked me what the 1 through 4 are for. They're referred to in the MP/M documentation as "compatibility attributes." They determine how MP/M responds to a COM file. If you have a COM file which opens the same file more than once, MP/M normally blows up on that error, because you're trying to open a file that's already open. By setting the compatibility attributes, you can say, "Look, I know that this program does that, but it's okay."

That's what they're used for in the MP/M sense. If you aren't running MPM, you can use them for highlighting files.

Let's say, you have a disk that contains your normal working software, and you like to give your public domain software to people. You want to set up a disk so you can just copy that disk. Compatibility attributes let you do that.

If you have the reverse video enhancement patched into your version of NSWEEP, you can see the attributes on the filenames as reverse video characters, underlined characters, or blinking characters. Then you run down your file list, tag the files you want with the 1 through 4, which are the first four characters of the filename, and then give those away as a common set.

There really aren't any undocumented flags. You can't have any more bits than those in the file name, because that's all CP/M knows about. You can set files that are currently Read Only or System back to directory status. You can use a D for any character that isn't — R, S, A, 1, 2, 3, 4.

Q: It can reset all the bits?

A: It resets all the bits except the ones you've specified, unlike other programs that only set certain bits, like STAT.

Q: Does any capability exist to tag all the files that have one of those bits on?

A: No. About 20 people have asked me for that. If you had an operating

(continued next page)

(continued from page 65)

system that resets the archive bit, you could tell it to tag all the entries that aren't archived and then back them up. You'd then have a better means of dealing with your files.

Q: I still don't fully understand archiving.

A: In NSWEEP, if you tag files and then do a mass operation on them, they get marked with a # sign instead of a star. That indicates that they used to be tagged but aren't anymore. The Again command retags those files that have a # sign beside them. So you would say Again, which would retag those files, Y, to go into set status, and then 8 to reset the archive bit on those files which you backed up.

Q: How do you deal with programs like WordStar that reset those bits?

A: WS doesn't reset them directly — only when you modify a file.

Q: So if you go in and edit a file, the backup file will have the bits set, but the new file won't.

A: That's correct. Now, believe or not, that's a feature because it allows you to use this archive bit feature of CP/M if your BDOS supports it. If it copied all the attributes of the old file, obviously you'd have problems.

Q: Why isn't there a feature to move a file from one user area to another user area without copying it?

A: That feature bent my head so much you can't imagine. The original version of NSWEEP had it, but I had to pull it out because I had to make direct file calls to do it, and it didn't work on CP/M 3.0, MP/M, or Turbo DOS. I could see no way of actually resetting that user number without going through a BIOS call.

Then Jim (a friend of mine who's written lots of good public domain programs, including the SQUEEZE section of the current NSWEEP) showed me up again. We were talking and he says to me, "Why don't you set the reserved bit and issue a close?" I said, "What?!" And he repeats, "All you have to do is set this undocumented bit here, and issue a close, and it works."

You can do it in one of two ways. The rename function has two modes. The first one is renamed on a file basis. So if you say R, it asks for a new name. Let's say you're in drive B: user 0 and you've got the file NSWEEP.COM. You want to change that to user 14. You say "14:NSWEEP.COM", and it moves the file from user 0 to user 14 in an operating system independent manner.

The second way is the wildcard mode. Let's say you want to take all the files in user 0 and move them up to user 14. You say "Rename[carriage return]." It says, "Old name." You say, "B0:*.*". It says, "New name." You say, "14:*.*[carriage return]". And it moves all the files from user 0 to user 14 with no interruption on your part, and it doesn't copy them.

Q: There are a couple of us who have non-video Kaypros. Is there any provision to patch NSWEEP so it can let us know about the bits?

A: A user group in Vancouver wrote me about how they did that. They made the "reverse video on" attribute be a left bracket. The "reverse video off" is a right bracket. It screws up the display a little bit, but it works.

Q: I've been using the Verify function, and it works great on mass copy, but I'm having trouble using it on individual files.

A: Real common problem. I had to make a decision on how I was going to implement that.

You're sitting on NSWEEP and you issue a C. It asks you for a new file spec. Now that differs from the move which asks you only for a drive and a user number. I was trying to accommodate people who are just copying in between drives with the copy command by allowing these forms: C, C0, C0:, and other similar forms. And you can even type things like 14, which means "Current drive, user 14." You don't have to put the colon; it just figures it out.

You type C0 V, and NSWEEP says, "I'm smarter than he is, so I'll copy it to a file called C0 on the current drive."

Here's how to get around it. You put in a colon which tells NSWEEP it's a drive user specification. Don't forget the space; it has to go in there. And that solves the problem, although it's not an ultimate solution. In the new version, there's an installable option that lets you chose whether you want to verify all the time or not.

Until Dave McCrady, a bulletin board operator in Edmonton, thumped on me for a month, I didn't put the

(continued on page 69)

(continued from page 67)

verify command in. When I design an operating system, it does verification on the sector level where you have an opportunity to retry and correct your mistake.

At the BDOS level and at the application program level, you have no opportunity to retry. All you know is something's gone wrong. Most BDOSs today are not as adequate as they could be. I've allowed you to install it to verify all the time, so if you're running on a BDOS that doesn't have a built-in verification feature, NSWEEP will do it for you.

32032 MS-DOS Interface On The DSI-32

I was responsible for porting all the compilers and interfacing the 32032 operating system to MS-DOS (the 8088/8086 operating system) on Definicon's DSI-32.

There are two parts to the software interface — the 32032 operating system which is resident on the DSI-32 card and handles all the requests of the 32032, and MS-DOS which is resident on the PC.

If you want to open a file, for example, you set up a few registers and issue a service call to open the file. We hand off the I/O processing to the 8088 or 8086 (which works well as an I/O processor since it has substantial memory for buffering).

The interface between the two operating systems is handled through a dual port memory arrangement (both the 32032 and the 8088 can access the 32's memory) and through the segmentation architecture. It was the only way we could interface the PC architecture to the 32032's wide addressing range. The PC can address only 1 megabyte at a time.

Through this interface we've defined a small section of RAM at the bottom of the 1 or 2 megabytes in our card for handling I/O requests.

When the 32032 wants to open a file, for instance, it puts the request in this special part of memory and interrupts the 8088. The 8088 then processes the request and resets a flag to tell the 32032 that it's finished.

The operating system kernel on the 32032 is much like a small UNIX kernel (it has the standard OPEN/CLOSE/CREATE/READ/WRITE type of interface common to most operating systems). All software (including compilers) works through this interface to talk to the devices on the PC. The compilers, for example, transfer through data this interface at 256K bytes per second — the limit of the PC.

Q: Is the DSI-32 operating system in ROM?

A: No, a PAL tells the 32032 to stop (it executes a DIAG instruction — a branch to itself) which prevents the processor from executing code while you're loading RAM from the 8088 side. When the operating system and the program you intend to run is in memory, the 8088 flips a bit, which tells the 32032 to start executing. No ROMS, and no need for ROMS.

Q: How much space does the operating system on the DSI-32 occupy?

A: The local operating system takes about 2K to 4K of memory, but we use a bit more than that for module tables and the C library. 32K is reserved, but you could reduce that to 16K if you needed those last few bytes.

Q: And if you want to use the PC for multi-tasking with the DSI-32?

A: You'll need Concurrent DOS for that.

Q: Can the 32032 read any segment of memory on the PC?

A: Dual porting is uni-directional. The card can't get control of the PC bus. The 8088 always retains control of the bus, an architectural limitation of the PC. But that's not the case in the AT.

The PC is the master of the bus, if you will, and can move anything within the PC address space to anywhere in the 32032 address space. But the 32032 must ask the PC to move memory. A full system call interface on the DSI-32 handles that, so you can do full screen updates, for instance, at 256K/sec (2.5 Megabaud).

■ ■ ■

High Resolution Graphics Using The NEC 7220

By N.T. Carnevale

State University of New York
Neurology Dept.
Stony Brook NY 11794

High resolution graphics has come into its own as smart graphics controllers have hit the scene. The 7220 does so much all by itself that designing new graphics circuits has become almost trivial and 1024 by 1024 by 4 colors almost commonplace. Hitachi has recently released the 63484 which has many times the resolution as well as a faster display time but the same ease of use. These are the chips that are leading the graphics revolution. If you're thinking of writing software drivers for a smart graphics board, stay tuned.

Sophisticated graphics capabilities are becoming increasingly common on microcomputers, and most of the credit for this can be given to graphics controller ICs. These devices perform tasks that used to require dozens of ICs — drawing characters, points, lines, and geometric figures, and generating the video signals needed to display the image on a monitor. By handling line and figure drawing, a graphics controller can free the microcomputer's CPU for other tasks.

The NEC uPD7220 Graphics Device Controller or its equivalent, the INTEL 82720 GDC, is appearing in more and more equipment with high resolution bitmapped graphics. Some of these products come with software, but there are several "OEM-style" boards that are essentially "code-it-yourself." This article presents the low level routines I use to run an S-100 graphics board with the 7220.

It should take only minor changes to make the graphics routines in this article to make them run on all the boards which contain the 7220.

Background

I set up an on-line data acquisition system in my lab about three years ago. The heart of the system was an I/O Technology A/D/A board (8 channels of 12 bit A/D and D/A conversion) mounted in a NorthStar Horizon.

This board performed both the data capture and the data display functions. The A/D conversion was fast, but the short-persistence oscilloscope I was using for a display had an annoying flicker. Also, the oscilloscope went blank each time the system sampled data.

As soon as low cost, high resolution graphics boards appeared for the S-100 bus, I decided to get one. The board I chose used the NEC 7220.

The Chip

The 7220 has two basic functions. First, it acts as an interface between the system bus and the graphics memory (the image to be displayed). Just a few bytes from the CPU are enough to make the 7220 write data to or read data from the display memory, or draw points, lines, rectangles, circles, arcs, or graphics characters.

The 7220 is also responsible for translating the contents of display memory into a video signal, complete with sync and blanking pulses. You control the screen format and sync pulses via software. You can even partition, zoom, pan, and scroll the screen on command.

Hardware Requirements

As versatile as it is, the 7220 needs a fair number of support devices including: clock, graphics RAM with refresh, data buffers, address decoders, and drivers for the monitor. Light pen, display zoom, and DMA transfer between graphics and system memory require additional hardware.

After considering several alternatives, I decided on the Pixeltronics 696GC96K. This S-100 board uses the NEC 7220 to display up to three planes of 32K bytes each for eight-color displays, or a single 32 to 96K plane of monochrome graphics. It provides all basic support circuitry. Its lack of a DMA controller is not a serious drawback for most applications.

Programming The NEC 7220

The Pixeltronics board came with full schematics and timing diagrams, theory of operation and application notes, and an NEC 7220 manual. There was also an 8" CP/M disk with Microsoft BASIC programs to put the board through its paces. The software supported several different monitors and sizes of graphics memory. It also included routines for monochrome, color, interlaced displays, and non-interlaced displays.

Unfortunately, I don't have Microsoft BASIC. Furthermore, my data acquisition software was written in Pascal.

Although the BASIC programs were reasonably modular, it seemed easier to rewrite the low-level graphics routines in Pascal from scratch than to attempt to translate from BASIC. It turned out to be more difficult than I expected because the NEC manual was rather obscure.

Later releases of the Pixeltronics board came with a set of pseudocode listings that would have made the job much easier. As it was, I found someone who was using a different 7220-based graphics board. He gave me a listing of C functions that answered many of my questions.

The low-level Pascal routines presented in this article are based partly on those C routines and partly on examples in the NEC literature.

Choice Of Language

I chose Pascal rather than C for two major reasons. First, all of our data acquisition software is written in Pascal, and I didn't want to rewrite it just to incorporate graphics. Second, Pascal tends to be more legible than C.

A C programmer with some knowledge of Pascal should be able to translate these routines with little difficulty, and may be able to generate more efficient code by careful use of C's special features.

I used Turbo Pascal because of its convenient editor, speed of compilation, and fast integer arithmetic. I have tried to avoid features peculiar to Turbo so these programs should compile under other Pascals with little change.

Low-level Routines

Unlike C80 or Pascal MT+, Turbo Pascal produces Z80 code directly, without generating relocatable files. Therefore it is not possible to build a library of routines that can be selectively searched by a linkage editor. Instead, all routines must be in the source file, or included at compile time with the $I directive.

I could have put all the low-level routines in a single giant file, but the COM files would have been larger than necessary. Instead, I grouped the routines into multiple "include" files, each of which contains closely related procedures. Then, by keeping track of "external" references, I was able to optimize code size by including only those source files which were necessary.

(Editor's note: Because of the length of the listings, we are not able to publish all of them. Therefore, only listings 1, 2, 3, and 18 appear here. The rest are available on the Micro Cornucopia bulletin board, 503-382-7643.)

Listing 1 contains board and monitor-specific constants. This file also includes the commands and mask bytes used to control and test the status of the 7220.

GDCTYPE.INC (Listing 2) contains TYPE declarations that simplify the coding of some procedures. Although I could have made this short file part of GDCCONST or GDCIO (Listing 3), I felt it was best to keep TYPE declarations separate as a kind of reminder.

Low Level Routines

Most of the low-level routines are straightforward, but some deserve special comment. The lowest-level functions and procedures are all in GDCIO.PAS. Users of other Pascals may find it necessary to replace Turbo Pascal's ported input/output statements —

```
in_datum:=port[input_port_number];
{get a byte}

port[output_port_number]:=out_datum;
{send a byte}
```

with syntax appropriate to their compiler. The functions 'lo' and 'hi' return the bottom or top byte of a 16-bit integer. The operators AND, OR, etc., perform bitwise operations on bytes or words. If necessary, SHR (shift right) and SHL (shift left) can be replaced by DIV and multiplication.

The only serious violations of Pascal style are in the procedures "wdat" and "rdat." These procedures take advantage of the representation of an array in memory as an adjacent block of bytes, and use pointer arithmetic to facilitate transferring any number of bytes between system memory and display memory. This application of pointers is a common practice in C, which is more hardware oriented than Pascal. The unusual construction —

```
pointer_variable:=
  PTR(ORD(pointer_variable)+1);
```

is a dodge to increment a pointer, circumventing Turbo Pascal's inability to handle pointer arithmetic directly. Other Pascals may require some other trick, or none at all.

The 7220 reads or writes data at the location in the display memory indicated by the cursor. The cursor address is specified by 18 bits that indicate the location of the target word in graphics memory, and a "dot address" that denotes which of the target word's 16 bits is to be tested or modified.

Procedure "curd" returns the cursor position.

Procedure "initialize" sets up monitor-specific parameters such as the timing of sync pulses, interlace/non-interlaced, lines per frame, and when display memory can be accessed. I set the DRAWACTIVE bit so display memory can be accessed at any time (I prefer to see the monitor flicker so I know something is happening.)

You can calculate the monitor related parameters from specifications of the monitor and algorithms from the 7220's manual. Because the Amdek 310A manual didn't provide the relevant specifications, I had to work these out by trial and error.

SETMON.PAS (Listing 18) makes sync mods easier for those of you who also have to use the trial and error method.

