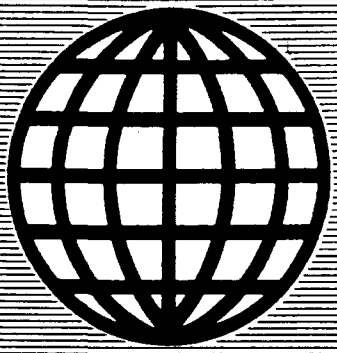


The COMPUTER JOURNAL®

Programming - User Support

Applications

Issue Number 30

$3.00

Double Density Floppy Disk Controller

An Algorithm for the CP/M Operating System

ZCPR3

Implementing IOP Support for the Ampro BIOS

32000 Hacker's Language

MDISK

Part 2: A One Megabyte RAM Disk for the Ampro L.B.

Non-Preemptive Multitasking

Software Timers for the 68000

Lillipute Z-Node

A Remote Access System for TCJ Subscribers

The CP/M Users' Corner

ISSN # 0748-9331

THE COMPUTER JOURNAL
190 Sullivan Crossroad
Columbia Falls, Montana
59912
406-257-9119

Editor/Publisher
Art Carlson

Art Director
Donna Carlson

Production Assistant
Judie Overbeek

Circulation
Donna Carlson

Contributing Editors
Joe Bartel
Bob Blum
C.Thomas Hilton
Bill Kibler
Frederick B. Maxwell
Jay Sage
Kenneth A. Taschner

***Subscription rates**—$16 one year (6 issues), or $28 two years (12 issues) in the U.S., $22 one year in Canada and Mexico, and $24 (surface) for one year in other countries. All funds must be in US dollars on a US bank.*

Send subscriptions, renewals, or address changes to: The Computer Journal, 190 Sullivan Crossroad, Columbia Falls, Montana, 59912, or The Computer Journal, PO Box 1697, Kalispell, MT 59903.

Address all editorial and advertising inquiries to: The Computer Journal, 190 Sullivan Crossroad, Columbia Falls, MT 59912 phone (406) 257-9119.

The COMPUTER JOURNAL

Features

Issue Number 30

Columns

Z sets you free!

Who we are

Echelon is a unique company, oriented exclusively toward your CP/M-compatible computer. Echelon offers top quality software at extremely low prices; customers are overwhelmed at the amount of software they recieve when buying our products. For example, the Z-Com product comes with approximately 92 utility programs; and our TERM III communications package runs to a full megabyte of files. This is real value for your software dollar.

ZCPR 3.3

Echelon is famous for our operating systems products. ZCPR3, our CP/M enhancement, was written by a software professional who wanted to add features normally found in minicomputer and mainframe operating systems to his home computer. He succeeded wonderfully, and ZCPR3 has become the environment of choice for "power" CP/M-compatible users. Add the fine-tuning and enhancements of the now-available ZCPR 3.3 to the original ZCPR 3.0, and the result is truly flexible modern software technology, surpassing any disk operating system on the market today. Get our catalog for more information - there's four pages of discussion regarding ZCPR3, explaining the benefits available to you by using it.

Z-System

Z-System is Echelon's complete disk operating system, which includes ZCPR3 and ZRDOS. It is a complete 100% compatible replacement for CP/M 2.2. ZRDOS adds even more utility programs, and has the nice feature of no need to warm boot (^C) after changing a disk. Hard disk users can take advantage of ZRDOS "archive" status file handling to make incremental backup fast and easy. Because ZRDOS is written to take full advantage of the Z80, it executes faster than ordinary CP/M and can improve your system's performance by up to 10%.

Installing ZCPR3/Z-System

Echelon offers ZCPR3/Z-System in many different forms. For $49 you get the complete source code to ZCPR3 and the installation files. However, this takes some experience with assembly language programming to get running, as you must perform the installation yourself.

For users who are not qualified in assembly language programming, Echelon offers our "auto-install" products. Z-Com is our 100% complete Z-System which even a monkey can install, because it installs itself. We offer a money-back guarantee if it doesn't install properly on your system. Z-Com includes many interesting utility programs, like UNERASE, MENU, VFILER, and much more.

Echelon also offers "bootable" disks for some CP/M computers, which require absolutely no installation, and are capable of reconfiguration to change ZCPR3's memory requirements. Bootable disks are available for Kaypro Z80 and Morrow MD3 computers.

Z80 Turbo Modula-2

We are proud to offer the finest high-level language programming environment available for CP/M-compatible machines. Our Turbo Modula-2 package was created by a famous language developer, and allows you to create your own programs using the latest technology in computer languages - Modula-2. This package includes full-screen editor, compiler, linker, menu shell, library manager, installation program, module library, the 552 page user's guide, and more. Everything needed to produce useful programs is included.

"Turbo Modula-2 is fast...[Sieve benchmark] runs almost three times as fast as