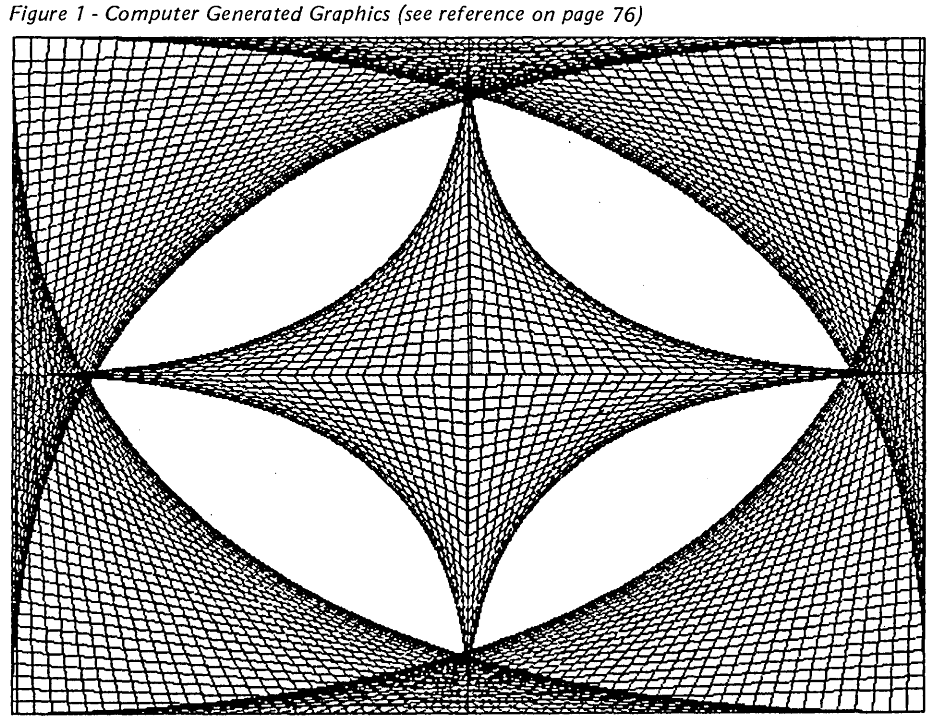
Other parameters set by this procedure include the starting address of display memory, whether or not to show the cursor, and what zoom factors to use for graphics display or write operations. This procedure ends by telling the 7220 to use a solid pattern for drawing lines, and to perform logical ORs when writing to display memory (SET mode).

Demonstration Programs

The first program I ran to test the Pixeltronics board was the simple point plotter POINTEST (Listing 17 on bulletin board). This reassured me that data was being written to display memory, but the sync parameters were incorrect so the display was scrambled.

(continued on page 76)

Figure 1 - Computer Generated Graphics (see reference on page 76)



Listing 1 - GDCCONST.INC

```
{ GDCCONST.INC--file of various constants
  for NEC7220 chip on Pixeltronics board }

CONST

{******************************************
  board- and monitor-specific definitions
******************************************}

{ i/o port assignments }
  CMD=$11;
  PARAM=$10;
  FIFO=$10;
  GDCRD=$11;

{ monitor constants--incl resolution, sync params etc. }
  MONHRES=576;     { horizontal resolution }
  MONVRES=432;     { vertical resolution }
  MONHS=4;         { horizontal sync width }
  MONVS=6;         { vertical sync width }
  MONHFP=3;        { horizontal front porch width }
  MONHBP=6;        { horiz back porch }
  MONVFP=5;        { vert front porch }
  MONVBP=12;       { vert back porch }

{ line type definitions }
  SOLID=$FFFF;
  DOTTED=$9999;
  DASHED=$C3C3;
  MIXED=$E187;

{ size of display }
  PARTLNTH=$3FF;   { length of display partition 1 in lines }
  WPL=$24;         { length of a line in 16 bit words,
                      i.e. "pitch"--this is just MONHRES DIV 16}

{******************************************
 commands, masks and bit definitions for
             the NEC 7220
******************************************}

{ status register bits }
  DATAREADY=$01;
  FIFOFULL=$02;
  FIFOMPTY=$04;
  DRAWING=$08;
  DMAEXEC=$10;     { does not apply to Pixeltronics board }
  VERTSYNC=$20;
  HORIZBLANK=$40;
  LIGHTPEN=$80;

{ command bytes }
  RESETC=0;        { reset to idle state }
  VSYNCC=$6E;      { lsb has master/slave definition }
  CCHARC=$4B;      { specify cursor & character row heights }
  STARTC=$6B;      { end idle mode & unblank display }
  ZOOMC=$46;       { specify zoom factors }
  CURSC=$49;       { set cursor position }
  PRAMC=$70;       { define start address & display area lengths }
  PITCHC=$47;      { specify pitch }
  WDATC=$20;       { write data bytes into display memory }
  MASKC=$4A;       { set mask register }
  FIGSC=$4C;       { specify parameters for drawing controller }
  FIGDC=$6C;       { draw figure }
  GCHRDC=$68;      { draw graphics character }
  RDATC=$A0;       { read data bytes }
  CURDC=$E0;       { read cursor position }
  LPRDC=$C0;       { read light pen address }
  DMARC=$A4;       { DMA read & write operations }
  DMAWC=$24;       { DMA not implemented on Pixeltronics board }

{ reset mode definitions }
  GRAFMOD=$02;
  CHARMOD=$20;
  MIXMOD=0;
  ILACE=$09;       { interlaced }
  ILRPT=$08;       { interlaced repeat field for character displays }
  NILACE=0;
  STATIC=0;
  DYNAMIC=$04;
  DRAWACTIVE=0;    { draw during display time & retrace blanking }
  DRAWRETRACE=$10;{ draw only during retrace blanking }

{ vsync mode definitions }
  VSSLAVE=0;       { accept external video sync input }
  VSMASTER=1;      { generate and output video sync }

{ wdat command modification definitions }
  REPLACE=0;
  COMPLEMENT=1;
  RRESET=2;
  SSET=3;

{ data transfer type field definitions }
  WWORD=$00;       { low, then hi byte }
  LOWBYTE=$10;     { low byte only }

{ figs command parameter bits }
  CHARACTER=0;
  LINE=$08;        { line or vector }
  GRAFCHAR=$10;    { graphics character }
  ARC=$20;         { arc/circle }
  RECTANGLE=$40;
  SLANTED=$90;     { slanted graphics character }
```

Listing 2 - GDCTYPE.INC

```
{ GDCTYPE.INC--type definitions used for NEC 7220 routines }

TYPE
  byteptr=^byte;
  bytearray=array [0..15] of byte; {used in procedure pram}
  intarray=array [0..5] of integer; {used in procedure figs}
  extadr=array [0..2] of byte; {used for first arg to curd and lprd}

{end of GDCTYPE.INC}
```

Listing 3 - GDCIO.PAS

```
(*
   GDCIO.PAS contains the most elementary commands needed
   to run the NEC 7220 on the Pixeltronics board.
   Contains the following routines:

PROCEDURE fifordy;  { waits until 7220's fifo is not full }
PROCEDURE fifoempty;  { waits until fifo is empty }
PROCEDURE outcmd(com:byte);  {sends a byte to the 7220's
                              command port }
PROCEDURE outpar(par:byte);  { sends a byte to 7220's data port }
PROCEDURE greset(rmode,aw,hs,vs,hfp,hbp,vfp:byte;
                 al:integer; vbp:byte);  { resets 7220.
                 All parameters are passed as arguments }
PROCEDURE vsync(vmode:byte);  { sets up master or slave mode }
PROCEDURE cchar(lr,dc,ctop,sc,br,cbot:byte);
                { sets up cursor & character characteristics }
PROCEDURE start;  { exits the idle mode }
PROCEDURE zoom(gchr,disp:byte);
            { zoom factors for graphics characters & display }
PROCEDURE curs(ead,plane,dad:integer);  { specifies cursor position }
PROCEDURE pram(sa,n:byte; list:bytearray);  { loads parameter RAM }
PROCEDURE pitch(p:byte);  { sets horz memory width }
PROCEDURE wdat(mode,typ:byte; n:integer; list:byteptr);
            { writes data into display memory }
PROCEDURE mask(maskv:integer);  { loads mask register }
PROCEDURE sendpar(i:integer);
            { used by figs to send integer parameters }
PROCEDURE figs(ftype:byte; dlist:intarray);
            { specifies drawing parameters }
Not included:  dmar, dmaw (use DMA feature which is
                 not implemented on the Pixeltronics board)
*)
PROCEDURE fifordy;  { waits until 7220's fifo is not full }
VAR status:byte;
BEGIN
  REPEAT status:=port[FIFO] UNTIL (status AND FIFOFULL)=0;
END;


PROCEDURE fifoempty;  { waits until fifo is empty }
VAR status:byte;
BEGIN
  REPEAT status:=port[FIFO] UNTIL (status AND FIFOMPTY)<>0;
END;


PROCEDURE outcmd(com:byte);  { sends a byte to command port }
BEGIN
  fifordy;      { wait for fifo }
  port[CMD]:=com;
END;

PROCEDURE outpar(par:byte);  { sends a byte to data port }
BEGIN
  fifordy;
  port[PARAM]:=par;
END;


{ resets 7220.  All parameters are passed as arguments }
PROCEDURE greset(rmode,aw,hs,vs,hfp,hbp,vfp:byte;
                 al:integer; vbp:byte);
{     rmode         mode for 7220 (graphics, character, mixed)
      aw            active words per line
      hs            horiz sync width
      vs            vert sync width
      hfp           horiz front porch
      hbp           horiz back porch
      vfp           vert front porch
      al            active lines per field
      vbp           vertical back porch
}
BEGIN
  outcmd(RESETC);           { sends reset command }
  outpar(rmode AND $3f);    { mode word }
  outpar(aw);               { active words/line }
  outpar( (hs AND $1f) OR ((vs AND 7) SHL 5) );
                            { hsync width & low bits of vsync width }
  outpar( ($fc AND (hfp SHL 2)) OR (3 AND (vs SHL 3)) );
                            { vs hi bits & h front porch }
  outpar(hbp AND $3f);      { h back porch }
  outpar(vfp AND $3f);      { vert front porch }
  outpar(lo(al));           { low byte of active lines/field }
  outpar( (3 AND hi(al)) OR ($fc AND (vbp SHL 2)) );
                            { hi bits of al & v back porch }
END;


PROCEDURE vsync(vmode:byte);  { sets up master or slave mode }
BEGIN
  outcmd(VSYNCC OR (vmode AND $01));
END;


{ sets up cursor & character characteristics }
PROCEDURE cchar(lr,dc,ctop,sc,br,cbot:byte);
{     lr          lines/character row (1 if graphics row)
      dc          dc=1 -> display cursor
      ctop        cursor top line # in row
      sc          sc=1 -> steady cursor, 0 -> blinking
      br          blink rate and attributes
      cbot        cursor bottom line # in row
}
BEGIN
  outcmd(CCHARC);
  outpar((lr AND $1f) OR ($80 AND (dc SHL 7)));
    { lines/row & displ curs }
  outpar((ctop AND $1f) OR ($20 AND (sc SHL 5))
          OR ($c0 AND (br SHL 6)));
    { cursor top, steady cursor, low bits of blink rate }
  outpar((7 AND (br SHR 2)) OR ($f8 AND (cbot SHL 3)));
    { upper bits of blink rate, cursor bottom }
END;


PROCEDURE start;  { exits the idle mode (opposite of greset) }
BEGIN
  outcmd(STARTC);
END;
```

(continued next page)

(Listing 3 continued from page 73)

```
( specifies zoom factors for writing graphic characters
  & for display )
PROCEDURE zoom(gchr,disp:byte);  ( zoom factors )
BEGIN
  outcmd(ZOOMC);
  outpar((gchr AND $0f) OR ($f0 AND (disp SHL 4)));
END;

PROCEDURE curs(ead,plane,dad:integer);  ( specifies cursor position )
(    ead      unsigned 16-bit integer holds low part of word address
     plane    upper two bits of word address
     dad      dot address of pixel within the word address
                (graphics mode only)
)
BEGIN
  outcmd(CURSC);
  outpar(lo(ead));  ( addr low byte )
  outpar(hi(ead));  ( addr hi byte )
  outpar((plane AND $03) OR ($f0 AND (dad SHL 4)));
                    ( addr top bits & dot addr )
END;

PROCEDURE pram(sa,n:byte; list:bytearray);  ( loads parameter RAM )
(     sa          starting address in parameter RAM (0-15)
      n           how many parameter bytes (1 to 16-sa)
      list        pointer to string of n bytes to be loaded
)
VAR i:integer;
BEGIN
  outcmd(PRAMC OR (sa AND $0f));  ( sends command & start addr )
  FOR i:=0 TO n-1 DO outpar(list[i]);
END;

PROCEDURE pitch(p:byte);  ( sets horz memory width )
BEGIN
  outcmd(PITCHC);
  outpar(p);       ( send width )
END;

( writes data into display memory )
PROCEDURE wdat(mode,typ:byte; n:integer; list:byteptr);
(     mode        modification operation:
                      0--replace
                      1--complement
                      2--reset to 0
                      3--set to 1
```

```
     typ        type of data xfer--
                        $00--word, low byte then hi
                        $08--invalid
                        $10--low byte only
                        $18--hi byte only
     n          number of bytes (2*num words) to be sent
     list       pointer to string of n bytes to be sent
}
BEGIN
  outcmd(WDATC OR (typ AND $18) OR (mode AND $03));
    { send cmd & mode & typ params}
  WHILE n>0 DO BEGIN
    outpar(list^);  { send list of data }
    list:=PTR(ORD(list)+1);
    n:=n-1;
  END;
END;


PROCEDURE mask(maskv:integer);  { loads mask register }
BEGIN
  outcmd(MASKC);
  outpar(lo(maskv));  { mask's low byte }
  outpar(hi(maskv));  { mask's hi byte }
END;


{ used by figs to send integer parameters }
PROCEDURE sendpar(i:integer);
BEGIN
  outpar(lo(i));            { low byte of parameter }
  outpar($3f AND hi(i));    { hi byte of parameter }
END;


{ specifies drawing parameters
PROCEDURE figs(ftype:byte; dlist:intarray);  }
{     ftype    CHARACTER=byte or individual dot
               LINE=line
               GRAFCHAR=graphics byte & area fill
               ARC=arc & circle
               RECTANGLE=rectangle
               SLANTED=slanted graphics byte
      dlist    points to an array of integer parameters:
               element       name     function
               -------       ----     --------
               dlist[0]      dir      drawing direction (0-7):
                                                5  4  3
                                                6  X  2
                                                7  0  1
               dlist[1]      dc       drawing count param
               dlist[2]      d        needed by all except char mode
               dlist[3]      d2          "    "   "    "     "    "
               dlist[4]      d1       needed by line, arc, rectangle
               dlist[5]      dm       needed by arc, rectangle
}
BEGIN
  outcmd(FIGSC);
  outpar(ftype OR (dlist[0] AND $07));      { type & direction }
  sendpar(dlist[1]);                        { dc }
  IF (ftype<>CHARACTER) THEN BEGIN
    sendpar(dlist[2]);                      { d }
    sendpar(dlist[3]);                      { d2 }
    IF ((ftype<>GRAFCHAR) AND (ftype<>SLANTED)) THEN BEGIN
      sendpar(dlist[4]);                    { d1 }
      IF (ftype<>LINE) THEN sendpar(dlist[5]);  { dm }
    END;
  END;
END;

(end of GDCIO.PAS)
```

Listing 18 - SETMON.PAS

```
PROGRAM setmon;  { used to adjust monitor-specific parameters }
{ N.T.Carnevale 5/18/84 }

{$I GDCCONST.INC}
{$I GDCTYPE.INC}

TYPE strng80=string[80]; (used for prompts)

VAR
  i:integer;
  hsnc,hfrp,hbkp,vsnc,vfrp,vbkp:byte;
  ans:integer;
  ptch:integer;

{$I GDCIO.PAS}

{$I LGCHRD.PAS}
{$I LSETCURS.PAS}
{$I LMODNPAT.PAS}
{$I LCLRPLAN.PAS}
{$I LDRAWLIN.PAS}


PROCEDURE setparams;
BEGIN       { values determined empirically }
  hsnc:=4;
  hfrp:=3;
  hbkp:=6;
  vsnc:=6;
  vfrp:=5;
  vbkp:=12;
END;

FUNCTION intprmpt(prompt:strng80; min,max:integer):integer;
VAR num:integer;
BEGIN
  write(prompt);
  REPEAT readln(num) UNTIL (num>=min) AND (num<=max);
  intprmpt:=num;
END;

PROCEDURE chngparms;
BEGIN
  writeln('HORIZONTAL--1. sync=',hsnc:3,'   2. frntprch=',hfrp:3,
          '    3. bkprch=',hbkp:3);
  writeln('  VERTICAL--4. sync=',vsnc:3,'   5. frntprch=',vfrp:3,
          '    6. bkprch=',vbkp:3);
  ans:=intprmpt(
       'Number of item to change (0 to go ahead, -1 to quit): ',
       -1,6);



  CASE ans OF
    0: writeln('No change');
    1: hsnc:=intprmpt('h sync: ',1,100);
    2: hfrp:=intprmpt('h front porch: ',1,100);
    3: hbkp:=intprmpt('h back porch: ',1,100);
    4: vsnc:=intprmpt('v sync: ',1,100);
    5: vfrp:=intprmpt('v front porch: ',1,100);
    6: vbkp:=intprmpt('v back porch: ',1,100);
    ELSE writeln('??');
  END;
END;

PROCEDURE init;
VAR
  rmode,aw,hs,vs,hfp,hbp,vfp,vbp:byte;
  al:integer;
  buf:bytearray;
  sad,len,wd:integer;
  lr,dc,ctop,sc,br,cbot:byte;
  zfw,zfd:byte;
BEGIN
  rmode:=(GRAFMOD OR NILACE) OR (DRAWACTIVE OR DYNAMIC);
  aw:=WPL-2;
  hs:=hsnc-1;
  vs:=vsnc;
  hfp:=hfrp-1;
  hbp:=hbkp-1;
  vfp:=vfrp;
  al:=MONVRES;
  vbp:=vbkp;
  greset(rmode,aw,hs,vs,hfp,hbp,vfp,al,vbp);

  { set video sync mode }
  vsync(VSMASTER);

  { set pitch }
  ptch:=WPL;
  pitch(ptch);

  { set up display partition 1 }
  sad:=0;           { start address of display partition 1 }
  len:=PARTLNTH;    { # lines in display partition 1 }
  wd:=0;            { for standard display cycle width }
  buf[0]:=lo(sad);
  buf[1]:=hi(sad);
  buf[2]:=len SHL 4;
  buf[3]:=(wd SHL 7) OR ($3f AND (len SHR 4));
  pram(0,4,buf);


  { set cursor characteristics }
  lr:=0;    { in graphics mode, # lines/row:=0 }
  dc:=0;    { don't display cursor }
  ctop:=0;  { these params don't matter if cursor isn't shown }
  sc:=0;
  br:=0;
  cbot:=0;
  cchar(lr,dc,ctop,sc,br,cbot);

  { set zoom factors }
  zfw:=0;   { for writing }
  zfd:=0;   { and for display }
  zoom(zfw,zfd);

  start;
  drawmode(SSET);   { select RMW operation }
  setpattern(SOLID);      { for solid line }
END;

PROCEDURE drawframe;
BEGIN
  drawline(0,0,0,MONVRES-1);
  drawline(0,MONVRES-1,MONHRES-1,MONVRES-1);
  drawline(MONHRES-1,MONVRES-1,MONHRES-1,0);
  drawline(MONHRES-1,0,0,0);
END;

BEGIN
  setparams;
  ans:=0;
  REPEAT
    init;  (a copy of initialize,
           modified to allow parameter changes)
    clrplane(0);
    drawframe;
    REPEAT chngparms UNTIL ans<=0;
  UNTIL ans<0;
END. (setmon)

{ end of SETMON.PAS }
```

(continued from page 71)

Obviously, the next program I ran was SETMON (Listing 18) which I used to set the sync parameters.

It draws a line around the edge of the display. I first adjusted the width of the sync pulses until the image stopped rolling and I was no longer getting bizarre folds. Next I changed the front and back porch settings (the amount of sweep time before the first pixel is displayed and after the last pixel is displayed) to center the display.

The program GRID draws a square grid on the monitor, which is handy for adjusting horizontal and vertical width and linearity controls. SHEAF tests the "drawline" procedure by generating a series of outline hyperbolas (Figure 1). CIRCTEST and RECTEST illustrate the speed of the 7220 in drawing specific geometric figures. The ability to combine figures and characters into a single display is a very useful feature of the 7220.

CHRTST.PAS and the accompanying FONT1.ASM show how I generated text characters using bit-mapped graphics.

In these programs, each character is a a matrix 8 pixels wide by 8 pixels high (8 bytes). Each byte corresponds to one horizontal row, byte 1 at the top, byte 8 along the bottom. Drawing direction is left to right, so the low order bit of each byte is on the left edge of the matrix. The character set in FONT1 is a "medium fancy" font with serifs, shown in Figure 2.

FONT1.ASM is just a convenient way of holding the HEX codes that represent the bitmapped characters. The ninth byte in each row of FONT1.ASM is the width of the character (used for proportional spacing).

I used ASM and LOAD to produce FONT1.COM and then I renamed it FONT1.DAT. CHRTST reads FONT1.DAT as a file of records consisting of the bit patterns plus the corresponding character widths.

Making FONT1 Turbo Digestible

The conditional portion at the start of FONT1.ASM generates two integers that denote the number of records (characters) and the length of each record (8 bytes for the pattern + 1 byte for character width), which Turbo Pascal expects to find at the beginning of every typed file. Other Pascals may not require or may even be confounded by such information.

Size & Position

CHRTST writes the character set to display memory with a user-specifiable zoom factor, so that the characters can be enlarged, making the position of each pixel more apparent. Descenders are handled by repositioning the cursor downward by an amount appropriate for each character before starting to draw.

Sending The Graphics To The Printer

SCRNDUMP shows how data can be read from display memory and transferred to another graphics device (in this case a dot matrix printer) to produce a hard copy. This program prints a 7-dot high swath with per each pass of the head. I used it to generate the printouts of the character set in Figure 2 and the display from SHEAF (Figure 1).

The Panasonic KX-P1091 which I used is similar to the Epson dot matrix printers, but you may need to change SCRNDUMP if yours is different. (The printer related code is near the top of the program.) Procedure printit will also need to be changed if your printer addresses the needles differently than mine (usually it's a matter of whether the top or bottom needle is bit 0).

SAVSCRN generates a data file that holds the contents of one 32K page of display memory. FILSCRN reads these files from disk into the display.

I haven't tried to do any data compression, but preliminary tests with SQ and USQ indicate that a 32K screen can be reduced to 4K or less. I recommend that someone add Huffman encoding and decoding to the SAVSCRN and FILSCRN programs (and send them in).

Figure 2 - Printing Done With FONT1, a medium fancy font

```
!"#$%&'()*+,-./01234
56789:;<=>?@ABCDEFG
HIJKLMNOPQRSTUVWX
YZ[\]^_`abcdefghijkl
mnopqrstuvwxyz{|}~

!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^
`abcdefghijklmnopqrstuvwxyz{|}~
```

How It All Turned Out

I have added these graphics routines to my data acquisition and analysis programs and the results have been excellent. I've added special functions to: set up markers along the time axis to mark the start and end times for measuring peaks or averages; superimpose multiple waveforms so that latency and amplitude differences are obvious; and print out selected waveforms.

I use keyboard control for cursor positioning, but it would not be difficult to adapt these programs to use some other input device such as a trackball, joystick, or graphics tablet.

Color

The three planes of the Pixeltronics board can be used to generate eight colors, counting black and white. Planes 0, 1, and 2 correspond to red, green, and blue, respectively, so drawing a dot at the same address in all three planes produces a white point on the monitor. Drawing to planes 0 and 1 results in a yellow dot.

Drawing in colors other than the primaries requires drawing the figure repeatedly in several planes. Therefore, the cursor must be placed at the same starting position in each plane. If starting position is not an explicit argument to your drawing procedures, it must be a global variable.

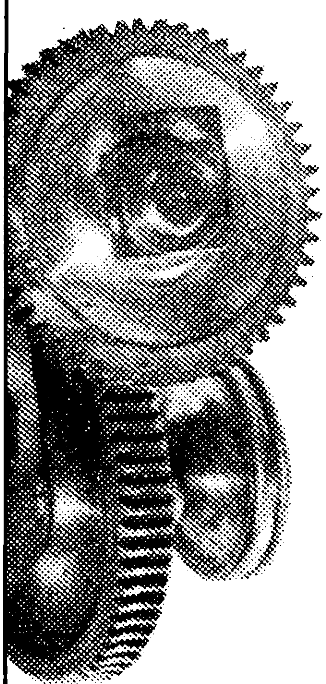
Finally

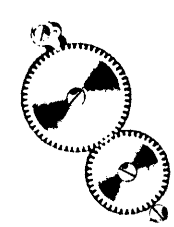
The Pixeltronics board has worked without a hitch for more than two years. Because of its excellent performance, I ordered the expansion set that brings it up to full 96K monochrome or 3x32K color capacity. Now that the ICs have arrived, I have a difficult choice to make — leave the board in monochrome configuration, or buy a color monitor. I've never seen a nerve cell produce eight-colored waveforms, at least not yet.

Reference

1. NEC manual for uPD7220/GDC, 1982.

■ ■ ■





On Your Own

By David Thompson

This column has been a sort of "Travels with Charlie" as I've written about my experiences starting Micro C. This time it's a "Conversation with Lewis." Please forgive the rambling, as we touch on several subjects more than once, but it's the way the morning went and I think you'll enjoy it.

Lewis Sternberg is a senior in electrical engineering at Oregon State University, but he spent the summer writing a fancy genealogy program in Turbo Pascal. (It charts the ancestral theory of relativity.) He called and asked if he could come to Bend and find out how to sell his program. Smelling a potential "On Your Own" column, I said, "Sure."

He did a lot of things correctly. First, he asked for help. Second, he made up a list of questions ahead of time and sent me a copy. Third, he laid out exactly what he was doing and why. I didn't have to waste my time or his playing 20 Questions.

The following is a lightly edited version of the discussion. (I only edited me enough to make me sound good.)

NOTE: In the first part of this article, Lewis' comments and questions appear in boldface. Later in the article, they are in quotes.

First, what is your audience?

Mostly people who have their computers sitting in their closets.

It's going to be fun to reach them. In fact, the first thing I consider when I'm looking at a new product idea is whether I can reach the audience at a reasonable cost (I don't know of a magazine targeted at closet computerists). Then I look at the competition (can I do better for less?). Finally, I look at demand.

If I can reach the audience, beat the competition, and I see a strong demand (those currently in that market are doing well), then I'll look at development costs (a new system? a new employee? no sleep for three months?) and development time (will the market still be there when I'm finished?).

If it still looks good, I'll add up the costs of manufacturing, advertising, development, documentation, support, and shipping and handling. The rule of thumb is that you at least triple this total (we've been running under this, but I don't recommend it). Many large (and sometimes profitable) companies will not manufacture a product if they can't charge at least 5 times their manufacturing cost, and they prefer to work in the 8 to 10 times range.

You calculate the advertising cost by dividing the total advertising budget by the number of units you expect to sell (your guess on this one). Obviously, if your product sells very well or is very expensive (or both), then advertising may only be 1/10 to 1/3 of the total cost. On the other hand, it's not unusual for advertising cost per item to be 10 times the retail price!

How do I find out how much advertising costs?

It's easy. Call or write to the publications that look like good prospects and ask for rate cards. Rates are generally determined by the circulation (paid subscribers + newsstand sales + free copies), competition, and the desirability of the audience.

A general, low-income, low-tech audience will be the cheapest to reach (per person). A specific, high-income, high-tech audience, in a small-circulation magazine will be the most expensive to reach (per person). Paid subscribers are the most desirable, newsstand sales are second best, and free distribution (often called controlled circulation) is definitely at the bottom. A lot of free issues see the world from the inside of a round file. If the publication doesn't break down circulation by type, call them and find out.

Media kits are a good way to get a lot of information about a publication, but don't ask for any you don't really need, as they are very expensive to produce and mail.

The rate card will tell you whether or not advertising in that publication is even vaguely feasible. If the rates look good, then ask for the kit. Advertising rates run from about $560 per full page in Micro C to almost $10,000 per full page in Byte. You have to have the ad designed, typeset, and laid out, so figure those costs, too. (Freelance graphic designers or typesetting companies are a lot cheaper than advertising agencies.)

What's a good response?

Boy, that's a good question. I guess you could say that a good response is one that makes money. Lots of things affect the response — price of the item, customer need, quality of the ad, size of the ad, location of the ad in magazine (important in catalogs like Byte or Computer Shopper), and the publication itself.

But to give you some round figures, you're very fortunate to get between 1/4 percent and 2 percent of your target audience per insertion (that may be only a small part of the whole readership), with the third ad (in a row) drawing about twice as well as the first. Most advertisers don't test a magazine with fewer than three insertions. Usually, skipping an issue sets you back almost to square one in terms of response.

How do I know if an ad is working?

Assuming you have ads in more than one publication, mark each coupon (use coupons if at all possible, since they improve response) with a special code, or add a department number to the address. You can ask anyone who calls where they heard about you (we do that), or you can do like Borland and have customers specify an extension number.

As a rule of thumb, an ad is effective about three times as long as the magazine's frequency; i.e., Micro C comes out every two months, so a Micro C ad should generate responses for six months. In Micro C, ads often work longer because people keep their issues for reference (and, of course, we sell back issues). Anyway, an ad in a monthly magazine will work about three months, and an ad in InfoWorld will work about three weeks.

People are interesting people. (You heard it here first.) Some people will purchase your item the instant they hear about it. They will call up immediately and insist that Federal Express simply isn't fast enough. But many folks won't even notice you until

(continued on page 81)

(continued from page 79)

they've seen you three times. Ads reinforce themselves.

What about VISA and MasterCard?

This is a tuffy. Chances are about 1 in a googolplex (a bunch) that you can become a VISA merchant without a storefront. Period. Too many ripoffs going around for them to trust anyone, so they make a blanket rule (and there are still ripoffs).

About your only hope is to tie up with a computer dealer in your community who is already a merchant. Set it up so you both make money. But first, make sure he (she) is someone you can live with, since this is going to be a long-term relationship. You may even get more than just access to VISA: you might be able to use his phone number, his order desk, and his address. He has to be there from 9 to 5 anyway.

If you're selling software, maybe you could also sell it bundled with hardware (the dealer gets the hardware sale). The software is $300 per copy, for instance, but complete with a Commodore and a printer the total might be only $325. What's the customer going to buy? Even if he already has a computer he'll buy the package and give the Commodore to his kid.

That way the dealer shouldn't mind handling details like taking orders and shipping the product. Plus, he gets national exposure.

What about mail order sales?

If you decide to go it alone you can force your customers to mail in orders with checks enclosed. Just give a post office box number or street address. That way you don't have to deal with phones or VISA cards.

However, this may not work. Unless

(continued next page)

(continued from page 81)

you're doing something that they can't get somewhere else, or your price is substantially better, then chances are they're going to purchase from someone who takes calls and cards.

You start out with no reputation and no recognition. People's first impression of you is the one your ad makes. That helps readers decide, first, if they're interested in your product, and subsequently, the ease with which they can get that product, and how well they think they'll be supported.

People like to be able to call you, even if they'll be ordering by mail. If they need support they don't want to wait for the mail; they want to know that they can talk to someone.

What are the costs of doing business?

If you can tie in with someone selling a product to the same market (preferably something that's not directly competitive), it's possible that they'll mail your flyer along with theirs, or they might even sell your product for you.

If you're having them send something out for you, you could offer them something like 10 percent to 30 percent of every order they generate. The customer contacts you directly, you handle the order, you do the shipping, all that stuff.

A mailing to 1,000 people will probably generate between 20 and 50 orders (2 percent to 5 percent), but the costs can be substantial. That is, buying the label (10 cents), putting the label on (2 cents), providing return postage (permit fee and 35 cents per piece returned), printing, graphic design, typesetting, illustrating, writing, envelope: all add up quickly. In fact, it would be very easy to spend $2000 or $3000 for your first 1000 pieces (between $40 and $100 per order).

Of course, as you increase the size of the mailing, the cost per contact drops, but even a huge mailing will cost you 40 cents per mailed piece, minimum.

Makes Magazine Advertising Look Good

As I mentioned earlier, a rule of thumb is that between 1/4 percent and 2 percent of your real audience will order from each ad. If you pay 3 cents per subscriber for a half-page ad (certainly cheaper than direct mail) then it'll cost you between $1.50 and $12 per order for space. To that you have to add production (another 25 cents to two dollars per order).

Choice of magazine is just as important as choice of mailing list. If you are selling genealogical software then you probably want to be in a genealogical magazine.

How much should I figure for distribution costs?

Once orders come in, you're going to have a new set of costs. There is the time involved in taking the order (phone or mail) and packaging it up. Then there's the disk, box, label, and postage.

You'll get some of the packages back because: the customers refused them (COD orders), a postal truck drove over them (the boxes looked like a challenge), the disk wouldn't boot (you didn't include a $150 copy of CP/M or a $60 copy of MS-DOS with your $30 product, shame on you!), or the address they gave you wasn't correct.

If you send out orders before the checks clear, you'll have to eat some bad checks (catsup helps). Some people won't like your software and will insist on returning it for a refund (after making a copy or two).

Should I include a manual?

It's very tempting to put the manual on the disk (it's what we're planning for Private Domain). You save on printing, packaging, and postage. However, a large printed manual discourages mass duplication. No one wants to stand in front of a Xerox machine for 25 hours.

You don't have to typeset the manual — a letter quality printer is just fine. One of those quick print places is great for small runs of small manuals, but for larger pieces (20 pages and up) and larger runs (5,000 and up) it may be cheaper to work with a medium size print shop. Whichever route you choose, check with them in advance for tips on how to prepare your original.

Printed manuals are also easier to use than the disk version. The user can refer to the manual while working on the program. You can add an index, quick reference section, and graphics to a printed piece. The manual and the packaging also suggest something about the quality of the software (rightly or otherwise). That's why Ashton-Tate spends huge amounts on fancy boxes and bindings. It makes their software look good and if you are charging a bunch, you owe the buyer a manual.

Is it okay to include the source?

It depends. If the user is a neophyte or if the program is a real breakthrough, I wouldn't release source. But if you've written yet another keyboard translator or spreadsheet, then I'd say release the source, especially if you are selling to a sophisticated user. Providing the source gives you a sales edge, and it might be enough of an advantage to increase your sales by a bunch.

Of course, purchasers could add a few bells and whistles to your program and then jump into the market faster than if they had to write from scratch, but they would still be faced with all the rest of the startup costs that you faced. Plus, you would have the advantage if they didn't release source.

How do I research the market?

Have you looked at the whole market? How many other people are already selling this kind of package?

(Lewis) "I put out 50 requests for information and wound up getting almost half of them back. Which is good, since the list was pretty old. Some were charging quite a bit for a crummy product. I think if I have something sufficiently better, then people will switch over."

What are the prices running?

"The cheapest was $25, the most expensive was $300. The difference between the actual products was not that substantial."

How did you find out about them?

"I got this reference book on genealogy, and in it was a chapter on what software is available. I wrote to everyone on that list."

I assume there are a couple of genealogical magazines.

"Yes, but I haven't really done my research."

You would want to see how many are still advertising. The key is how many people have advertised continuously for the last year. Those are the people who are at least breaking even.

Then you should get a media kit from these magazines to see how much the advertisers are spending and see if it looks feasible to advertise your product.

Complete Packages

You might even consider selling a complete package. Of course, that takes it out of the arena of a struggling student with no phone. But, as we discussed earlier, this might be a great enticement for a dealer to work with you.

Another advantage of a complete package is that you know exactly what customers have. You can tell them which disk to put into drive A (label up), what keys to press to bring up the menu (if it doesn't come up automatically), and which selections to make. The printer comes with all the cables, the system knows how to talk to the printer, and so on.

(Lewis) "I've never heard of people selling a system to go along with their software."

People are buying computers not for the sake of buying computers but because they want to do something. Sure, you could sell your software by itself, but it wouldn't hurt to let them know you offer a complete system.

"All of a sudden I can see porting it over to the Commodore 64."

Or whatever. The Commodore if you want to be (and look) cheap. Or even cheaper, the Adam for $69. The system might include enough software so that they could also do word processing and spread sheets.

Differentiation

We haven't dealt with a very important aspect, though we've been thrashing around it. That is, what's really unique about your product? What do you have that will make the customer sit up and take notice?

If you can't explain the difference in under six words you have a problem.

"It has better graphics."

Great. Let people see your graphics. Tell them enough about your package so they are really convinced that it has better graphics.

"Could I distribute my software via bulletin board?"

No. Not unless you really limited access to the board. If the board were open, you wouldn't get paid for your software. Plus, only a small part of your audience would have the equipment or the sophistication to use a bulletin board.

Overhead

"I can see that I don't have to worry so much about the labor. It's the advertising and support that are going to kill me."

Yeah. Getting the word out. Letting people know how special it is, setting the price properly. All those are reasons why large software outfits spend $1,000,000 to introduce a new software product.

■ ■ ■

Solderless CAS/MUX Mod

I devised a solderless method of doing the CAS/MUX modification (Micro C, issue 24, page 35).

A 14-pin wirewrap socket is the only part required. I removed pin 5 of the wirewrap socket with diagonal cutters, then bent pins 3 and 4 with long-nosed pliers so they line up with holes 4 and 5 of the socket on the motherboard. (See Mortensen Figure 1.) All the pins of the socket were then shortened. I then tapered the pins with a file to make it easy to insert the socket into the main board. (See Mortensen Figure 2.)

To complete the job I removed U66 from its socket, inserted it into the modified wirewrap socket, and replaced the combination in the motherboard, being sure that pins 3 and 4 went into holes 4 and 5. (See Mortensen Figure 3.) The wirewrap socket of necessity stands above the original socket but its stiff pins make it quite solid.

Figure 1 - Cutting and Bending

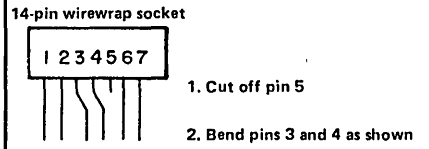


Figure 2 - Tapering the Ends of Pins

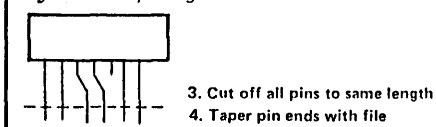
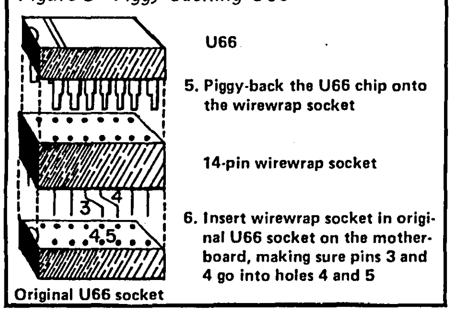


Figure 3 - Piggy-backing U66



I also used a wirewrap socket to take care of the U86 modifications for the speed-up. I bent up pins 4 and 5 on a 14-pin socket and soldered the leads from the speed switch to them. Then (as above) I shortened and filed all the pins, put U86 into the new socket, and put the assembly back into the original U86 socket.

The only soldering needed on the board was to the front end of R26 which I could do even with my big, clumsy hands. This method relieves one of the worry of frying a microchip with a soldering iron.

Joseph I. Mortensen
4214 Chelsea Ct.
Midland MI 48640

5MHz Without Surgery

Here's an adaptation of Trevor Marshall's excellent 5MHz modification for the BBI (Micro C, issue 11, page 4).

You'll need to buy three 14 header plugs (part# 14pinHP @ $.65 each) and three header covers (part# 14pinHC @ $.15 each) from JAMECO Electronics, 1355 Shoreway Rd., Belmont, CA 94002.

First, carefully solder a 3K 1/4 watt resistor between pins 3 and 7 of a header plug, placing the resistor snugly against the pins and in the center part of the plug. Next, solder a small length of wirewrap wire on one end of a 33 ohm resistor. Place this resistor beside the 3K resistor in the center and solder the other end to header plug pin 8. Now take a 74S04, bend pin 8 up, and solder all pins (except pin 8) to the header plug. You may now solder the wirewrap wire to pin 8 of the IC. If you've been careful with your soldering, the header cover will snap nicely on top of the assembly making a neat little module to insert in place of U77 (74LS04).

Remove U96 and bend pin 4 up. Bend pin 5 out just slightly. Solder all pins of U96 (except pins 4 and 5) to the corresponding pins on a second header plug. Now bend pin 5 slightly on the left so it rests on header plug pin 4 and solder it there. You may now snap the cover and put this module in socket U96.

Remove U76 (74164) and bend pin 5 completely upwards. Bend pins 3 and 4 slightly out. Solder all pins of the 74164 (except 3,4, and 5) to the corresponding pins on the third header plug. Now bend pin 4 slightly to the right so that it rests on header plug pin 5. The same with pin 3 so that it rests on top of header plug pin 4 and solder both in place. Snap the cover and plug module in socket U76.

All that's left to do is change transistor Q2 to an MPS 3640 inserting a 33 ohm resistor in series with the collector lead.

This modification has been in operation for several hundred hours in my system with 100 percent reliability. I use B types for all Z80 chips and memory is 200 ns. 4116s.

Christian Phaneuf
972 Guillaume Boisset
Cap-Rouge G1Y 3E4 P.Q. Canada

Xerox Formatter Fix

If you have a Xerox 820-I, you may have experienced the same problem I did with the format programs from user's disk #1. They didn't work! To make matters worse, a pre-formatted disk would shortly become un-pre-formatted. But since the user's disk also includes the formatter source code, a fix is a relatively simple matter. Just insert a short loop after the write track procedure that will test the 1771's busy status bit until it is ready for the next seek command. Xerox uses the same routine in its system monitor for all disk reads and writes.

```
NXTBYT: HALT
        OUTI
        JR      NZ,NXTBYT
        DEC     D
        JR      NZ,NXTBYT
________________________________
                                ;Insert Loop
                                ; here
BUSY:   IN      A,(WDSTAT)
        BIT     0,A
        JR      NZ,BUSY
________________________________
; PRINT TRACT # AND .......
```

Now you just reassemble and go, right? Well, maybe. If you are using the Crowe assembler from user's disk #1 you still have a little work to do. Although Croweasm has been fairly well discussed in several early issues of Micro C, here's what it needs in order to assemble the formatter programs:

1. The long ASCII strings near the end of the program must be broken down into multiple strings no greater than 32 characters and redefined using the pseudo-op DEFM instead of DEFB. Sometimes you might just want to shorten the message, but in some cases you'll need to define a long display (such as the track header and the track number display) with two or three sequential DEFMs.

2. The multi-byte definitions used in the programs must also be broken down. DEFB CR,LF,LF has to be divided into three separate statements for Croweasm.

3. Croweasm has a problem with relative addressing when it encounters decimal numbers greater than 2559 as it does in this formatter program. But Croweasm doesn't mind hexadecimal numbers, so that's how we'll get around it.

```
Old:        LD  SECT1+4836
________________________
Change to: LD SECT1+12E4H
```

Now you can reassemble and go.

Les Garrenton
3305 Scott St.
Portsmouth VA 23707

Some Like It Hot

My Kaypro 4 (1983 version with sockets for all the ICs) refused to boot properly for the first minute after turn on. It would give the A> prompt but would then respond to DIR with a question mark.

Replacing the 1793 and 9216 did practically no good. Replacing the A drive did no good, either.

I noticed that the time wait for correct action was worse with the cover off, and the only source of heat was the CRT. With the unit warmed up and operating correctly, I sprayed the eight 4164 250 ns. RAM chips with freeze spray and did a DIR. The problem was back. Then I warmed the RAM chips with a lamp. The problem was gone!

Replacing the RAM chips with 200 ns. devices has resulted in perfect operation.

John C. Reis
Box 874
Bensenville IL 60106

(continued from page 1)

Finally I remembered the outfit selling replacement IBM compatible keyboards at the West Coast Computer Faire. I had bought a replacement for my K16 keyboard from them. It was cheap, it was Taiwanese, and it worked.

Sandy dug up the receipt (Sky High Company), and I called them. Sure enough, they had Taiwanese everything. Did I want to buy an assembled system?

No, I wanted boards, cabinet, power supply, keyboard, monitor. Pieces. So they sent me their literature. When I saw the prices I knew I had found the source. See the "Dirt Cheap Clone Anyone Can Build" article for details.

Why All The Fuss?

So why all the excitement now? When the Kaypro 16 arrived (our very first clone) we were all curious about it. It introduced us to the peculiarities of MS-DOS (it's not very different from CP/M, so moving back and forth is easy). But the 16 was too expensive, too inscrutable, too much like a black box. (I'm not going to mess with anything that expensive and that poorly documented.)

You take it out of the box, turn it on, and hope it runs. If it doesn't you send it back.

That's no fun. If you open it up, you probably won't know which boards do what, and you can't operate the K16 with the boards exposed.

Anyway, it's a lot easier to futz with a system if zapping a board won't cost you the better part of a grand. (IBM wants $720 for an XT processor board.)

PC Resistance

I resisted moving into the PC environment for several reasons. First, PC software was more expensive than the equivalent CP/M software. Second, the clones offered no speed advantage over a 4MHz Z80 (sometimes, depending on the software, they were a disadvantage). Third, I resented IBM's loud proclamations that their "16-bit" 8088 was much more powerful than the "outdated" 8-bit machines. Well, the 8088 looks a lot more like a Z80 with bank select than a 68000. Fourth, I felt that Intel's convoluted segmented addressing scheme was a pain I didn't need.

There had to be some very good reasons before I would get excited about the clonal market. I finally found some.

First, clones are finally cheap, about half the price of the cheapest Kaypro Z80 system if you already have a couple of drives.

Second, you can put a clone together, futz with it, and add additional boards (there are lots available), all by yourself.

Third, the software is coming way down in price and going way up in function. The compilers are getting better and some folks are even writing in assembly language.

Fourth, unlike bank switching, you can really use the additional memory. A Pascal program that requires five or six overlays in CP/M will compile straight (and run faster) under MS-DOS.

Fifth, MS-DOS has some advantages over CP/M such as: named directories, redirection, built-in port drivers, and room in memory to add new features. The Z system (ZCPR3) has some of these features but it's larger than CP/M, and that is a definite penalty when you're confined to 64K.

Cheap Winchesters

The size of the PC environment and the surplus of imported hardware have cut sharply into the cost of winchestering a PC. For instance, you'll pay $1100 to $1200 for a 20 meg hard disk package for your Kaypro, but for about $1300 you can put together a complete XT with two floppies, 20 meg winchester, 640K, color and B/W graphics — the works.

DSI-32

Another reason that I started the search for a really cheap (and easy) clone is that the biggest cost of running a DSI board is the system it runs on (if you don't have one already). So, if you can put together a clone for $600, borrow the drives out of your Kaypro for $0, and then add a DSI board for $1000, you'd have a 6MHz 32032 screamer (just change the crystal for 8MHz) for the power-hungry stuff for $1600. Plus you'd still have a reasonable machine for the mundane stuff like editing (I do it all the time).

The Key To PC Comfort

If you are thinking about building up a cheap PC, but aren't sure you can handle the transition to another operating system, here are a couple of tips.

The transition will be easiest if you get MS-DOS versions of the software you are already using. For instance, if you do a lot of word processing and are comfortable with WordStar, then get an MS-DOS copy of WordStar (or one of the cheaper look-alikes). If you are working in Z80 dBASE II, then get MS-DOS dBASE II (or III). You'll be right at home because they kept the bugs and the same slow speed.

The Manual Gets Easier

I felt pretty spastic when I first started using the operating system. I couldn't move about among the named directories very dependably. (The manual seemed particularly inscrutible at first, but now that I don't need it, it seems much clearer.) The exact syntax of the commands is just different enough to keep an experienced CP/Mer off balance for a while. You'll get up to speed fastest if you have an experienced MS-DOSer peeking over your shoulder for an hour or two.

However, as an experienced CP/M user, you have some decided advantages over computer illiterates. There are so many strong similarities between the two systems that once you're over a few hurdles (e.g., use COPY SOURCE.FIL DESTINAT.FIL instead of PIP DESTINAT.FIL = SOURCE.FIL), you're on your way.

Voting With Your Wallet

A week ago I was speaking at the Eugene Oregon Kaypro group meeting. There were 12 attendees (down from 30+ two years earlier). The talk was pretty much a freeform question and answer session.

At the beginning we talked about 83 Kaypros vs. the 84 series. This kind of discussion is home ground for me. Then the discussion turned to the MS-DOS world. I talked about the way that CP/M had become the standard operating system for the 8080/Z80 world (rather than such proprietary notables as Heath-DOS or TRS-DOS), and the advantages of having a standard.

Then I mentioned that the PC had become another standard (with MS-DOS riding along on its coattails). Actually, it's an even better standard than CP/M because it defines an upward path (more memory, faster processors, older whiskey...), it supports graphics (all the way to 1024 by 1024), it supports sound (not fancy, but sound), and it has a larger installed base.

We thought we could survive quite a while in the Z80 arena and then leapfrog most of the 8088/86 PC environment. After all, there are bunches of PC rags out there and very few (approaching 0) publications dedicated to CP/M.

But CP/M is really going away, fast. I'm beginning to worry about where people are going to hang their coats as closets fill with Kaypros and Morrows. Even Ciarcia's new board hasn't lit much of a fire under CP/M.

Ciarcia's 64180 Board

Hooray, I thought. Faster than a speeding Z80, leaps half a megabyte in a single JMP, the 64180 deserves a red S and a private phone booth. After all, when you have a fast processor which can directly access 512K of RAM you can run large programs, three or four memory resident helpers (like Sidekick), and a zingy operating operating system, simultaneously (and schedule Lois Lane, too).

I was disappointed when Byte hit the streets. Ciarcia provided no support for a larger program area (I knew it wouldn't be easy, but I was still hoping). There was no support for multi-tasking, and the rest of the on-

(continued next page)

(continued from page 87)

board 256K was restricted to RAM disk service.

It would be difficult to tell the 6MHz 64180 from a 5MHz Kaypro in terms of system operation (in fact, for very large assemblies or compilations, the Kaypro with a 1 meg RAM disk might well outperform the 64180).

Plus, you'd have to work very hard to build up a 256K 64180 system for less than you could put together a 640K XT clone. (And what about serial port drivers, winchester drivers, graphics software?) If you want to build up a CP/M system and write your own drivers for it, then do it. It's a heck of an education. However, don't expect to have the most cost effective system around.

Back To The DSI-32 Board

Speaking of "cost effective," munch on the following: Yesterday I was sitting at the Micro C booth at the Oregon State University computer expo. Two booths down, Stride Micro was showing its $12,000 68000 based super system. No, it didn't have color graphics, no it didn't run PC software, but it did crunch numbers very well.

At our booth was a system that had a 20 meg winchester, ran all PC software, and because of the 10MHz DSI-32 board, it out crunched the Stride by a factor that wasn't funny (at least not to Stride). Our complete box cost only $3,000 including the $1,500 DSI-32.

In fact, a number of researchers at the school stopped by our booth to peek at the DSI-32. One had an economic model of U.S. agriculture (when weather squalls, then corn prices pop and hog prices squeal) written in FORTRAN (of course). The program took 80 minutes of CPU time to run on a Cyber (they ran it between 2 a.m. and 6 a.m. because that was the only time they could afford).

He figured that a 2 megabyte version of the 32032 board and the Green Hills FORTRAN compiler running on a cheap clone could not only handle the project but would cost less than what he'd budgeted for a year's Cyber time.

It's been five years since I used a Cyber (no, Micro C has never fired up its Cyber, what with electricity costs and all). If memory serves me correctly, it is extremely powerful (a 64-bit wide processor), extremely unfriendly (the Cyber's text editor almost made me give up writing), and extremely impressive. You can tell just by looking at a Cyber that the numbers it spits out are going to be as correct as numbers can get.

The fact that the obviously intelligent person standing in front of me was considering using the homely little clone on the table in place of a Cyber was pretty impressive. Pretty impressive.

Turn In Your Friendly Neighborhood Dealer

If you know of a bookstore, parts house, or computer store that should be carrying Micro C, send us their name and address, and we'll send them a copy and some information.

If they sign up and you were the first person to turn them in, then we'll send you a certificate good for a free public domain disk. If you send in two winners you get two certificates. (If you show them Micro C they'll probably tell you if they are interested.)

So far, all but two stores that have carried Micro C have sold out of every issue (we've had a total of five returned), and some are now ordering and displaying back issues.

The shops which should be most successful carrying Micro C are the ones with a good selection of really technical computer books (compiler design, hardware design...).

Send names and addresses to:

Turn In A Dealer Department
Micro Cornucopia
PO Box 223
Bend OR 97709

In This Issue

How to build a really cheap XT or AT is the feature article this issue. After several other magazines ran similar articles for high priced versions, I figured they could be beat for cheap (after all, that's the main reason we've been building all along).

We're including a short overview at the beginning of each article and column, these overviews should give you a good feeling for what the pieces cover and why we selected them for publication.

Coming Up In Micro C

The Turbo Pascal contest deadline has come and gone, and we'll be making February "National Turbo Month." We received more than 100 entries, much of them containing really good stuff! All the Pascal you ever thought possible and more will be in this special February Turbo issue (great for those cold winter days).

April (no foolin' — we're having another April issue) will be our 68000 issue. This means a close look at Amiga (its lack of appearance hasn't stopped anyone else from pretending it's real), plus an answer to the burning question: Do real programmers prefer doing it on the 68000? (If you want a clue to the answer, check out the instruction set on the 8086/8088.)

If you have any ideas for articles on Turbo, 68000, Amiga (or ST), XTs, ATs, an 8088 instruction set primer, 32000, you name it, then let us hear from you. If it's new and you're excited about it, then let us know.

Back Issues, Half Price

Over the first four and a half years we've kept all the issues of Micro C in print. But now, as we make our move to new systems and processors we're going to clear those back issues out of the basement. (This is going to shake up some very settled spiders.)

If you order six copies or more you pay only $1.50 each if you're a U.S. resident, $2 each if you live anywhere outside the U.S. (surface shipping). If you've been working with a partial deck and have considered filling out your collection of Micro Cs, this is definitely the time to do it. We currently have copies of all our back issues, but once they're gone, they're gone.

■ ■ ■

MAKE YOUR KAYPRO SPECIAL!

The Kaypro Disk Library

37 Disks of Selected Public Domain Software

KAYPRO DISK K1
Modem software
KAYPRO DISK K2
Utilities
KAYPRO DISK K3
Games
KAYPRO DISK K4
Adventure
KAYPRO DISK K5
MX-80 Graphics
KAYPRO DISK K6
Word Processing Utilities
KAYPRO DISK K7
Small C Version 2 Compiler
KAYPRO DISK K8
Small C Version 2 Source
KAYPRO DISK K9
ZCPR
KAYPRO DISK K10
Assemblers
KAYPRO DISK K11
Library & Checkbook Programs
KAYPRO DISK K12
FORTH
KAYPRO DISK K13
Source of fig-FORTH
KAYPRO DISK K14
Smartmodem Program
KAYPRO DISK K15
Hard Disk Utilities
KAYPRO DISK K16
Pascal Compiler
KAYPRO DISK K17
Z80 Tools
KAYPRO DISK K18
System Diagnosis
KAYPRO DISK K19
Prowriter Graphics
KAYPRO DISK K20
Color Graphics Routines
KAYPRO DISK K21
SBASIC Routines & Screen Dump
KAYPRO DISK K22
ZCPR (Again)
KAYPRO DISK K23
Fast Terminal Software & New BYE
KAYPRO DISK K24
MBASIC Games & Keyboard Translater
KAYPRO DISK K25
Z80 Macro Assembler
KAYPRO DISK K26
EPROM Programmer & Character Editor
KAYPRO DISK K27
Typing Tutor
KAYPRO DISK K28
Modem 730
KAYPRO DISK K29
Turbo Pascal Games 1 With Source
KAYPRO DKSK K30
Turbo Pascal Games II With Source
KAYPRO DISK K31
Turbo Bulletin Board
KAYPRO DISK K32
Forth-83 Much Fancier Forth
KAYPRO DISK K33
A super utilities disk
KAYPRO DISK K34
Five games plus source (mostly Turbo)
KAYPRO DISK K35
Small C Compiler & Source - Vers 2.1
KAYPRO DISK K36
Small C Library of 100 functions

Disks . . . . $12.00 each

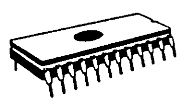
ROMS

Pro-8 Version 3 for Kaypro 4-83

Features include:
1. Screen dump with selectable dump character.
2. Select slow or fast step rate for each drive.
3. Automatically figures out what type of drive you are using.

Plus: User selectable cursor (blinking or not), ignores nulls, and your choice of 1-4 drives of the 191K, 390K, and 784K variety. (Use of 3 or 4 drives requires decoder.)

Installation requires no cuts or jumpers. The ROM simply plugs into a Kaypro 4-83 (or II-83 with a Kaypro 4 processor board). If you have a Kaypro II with the original II processor board then you must do the II to 4 upgrade. See issue #21 for details. Your Kaypro II has the original II board if the monitor ROM (a 20-pin chip with paper stuck to its top) is marked 81-149. The 4 ROM is marked 81-232.



for 84 KayPros

Pro-884

This ROM lets you run any mix of quad density (784K) drives and double-sided double density drives as A: and B: on you 84 Kaypro. Plus, if you plug-in the decoder board, you can run up to four drives.

Pro-884 MAX

The MAX gives you all the advantages of the Pro-884, plus faster screen scrolling, ZCPR in ROM (you 'warm boot' on any disk, whether it has system tracks or not), screen dump, ctrl-p type printer output, and VT52H/19 terminal emulation in addition to the original ADM3, to name a few. (We worked on this hummer for nearly a year.) If you need more information on the Pro-884s or our '83 compatible ROMS, call, write, or yell for our KayPro Users Catalog

Pro-Monitor ROMS

Pro-8 Ver 3 . . . . $49.95
Pro-884 . . . . $59.95
Pro-884 MAX $79.95
Decoder Board $39.95

Kaypro Add-Ons

Eight Inch Adaptor Board

OK, you asked for it: the 8 inch adapter board for the Kaypro. Now you can have: 1. 4 drives 2. Up to 1.3 meg of storage per disk. 3. Access to SIGM and CPMug software. 4. Super fast data access. The circuit board, ROM, and disk in this package will let your Kaypro run up to four drives. A and B have to be 5" drives, C and D can be either 8" or 5". The eight inch drives can read and write single density, double density, or double-sided double density. Plus, if you have an 84 2 or 4 you get all the Pro-884 MAX features.

The 8" adapter board plugs right in (no cuts, jumpers, or soldering — unless you have a Kaypro II, then you have to do a II to 4 upgrade.) You supply the drives, cabinet, and power.

8" Adapter Board $190.00

Schematic Packages

Finally, a schematic of your processor board, logically laid out on a single 24" by 36" sheet, plus a very complete, illustrated, Theory of Operation that's keyed to the schematic. You'll get information that's available nowhere else.

KayPro Schematic Packages

Kaypro II & 4 (pre-84) $20
Kaypro 10 (pre-84) $20
Kaypro 84 series (II, 4 & 10) $20

SPECIAL PRO-884 NOTE:

(And for 8" Adaptor Board)

The Pro-884s are sensitive to the version of CP/M you are running.

1. Neither the Pro-884 nor the Pro-884 Max will run on CP/M 2.2U. However, if you can locate a CP/M 2.2F or 2.2G system disk (your dealer should have a copy) you should be able to run our 884 monitors. (Don't try to boot F or G before you change monitors.)

2. There are two distinct versions of CP/M 2.2G. Only the Pro-884 Max is sensitive to the version of 2.2G you have - it's the ZCPR in ROM that's the problem. (If you have CP/M 2.2F then you have a Normal CP/M.) So, before ordering the Max, boot up your original system disk and read the sign-on. If it's CP/M 2.2G then we need to know whether it is the high (normal) version or the low (minus) version.

To determine your G version (you'll become a G Whiz!):

A>DDT <cr>
L5 <cr>
(ddt's response)

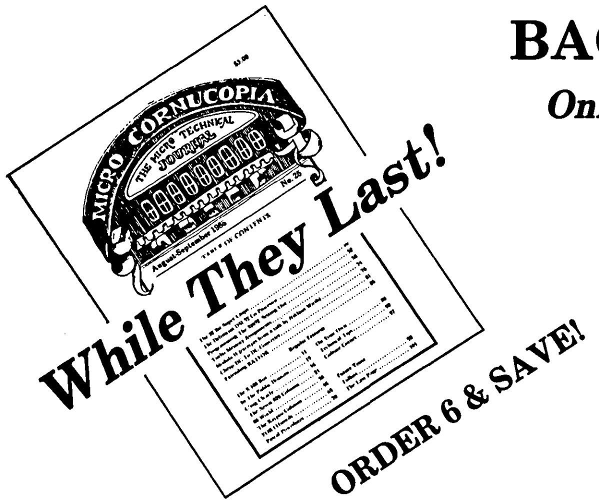
The first line of the response will be a JMP D600 or a JMP D800. The JMP D600 means that you have a low (minus) version, and the JMP D800 means that it's a normal version. When you order your Pro-884 Max, be sure to specify whether you want the normal Max or the minus Max. Otherwise, we'll just guess that you need the normal Max.

MICRO CORNUCOPIA
P.O. Box 223 • Bend, Oregon 97709
Order Number 9-5 PST (503) 382-5060 • Technical Information 9-Noon PST Only (503) 382-8048

WE'RE CLEARING THEM OUT!

BACK ISSUES OF MICRO C

Only $1.50 each when you order 6 or more
All Foreign $2.00 each for 6 or more
(includes surface postage)



ISSUE NO. 1 (8/81)
Power Supply
RAM Protection
Video Wiggle
1/2 PFM, PRN
16 pages

ISSUE NO. 2 (10/81)
Parallel Print Driver
Drive Motor Control
Shugart Jumpers
Program Storage Above PFM
1/2 PFM, PRN
16 pages

ISSUE NO. 3 (12/81)
4MHz Mods
Configuring Modem 7
Safer Formatter
Reverse Video Cursor
FORTHwords Begins
16 pages

ISSUE NO. 4 (2/82)
Keyboard Translation
More 4MHz Mods
Modems, Lync, and SIOs
Undoing CP/M ERASE
Keyboard Encoder
20 pages

ISSUE NO. 5 (4/82)
Word Processing
Two Great Spells
Two Text Editors
Double Density Review
Scribble, A Formatter
20 pages

ISSUE NO. 6 (6/82)
BBI EPROM Programmer
Customize Your Chars
Double Density Update
Self-Loading ROM
Terminal In FORTH
24 pages

ISSUE NO. 7 (8/82)
6 Reviews Of C
Adding 6K of RAM
Viewing 50 Hz
On Your Own Begins
24 pages

ISSUE NO. 8 (10/82)
Drive Maintenance
Interfacing Drives
Installing A New BIOS
Flippy Floppies
C'ing Clearly Begins
Xerox 820 Begins
28 pages

ISSUE NO. 9 (12/82)
BBII EPROM Program
Relocating Your CP/M
Serial Print Driver
Big Board I Fixes
Bringing Up WordStar
Cheap RAM Disk
32 pages

ISSUE NO. 10 (2/83)
Saving A Flakey Disk
Hooking Wini To BBII
The Disk Inspector
JRT Fix
Serial Keyboard Interface
Pascal Procedures Begins
36 pages

ISSUE NO. 11 (4/83)
BBI Expansions
BBII Details
Dyna, RAM Disk Review
Easier Reverse Video Cursor
PlannerCalc Review
Kaypro Column Begins
36 pages

ISSUE NO. 12 (6/83)
256K For BBI
Bringing Up BBII
dBASE II
Look At WordStar
Double Sided Drives For BBI
Packet Radio
5MHz For Kaypro
40 pages

ISSUE NO. 13 (8/83)
CP/M Disk Directory
More 256K For BBI
Mini Front Panel
Cheap Fast Modem
Nevada COBOL Review
BBI Printer Interface
Kaypro Reverse Video Mod
44 pages

ISSUE NO. 14 (10/83)
BBII Installation
The Perfect Terminal
Interface To Electronic Typewriter
BBI Video Size
Video Jitter Fix
Slicer Column Begins
Kaypro Color Graphics Review
48 pages

ISSUE NO. 15 (12/83)
Screen Dump Listing
Fixing Serial Ports
Playing Adventure
SBASIC Columns Begins
Upgrading Kaypro II To 4
Upgrading Kaypro 4 To 8
48 pages

ISSUE NO. 16 (12/84)
Xerox 820 Column Restarts
BBI Double Density
BBII 5"/8" Interface Fix
Kaypro ZCPR Patch
Adding Joystick To Color Graphics
Recovering Text From Memory
52 pages

ISSUE NO. 17 (4/84)
Voice Synthesizer
820 RAM Disk
Kaypro Morse Code Interface
68000-Based System Review
Inside CP/M 86
56 pages

ISSUE NO. 18 (6/84)
Kaypro EPROM Programmer
I/O Byte: A Primer
Kaypro Joystick
Serial To Parallel Interface
Business COBOL
60 pages

ISSUE NO. 19 (8/84)
Adding Winchester To BBII
6MHz On The BBI
Bulletin Boards
Track Buffering On Slicer
4MHz For The 820-I
64 pages

ISSUE NO. 20 (10/84)
HSC 68000 Co-Processor
DynaDisk For The BBII
Serial Printer On BBI Sans SIO
Cheap & Dirty Talker For Kaypro
Extended 8" Single Density
72 pages

ISSUE NO. 21 (12/84)
Analog To Digital Interface
Installing Turbo Pascal
Low Intensity BBI Video
Turbo Pascal, The Early Days
80 pages

ISSUE NO. 22 (2/85)
Xerox 820-II To A Kaypro-8
Sound Generator For The STD Bus
Reviews Of 256K RAM Expansion
In The Public Domain Begins
88 pages

ISSUE NO. 23 (4/85)
Automatic Disk Relogging
Interrupt Driven Serial Printer
Low Cost EPROM Eraser
Smart Video Controller
Review: MicroSphere RAM Disk
Future Tense Begins
88 pages

ISSUE NO. 24 (6/85)
C'ing Into Turbo Pascal
8" Drives On The Kaypro
48 Lines On A BBI
68000 Vs. 80X86
Soldering: The First Steps
88 pages

ISSUE NO. 25 (8/85)
Why I Wrote A Debugger
The 32-Bit Super Chips
Programming The 32032
Modula II
RS-232C: The Interface
104 pages

ISSUE NO. 26 (10/85)
Inside ZCPR3
Two Megabytes On DSI-32
SOG IV
The Future Of Computing
MS-DOS In The Public Domain
Graphics In Turbo Pascal
104 pages

BACK ISUES

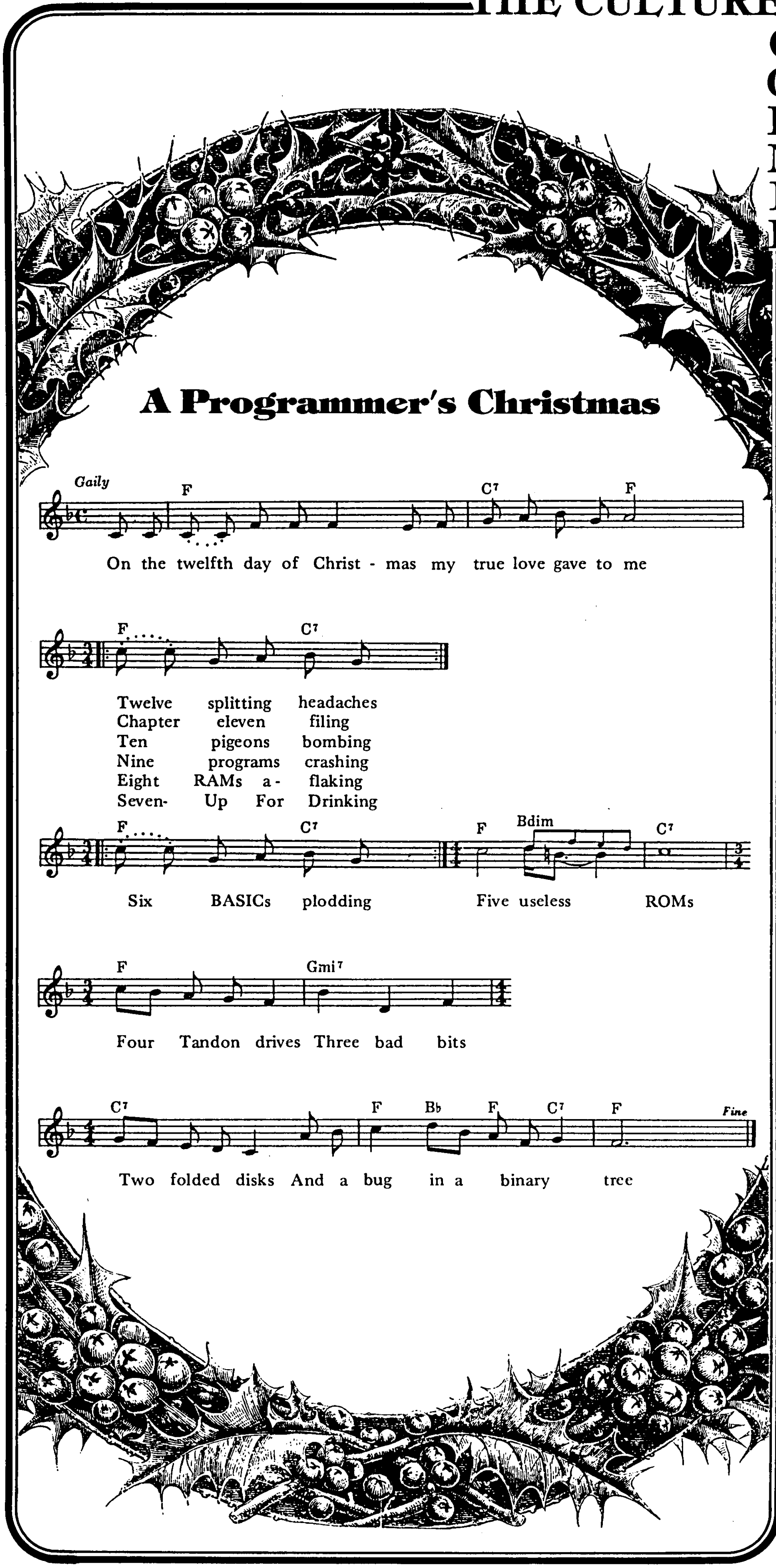
U.S. Regular Price .... $3.00 each
6 or more 1/2 price.......................... $1.50 each

Regular Foreign Price (air mail).............. $5.00 each
6 or more (including Canada) .... $2.00 each
Surface Postage Included

ASK FOR FREE CATALOG!

MICRO CORNUCOPIA

P.O. Box 223 • Bend, Oregon 97709
Order Number 9-5 PST (503) 382-5060 • Technical Information 9-Noon PST Only (503) 382-8048



A Programmer's Christmas
Gaily
F C7 F
On the twelfth day of Christ - mas my true love gave to me
F C7
Twelve splitting headaches
Chapter eleven filing
Ten pigeons bombing
Nine programs crashing
Eight RAMs a - flaking
Seven- Up For Drinking
F C7 F Bdim C7
Six BASICs plodding Five useless ROMs
F Gmi7
Four Tandon drives Three bad bits
C7 F Bb F C7 F
Two folded disks And a bug in a binary tree
Fine

WANT ADS

The following folks are reaching you for only 20 cents per word. If you would like to reach the same audience, send your words and 20 cents for each to Micro Cornucopia.

Disk Drive Sales & Service. For Sale — Shugart SA 800-2, $79; SA 850, $129; SA 850/851R, $139; MPI 52S 5.25" DSDD, $55; New TEC FB 501 5.25" SSDD, $59; Case w/ power supply for two half-hgt. drives, $59. SERVICE — SA 800/801, $25; SA 850/851, $35; 5.25" SS, $30; 5.25" DS, $40; 8" SS, $40; 8" DS, $50. All drives and service gruaranteed for 60 days. Prices do not include parts or shipping. LDL ELECTRONICS, 13392 158th St. N., Jupiter, FL 33458 (305) 747-7384. 28LY

All types of PROMs, EPROMs and PALs programmed. Very reasonable rates. Send $1.00 (refundable with purchase) for price list and instructions. Programming, 10663 Maple St., Cypress, CA 90630. 27VO

Hardware for the Kaypro Computer voice synthesizer unlimited speech $70. EPROM programmers 2516, 2716, 2532, 2732, 2764 $110. General purpose power supply used with EPROM programmers, $30. All hardware comes in kit or bareboard form with software and schematics. Call or write Busch Computer, 395 Totoket Rd., Northford, CT 06472, (203) 484-0320. 29BU

15" Green Monitors. New separated sync 15.7KHz for Xerox or 18.6KHz for IBM, $30. Cases extra. Also, new green CRTs 12", $15; 15", $20. 8" disk drives cheap!! Call evenings R. Breinlinger, Hampstead, NH (603) 329-5838. 27BR

For Sale — Billings Computer system, 4MHz, 64K, 2 DSDD 8" disk drives, Oasis O/S, COBOL, FORTRAN, BASIC, $650. TI 810 printer, $550. BBII bare board with extra parts, $120. Two FDD-100-8 disk drives, $95 each or both for $175. Will pay shipping on all above items. Duane Uhlenkott, 866 Olympic, Medford, OR 97504, (503) 773-5403. 27UH

For Sale: CompuPro CPU-Z 6MHz, $125. Disk 1A and CP/M-80, $475. 64K static RAM, $97. ALL equipment is in new condition. James Walker, 1205 N. Main, O'Fallon, MO 63366. 27WA

Public Domain UG Software Rental: CP/M UG Vol 1-92 on 46 8' flippies, $45, SIG/M UG Vol 1-216 on 108 8' flippies, $125. PICONET Vol 1-34 on 17 8' flippies, $25, Pascal-Z UG Vol 1-25 13 8' flippies, $25, KUG (Charlottesville) 39 disks, $25, IBM-PC SIG 1-300 PC-DOS, $325, PC-Blue 1-110 MS-DOS, $125, directory disks $5 PP. Also available on 209 5.25' formats. Rental is for 7 days after receipt with 3 more days grace for return. Credit cards accepted (preferred). Downloading-disk format conversions. Call. User Group Software Automatic Update Service, $7.50 per 2 volume set PP. 619-727-1015 24 hrs. 619-941-0925 info. 9-5. National Public Domain Software Center, 1533 Avohill, Vista, CA 92083.

TBKUG/Data COM Network supports CP/M-80, CP/M-86, Kaypro, S-100, IBM, and compatible users with public domain software via modem. We have three on-line databases holding over 40mby of CP/M and PC/MS-DOS programs available 24 hours at either 300 or 1200 baud. We specialize in CP/M disk utilities, ZCPR2 & 3, MEX, DOS utilities, Turbo Pascal, word processing, RCP/M utilities, BBS software, library utilities, catalog utilities, and much more. The Florida systems will soon be merging into one large multi-user system with many incoming lines. The TBKUG has 63 specific user disk volumes for Kaypro owners and over 5,000 files available by mail. We sponsor a monthly magazine that is distributed electronically by various national bulletin boards called the $R/O Read Only. It has many product reviews, public domain update information, Turbo Pascal column, and others of interest to CP/M and DOS users. The TBKUG has been in existence for over two years and has more than 600 members worldwide. We have been mentioned in numerous Profiles' articles and proudly support the CP/M users of the world with high quality software. An application may be downloaded via modem by calling (813) 937-3608, or send a SASE to: TBKUG/DataCOM Network, 14 Cypress Drive, Palm Harbor, FL 33563. Annual dues are $30.00.

Slicer Board For Sale. 8MHz Slicer system board. Includes MS-DOS operating system, BIOS/monitor disk, Slicer manual, disk drive cable, and two header to DB-25 cables for serial I/O. Works perfectly, need IBM compatible. $500 or best offer. Jim (505) 299-1255 (NM). 27DA

Keyboards for computer builders - 83 keys, full ASCII; upper/lower case, all control characters, numeric pad, caps-lock, repeat, self-test! Brand new, hundreds sold already to builders of Apples, Big Boards, Xerox 820s. Parallel output, positive TTL logic, strobe. Uses only 106mA of +5 volts. Custom case available. 90 day warranty unmodified. Keyboard $35. Documentation (21 pgs.)/cable package $5. Spare custom CPU/ROM $4. UPS included. Call/SASE for detailed spec sheet. Electrovalue Industrial Inc., Box 376-MC, Morris Plains, NJ 07950. (201) 267-1117.

20 Meg. CP/M hard disk computer TeleVideo 802H-20 for sale. CP/M 2.2, plus hardware specific backup, function key & keyboard buffer, programs. Other software including WordStar 3.3. Reduced to $1500. J.G. Eckmann, 16463 S.W. Boones Ferry Rd., Lake Oswego, OR 97034, (503) 636-8155 M-TH 9-4. 27EC

Mitsubishi M4853 5.25" half-height floppy drive. Double-sided quad density, 96tpi, $79 each. Seagate ST225 25 megabyte hard disk. New. $475. Limited quantities. B.W. Systems, P.O. Box 9791, Austin, TX 78766, (512) 255-8350. 27BA

Big Board II, keyboard, monitor, printer, software, 5 floppy drives, extras, $600. (503) 640-5367. 28MU

■ ■ ■

By Gary Entsminger

Future Tense Editor

Vedit Plus For MS-DOS & CP/M

CompuView has released two new Vedits — one for CP/M computers and one for PC compatibles. They're bigger (24K and 31K, respectively), a little flashier (although Vedit's never been known for flash), and just as easy to use. They're appropriately titled "Vedit Plus."

The full screen editing mode in Vedit Plus is identical to Vedit, plus includes a lot more registers for storing text or files (0-9 and A-Z), multiple file editing, a calculator for integer arithmetic, operating system commands from within Plus (DIR, DELETE), an optional menu (on the CP/M "Plus"), online help, a much improved installation program (on the MS-DOS version), and an extensive macro programming mode.

Several macros come ready to use on the CP/M version — a menu, a file comparison, and a mailing list sort — and the Vedit Plus manual details how you can write your own. (My MS-DOS copy is a Beta Test version and doesn't currently come with ready-to-run macros, but does support the full macro programming mode.)

You can edit files of unlimited size concurrently (very handy for reusing some of the text in a file, editing book chapters, or outlining).

I've been using Vedit (for a year) and Vedit Plus (for 2 months) on a Kaypro, a PC (personal clone), and an AT. I like it. It's a fast, dependable editor with thorough manuals. Costs $225.

For more info:

CompuView
1955 Pauline Blvd.
Ann Arbor MI 48103

Exploring Pascal By Disk, By Book

If you're just starting to learn Pascal or if you want to dig deeper, a book on disk, "Fast Track To Pascal," and a book, "Complete Turbo Pascal," will be helpful.

"Fast Track To Pascal" requires MS-DOS and has the advantage of on-screen graphics (but doesn't take full advantage of them). You can page backward and forward, jump to any subject area (procedures, arrays, records, graphics, etc.), run demonstration programs, and take quizzes. But be careful, you can't abort a quiz once you start it.

I found the prose a little stiff but tolerable (lighten up, you sound like a committee of educators!), but the idea's good, and you'll learn a lot by playing with the demonstration programs.

"Complete Turbo Pascal," on the other hand, is just a book, but a very good one, aimed specifically at teaching Pascal via a useful, available compiler — Borland's Turbo Pascal.

Author Jeff Duntemann (technical editor at PC Tech Journal) argues that standards are fine, but in this case, Standard Pascal can't do anything. "That includes frills like graphics along with nonfrills like disk I/O. The people who developed the ISO Standard definition for Pascal weighed language utility against portability and decided that portability was more important. I disagree strongly; the purpose of a computer is to get a job done, whatever it takes."

Examples are written for MS-DOS and CP/M, so no one gets slighted. The prose isn't light here either — programmers are such serious types — but it's clear and informative.

If you're learning Pascal with Turbo, it's a good tutorial, and for $10 extra you can get the example programs on disk.

If you've already gotten your feet wet, "The Complete Turbo Pascal" makes a good reference.

$19.95 buys the book from your local bookstore.

$10.00 buys the program disk from

Jeff Duntemann
805 Seaword Road
Towson MD 21204

For more info about "Fast Track To Pascal" (costs $39.95) contact —

Congenial Software
203 W. 16th St.
Lumberton NC
919-739-3733

DSI-32 Corner

Lots happening at Definicon (makers of the DSI-32 32032 plug-in board for the PC) this month.

Trevor Marshall, DSI-32 hardware designer, says they intend to release a Virtual Memory Unit and Unix System 5 by early December.

The Virtual Memory Unit will handle 16 megabytes. It requires some added hardware (a Memory Management Unit and possibly an Interrupt Control Unit) and new software (a systems upgrade). The additional memory will especially benefit FORTRAN programmers using the Green Hills compiler.

Green Hills FORTRAN currently requires 2 megabytes of memory because the compiler optimizes more highly than the Green Hills Pascal and C compilers and uses huge modules. Green Hills is working on a version of FORTRAN which will require less memory (both the Pascal and C compilers will run in 1 megabyte and in a limited way in 256K).

Unix System 5 will also require a hardware and software upgrade — an MMU, Interrupt Control Unit, and systems software. No price has been set yet for Unix System 5 or the Virtual Memory Unit.

Also, Definicon has upgraded their symbolic debugger — now it disassembles global symbols. Costs $95.

Miscellaneous DSI Users Notes — There's a free hardware fix for boards with serial numbers below 20. Stops the CPU from overheating. And (also free) a new loader with full MS-DOS parse and subdirectory capability.

For more info contact —

Definicon Systems, Inc.
21042 Vintage St.
Chatsworth CA 91311
818-341-5654

Inside The PC

If you already know more than a little about computers, and you want to read one book about the inner workings of the PC, then "The IBM Personal Computer From The Inside Out" by Sargent and Shoemaker is the one. I began reading it two weeks ago and still haven't finished; it's packed with information.

(continued next page)

(continued from page 95)

Starting from an overview/history of the PC, it covers beginning and advanced assembly language programming, digital circuitry — and such necessary subjects as handling I/O ports, a real-time clock interrupt scheme, data synchronization techniques, keyboard I/O, video display, controlling devices, monitoring devices, data communications, and building your own interfaces.

A useful example from one of the first chapters is a model layout for a simple assembly language program, setting up the segment registers, for the IBM Macro Assembler. See Figure 1.

To use this model for your own programs, replace the three lines of code beginning with "MOV DX,OFFSET MSG" with your own program, and replace the "MSGDB" with the variables and data you need.

"Inside Out" contains many good programming examples and numerous detailed drawings and explanations of PC hardware. In short, if you want to learn a lot about the PC, buy it (and read it patiently).

Have your bookstore order it from Addison-Wesley Publishing. It's $16.95.

TASM

If you're just starting to learn assembly language programming on the PC, purchasing EDITASM, a macro assembler from Speedware, will make learning a lot easier (I wouldn't want to learn without it).

EDITASM (or TASM) has a built-in editor (which is not easy to distinguish from Turbo Pascal's), and signs on with a similar-looking menu of options (shown in Figure 2).

From the options menu you can choose whether to display the assembly to screen or printer, assemble to memory, wait if error, create a .OBJ or .COM file, and more.

The wait if error option is great for initial debugging. The assembler stops at the error, and after you key in an "escape," it places you in the editor at your mistake. Sounds like Turbo, doesn't it?

TASM purports to be MASM-compatible. I can't yet verify the claim, but I've assembled short programs quickly and successfully (it's fun and speedy!). Bruce found one bug — TASM wasn't handling NESTED IF THEN ELSE STATEMENTS correctly — which Speedware promptly fixed. Uriah Barnett, vice president at Speedware, says they fix bugs as soon as they're reported. (Hooray for fixed bugs — keeps them from littering your system with buglets.)

TASM is an amazing product, essential for introductory programming, and assuming MASM compatibility, a major step forward for professionals. The built-in editor for interaction is the only way to fly.

TASM without .OBJ capability (won't let you assemble to .EXE files) costs $49.95. Full macro assembler costs $99.95.

For more info contact —
Speedware
118 Buck Circle
Folsom CA 95630
916-988-7426

Modula 2/86

Logitech's Modula 2 is out in an updated version (1.1) and remains microcomputer Modula state-of-the-art despite rumors of a Borland Modula.

Modula 2/86 is complete (albeit expensive and large) and easy to use. It features a native code compiler, an extensive library of standard modules, support for the 8087, support for REAL emulation, support for 1 megabyte of address space, access to underlying hardware and DOS functions, support for overlays, and a symbolic debugger.

Figure 1 - Minimal Assembly Program

```
DSEG    SEGMENT                     ;usually all variables go in the
MSG     DB  'hello world$'          ;data segment
DSEG    ENDS

SSEG    SEGMENT STACK               ;DOS will automatically set up
        DW      80 DUP (?)          ;a stack in the stack segment
SSEG    ENDS

CSEG    SEGMENT
        ASSUME CS:CSEG,DS:DSEG

MAIN    PROC    FAR
        PUSH    DS                  ;push start address of program
        SUB     AX,AX               ;prefix segment on the stack
        PUSH    AX
        MOV     AX,DSEG             ;then point DS AT DATA SEGMENT

        MOV     DX,OFFSET MSG       ;here's the main program body
        MOV     AH,09
        INT     21H                 ;it's only 3 lines

        RET                         ;a far return gets back to DOS
MAIN    ENDP
CSEG    ENDS

        END

((end listing))
```

Figure 2 - Menu of Options

```
Assem Source    Edit Source    Get Source   Write Source

Run Codefile    Hexdump File   Kill File    List File

Symbol List     Xrefer List    Directory    New Drive — Dir

Ass Options     Value          Quit
```

Figure from Last Page - CONFIG.SYS COMMANDS

```
BREAK        ;allows you to set BREAK=ON or OFF
BUFFERS      ;allows you to set # of buffers
COUNTRY      ;lets you set country for date/time format
DEVICE       ;lets you install device drivers
FCBS         ;lets you set # file control blocks that can be
              open concurrently
FILES        ;lets you specify the max # of files that can be
              open concurrently
LASTDRIVE    ;sets the maximum number of drives you can access
```

It requires PC compatibility with at least 256K RAM, 2 drives, and MS-DOS or CP/M 86 (sorry, CP/M 80 guys, you'll still have to wait for Borland).

Unlike Turbo Pascal, Modula 2/86 doesn't have a built-in editor, so editing, compiling, linking, and executing require separate steps. And Modula 2/86 doesn't automatically generate a .COM file. You need to execute your creation with a program called "m2" or use LOD2EXE, a utility which produces a simple .EXE file, that comes with version 1.1.

If you need the flexibility of Modula 2 and need to write large Pascal-like programs, Logitech's Modula 2/86 is the compiler of choice. It's dependable, standard, and the manual and support (at Logitech) are excellent.

But if you're writing smaller programs and have been spoiled by Turbo's built-in editor and non-standard functions, you might want to stay with Pascal a little longer. It's cheaper and easier to use. Modula 2/86 costs a whopping $495.

For more info contact —

Logitech
805 Veterans Blvd.
Redwood City CA 94063

FORTH Interest Group

FIG has moved, but you can reach them at —

P.O. Box 8231
San Jose CA 95155

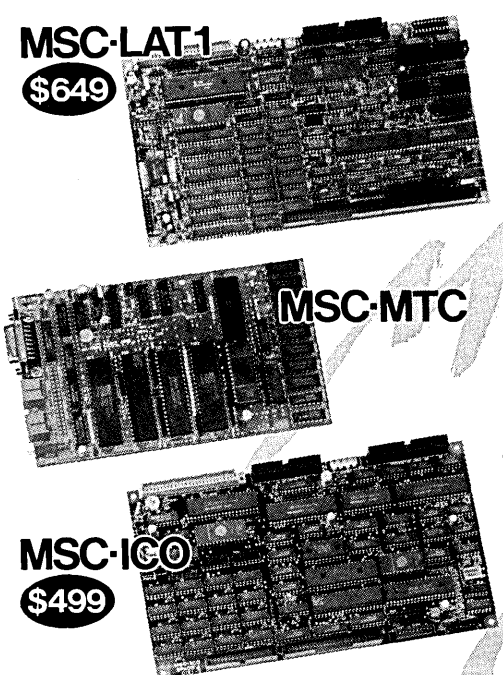
ICO And LAT Users' Group

If you're using Southern Pacific's single board computer (not the railroad), you might want to contact a users' group that's just organizing —

ICO-LAT-UG
C/O Andrew P. Porter
774 Joyce Street
Livermore CA 94550
415-443-4041

And That's Tidbits!

■ ■ ■



A Cube Root Function For Turbo Pascal

By Cheung S. Kwan

4514 Walnut St
Philadelphia PA 19139

This is a quick and easy programming project that really does something. In fact, these kinds of little library functions often take the drudgery out of major projects. The key to finding roots quickly is in calculating the next guess. Cheung does this quite elegantly.

I'm a math student and often need to compute the third, fourth, or higher roots of numbers. When I was programming in MBASIC, this was no problem since MBASIC's '^' operator allowed me to raise a number to any power. To find the cube root of x, just write "x^(1/3)" since raising a number to the (1/n)th power is the same as finding its nth root.

Turbo Pascal, my current language, has no such operator, and there is no easy way to find higher roots.

Editor's note: Here is an instance where the real world and mathematics are at odds. Give any guy a shovel and he'll swear that the higher roots are actually easier to find than the lower ones.

Anyway, I decided to put my math education to use and write a function to find the cube root of a number. This function can easily be modified to find the fifth, seventh, eleventh, and higher roots.

Finding An Algorithm

Let's say we want to find the cube root of 16. If we let x be the cube root of 16, then the problem becomes that of solving the equation x*x*x-16 = 0.

The algorithm might go like this —

1. Pick an initial guess for x.
2. Plug it into the equation x*x*x-16 (from now on referred to as f(x)) and see how close our answer is to 0.
3. If it's close enough to 0, then our guess is close enough to the true value of the cube root of 16 to suit our purposes. (For most numbers, we can never get the exact answer since the cube roots of most numbers are irrational, that is, their decimal parts go on forever, never repeating.)
4. If it's not close enough, then we find a better guess and plug that in and see if that result is close enough. If it is, we stop.
5. If not, we continue until we find a number that yields a result that's satisfactory.

Example

For the cube root of 16, a good initial guess would be a number that falls between the square root of 16 and the fourth root of 16 since we know that the cube root of 16 falls between these two numbers. In Turbo Pascal we can get the fourth root of 16 by writing sqrt(sqrt(16)). So let's use the number that lies midway between these two numbers, i.e., 0.5*(sqrt(16) + sqrt(sqrt(16))), as our initial guess.

This number turns out to be 3 and, plugging this back into our equation x*x*x-16, we get 11, not nearly close enough to 0. So now we look for a better guess.

Using Newton's Method, we can derive a better guess based on our previous guess. The mathematics of this is a little too complicated to go into here, but we can guarantee that if x0 is our previous guess, then x0-f(x0)/f'(x0) is a better guess than x0.

F(x0) in this case is x0*x0*x0-16 and f'(x0) is the first derivative of f(x) at x0 or, 3*x0*x0. Substituting 3 for x0, we get as our second guess 3-11/27, or 2 16/27.

*Editor's note: Derivatives are easy to do. You just reduce the power by one and use the old power as a multiplier and drop the constant. E.g., the derivative of Y*Y*Y*Y*Y-30 (Y to the fifth - 30) would be 5*Y*Y*Y*Y (5 times Y to the fourth).*

Then the process starts all over again. We plug 2 16/27 into f(x) and get the result 1.426205 which is a lot closer to 0 than 11 was but still not close enough. So we repeat the process until the result is close enough to 0 to suit our purposes.

In Sum

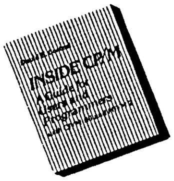
The above method is only one of several for finding the cube root of a number, but it's one of the fastest and simplest. And by making minor modifications to the code we can find just about any root of any number. Hence this function allows us, in a simple way, to make up for the lack of an '^' operator in Turbo Pascal.

■ ■ ■

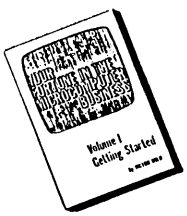
Turbo Pascal Cube Root Function

```
function cbrt (x:real):real;

  var a,b,c,d:real;

  function f(x,c:real):real;
    begin  f:=x*x*x-c  end;

  function df(x:real):real;  { This is the first derivative of f(x) }
    begin  df:=3*x*x  end;

  begin

    if x=0 then cbrt:=0 else
      begin
        c:=abs(x);
        b:=sqrt(c);
        if c>1 then d:=c*1E-11 else d:=1E-11;
                { This sets our error tolerance.  If c > 1, d = c*1e-11
                  instead of d = 1e-11 because if c is large enough, then
                  Turbo Pascal can not calculate an accuracy close enough
                  to 1e-11 to stop our loop below.  For example, if we want
                  the cbrt(1e21), Turbo Pascal will yield 10000000.000
                  (11 digit accuracy) which, when plugged back into our
                  function, cannot yield an accuracy approaching 1e-11. }

        a:=0.5*(b+sqrt(b));
        repeat
          b:=f(a,c);
          a:=a-b/df(a);
        until abs(b)<d;
        if x<0 then cbrt:=-a else cbrt:=a
      end

  end;
```

BOOKS, MAGAZINES, AND T-SHIRTS FROM MICRO C

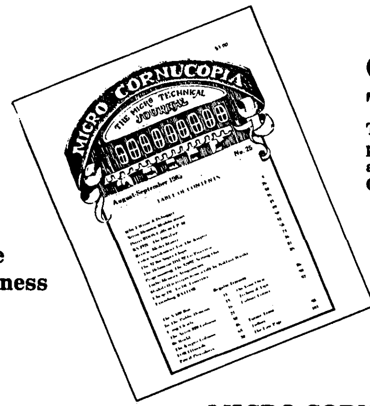


Inside CP/M
By David Cortesi
$27.95 US, Can, Mex
$37.95 (Other Foreign)
This is one of the best books on CP/M for everyone, users and programmers. It's our reference here at Micro C.

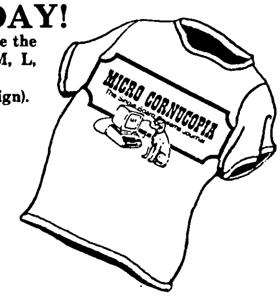


Your Fortune in the Microcomputer Business
By Victor Wild
$26.45 (US, Can, Mex)
$36.45 (Other Foreign)
This is the best, most complete collection of working for yourself information We've found.

The Small C Handbook
By James Hendrix
$17.95 (US, Can, Mex)
$22.95 (Other Foreign)
If you really want to solve the mysteries of the Small C compiler (or compilers in general) you'll want this book.



ORDER YOUR MICRO C T-SHIRT TODAY!
These two-color tan shirts are the perfect software. Sizes S, M, L, and XL.
Only $6.95 ppd. ($8.50 all foreign).



MICRO CORNUCOPIA MAGAZINE

Subscription Rates
1 yr US $16.00
2 yrs US $30.00
3 yrs US $42.00
1 yr Can & Mex $22.00
1 yr Other Foreign . . . $30.00

Inside information on the latest processors and languages.

MICRO CORNUCOPIA ORDER FORM

QUANT.	DESCRIPTION	PRICE EACH U.S.	CAN & MEX	OTHER	TOTAL
	SUBSCRIPTION (6 issues per year) ☐ New ☐ Renewal	1 yr. $16 ☐ 2 yr. $30 ☐ 3 yr. $42 ☐	☐ $22	☐ $30	
	USERS DISKS: *please specify #'s*	$12	$12	$12	
	KAYPRO SCHEMATIC PACKAGES *please specify #'s*	$20	$20	$20	
	BACK ISSUES: *please specify #'s*	$3	$3	$5	
	T-SHIRT *(specify S,M,L,XL)*	$6.95	$8.50	$8.50	
	FREE CATALOG				
	OTHER ITEMS:				
				TOTAL ENCLOSED	

NAME ____________
COMPANY ____________
ADDRESS ____________
____
CITY ____________
STATE ____________ ZIP ______
____

Are you a current subscriber to Micro Cornucopia? ☐ yes ☐ no

☐ Check or money order enclosed
(U.S. funds only drawn on a U.S. bank)
Prices include postage (Air Mail for Other foreign)

Make checks payable to:
MICRO CORNUCOPIA

What kind of computer do you own/operate? ____________

Card No. ______ - ______ - ______ - ______ Exp. ______ ____________
☐ Visa ☐ MasterCard Signature

MICRO CORNUCOPIA

P.O. Box 223 • Bend, Oregon 97709
Order Number 9-5 PST (503) 382-5060 • Technical Information 9-Noon PST Only (503) 382-8048

ASK FOR FREE CATALOG!

The Micro C MS-DOS Library

MS-1 Essential Utilities
SWEEP, LU86403, DSKRTN, LDIR103, WHEREIS, SQUEEZE, UNSQUEEZE, WASH

MS-2 Cheap Assembler & Disassembler
CHASM, ASMGEN, MEMBRAIN, UNWORDSTAR, FSPOOL, RAMDISK, SDIR, SDIR SOURCE

MS-3 Adventure
The original and most cussed adventure in C, with source.

MS-4 Single User RBBS
A decent bulletin board with source.

MS-5 File & Directory Utilities
UTIL, Z, VTYPE, ST, PROTECT, UNPROTECT

MS-6 fig FORTH
We combined fig FORTH source (MS-7) with this disk so you get both on one disk for $12.00.

MS-8 Super Games
CASTLE, a graphic and conversational adventure, TREKRUN Star Trek Command adventure, and BOUNCING BABY (a finger exercise).

MS-9 PC Musician, Pascal Games, Calculator
You might call this an educational disk. PC MUSICIAN teaches you music while you're writing songs. They play right on the PC. 15 PASCAL GAMES with source. The games aren't memorable but they are good examples of Turbo code. PORGRAMMER'S CALCULATOR shows the internal workings of a stack, memory, and registers.

MS-10 Modems
QMODEM and KERMIT

MS-11 Multi-User RBBS 12.3
Multi-user Remote Bulletin Board System with source disk (MS-12) included free.

MS-13 Editor & Formatter
ROFF4, PC-WRITE, and manuals on disk. Very popular freeware.

MS-14 FORTH 83

All disks are PC Standard DS/DD 360K — $12.00 each, ppd.

MICRO CORNUCOPIA

P.O. Box 223 • Bend, Oregon 97709

Order Number 9-5 PST (503) 382-5060 • Technical Information 9-Noon PST Only (503) 382-8048

ADVERTISER'S INDEX

The Last Page

By Gary Entsminger

Micro C Staff

If you know how to use BATCH (or SUBMIT) files, you can avoid redundancy and eschew obfuscation. (Let me repeat that...)

BAT And SUB

Both MS-DOS and CP/M allow you to execute programs from special files that have the extensions .BAT (in MS-DOS) or .SUB (in CP/M). You create these BATCH or SUBMIT files with your editor or (in MS-DOS) at the command line.

For example (in DOS), you can link an assembled source file (a .OBJ file) into a .EXE file and then use EXE2BIN to convert it from an .EXE to a .COM file by writing a batch file, EXAMPLE.BAT —

```
LINK ASMFILE
EXE2BIN ASMFILE
```

Then to execute EXAMPLE.BAT enter EXAMPLE at the DOS command line. MS-DOS has its batch (submit) function built in, so you run a .BAT file just as though it were a .COM or .EXE file.

Note: .EXE files are the default type and can be very large. .COM files are limited to 64K total but load and execute faster than .EXE files. So for smaller programs, it's an advantage to convert them into .COM files.

In both MS-DOS and CP/M you can add parameter substitution within the batch (submit) files. To substitute the first parameter on the command line into the file, you use %1 in MS-DOS and $1 in CP/M. (Curiously, under MS-DOS, $0 is the name of the .BAT file itself.)

Thus, if CPYBAK.BAT contained:

```
COPY %1.ASM %1.BAK
TYPE %0.BAT
```

and you entered on the MS-DOS command line:

```
CPYBAK ASMFILE
```

then MS-DOS would do the following:

```
COPY ASMFILE.ASM ASMFILE.BAK
TYPE CPYBAK.BAT
```

Of course, there are %2, %3, and $2, $3, and so on. Each parameter is the next entry on the command line when you call the .BAT or .SUB file.

AUTOEXEC.BAT And CONFIG.SYS

If you need to execute the same batch of files every time you boot, you can have DOS and CP/M do it automatically — with an AUTOEXEC.BAT. It's similar to other .BAT files, except the command processor searches for it at boot time. And if it finds it, executes it before giving you the green light, the A>.

For example, in MS-DOS I use AUTOEXEC.BAT and CONFIG.SYS to autoinstall SideKick, Uniform, and to partition off part of memory as a RAMdisk.

CONFIG.SYS contains the commands MS-DOS uses to configure your system at boot time. If you don't write your own CONFIG.SYS file, DOS will assign default values for the configuration commands. (See "Tidbits" for a list of CONFIG.SYS commands. DEVICE, FCBS, COUNTRY, and LASTDRIVE are new commands in 3.0.)

When you boot DOS, it looks for a CONFIG.SYS file just before it looks for an AUTOEXEC.BAT.

In 3.0 we can use DEVICE, one of the CONFIG.SYS commands, in a CONFIG.SYS file to add SideKick, Uniform, and a RAMdisk at bootup (who says man won't use all that RAM?) —

```
SK
DEVICE=UNIFORM.SYS AT+ DR=1
FC=1 IN-
DEVICE=MEMBRAIN.SYS
```

By using several disks and several different AUTOEXEC.BAT and CONFIG.SYS files you could vary your system from task to task, improving your efficiency while preserving your fingertips.

CRC And COMP

At Micro C we use a BATCH file, CRC1.BAT, to verify the programs on the public domain disks we distribute. First, we create a file called CRCKLIST.CRC and rename it to L.CRC.

```
CRC *.* f
RENAME CRCKLIST.CRC L.CRC
```

CRC creates a unique code (X^16 + X^15 + X^13 + X^7 + X^4 + X^2 + X +1) for each file on the disk and writes the code into a file named CRCKLIST.CRC. Unlike the CP/M public domain program, CRC, MS-DOS's public domain version doesn't compare the files on a disk with the CRCs in the CRCKLIST.CRC file.

But we can rename CRCKLIST.CRC to L.CRC so it won't be overwritten when CRC creates another CRCKLIST.CRC. Then you can compare the new CRC file with the one we created. We put CRC2.BAT on each disk to handle this. It contains:

```
CRC *.* f
COMP CRCKLIST.CRC=L.CRC
```

Summing, Not Slumming

This is by no means all there is to BATCH processing. But it should whet your appetite. With IF, GOTO, FOR, ECHO, PAUSE, and REM subcommands, BATCH is very powerful.

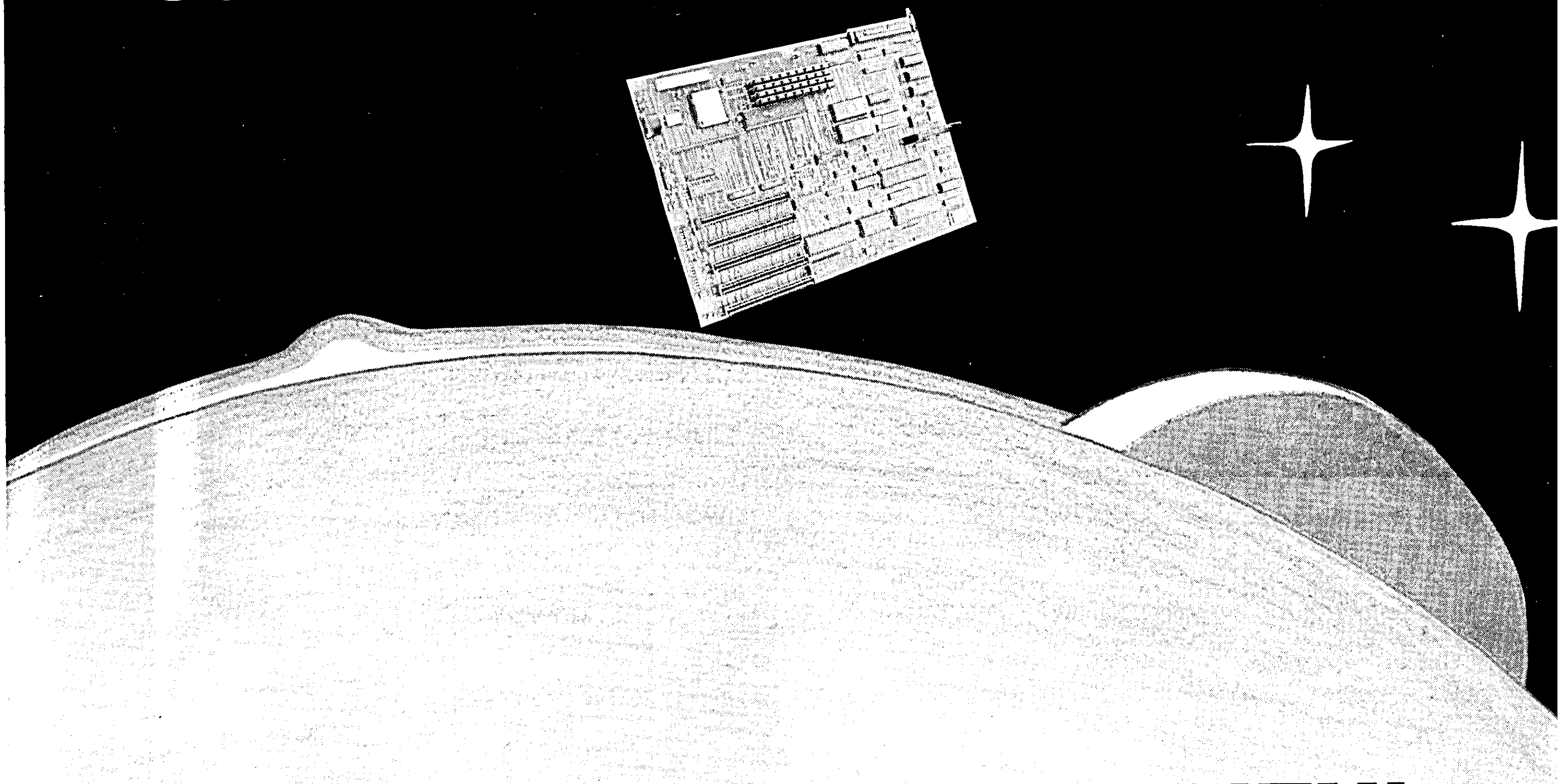
Nearly anything that you repeat over and over (e.g., compile/assemble/link) can be combined into a .BAT or .SUB file. Now if only we could run the following .BAT file:

```
FOR ARTICLES=GOOD EDIT
IF NO COLUMNS CALL COLUMN-
ISTS
GOTO TYPESETTING
REPEAT FOREVER
```

■ ■ ■

NEW LOWER $

the X-16 gives you Out of This World Performance



PLUS PC COMPATABILITY!

Features:

- 8 MHz 80186 microprocessor with true 16-bit data bus.
- True PC compatability with our own custom ROM BIOS and PC cardslots for the video of your choice.
- Standard 512K zero wait DRAM, 640K or 1M options. NEW
- 8087 numeric coprocessor port.
- On-board floppy disk controller for up to four 5¼" drives.
- On-board SASI hard disk controller port.

FOR MORE INFORMATION CONTACT:

PCTECH

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

COD, VISA, MASTER CARD, CERTIFIED FUNDS

Kits

NEW LOWER PRICES

- Not So Bare Board .... $ 250.00
- Quick Kit .... $ 625.00
- Full Kit .... $ 795.00
- Assembled and Tested .... $ 895.00
- 128K Add-On .... $ 90.00
- ½ MEG Add-On .... $ 240.00
- XT Style, Flip Top Enclosure ...... $ 85.00
- 150 Watt Power Supply .... $ 135.00
- 5¼" 48 TPI DS ½ Height Floppy Disk Drive .... $ 125.00

NEW

Total Solutions:

Assembled and Tested Systems

- Mono-Chrome System $1,865.00
  Includes: 640K, Power Supply, Enclosure, Keyboard, Monitor, Two Drives, and Printer Port.
- Color System .... Call
- Quantity Discounts Upon Request
- Call For Info on Add-Ons, Hard Disks, Etc.

MICRO CORNUCOPIA
The Micro Technical Journal

3
97709

FORWARDING & ADDRESS CORRECTION REQUESTED

BULK RATE
U.S. POSTAGE
PAID
Forest Grove, OR
Permit No. 36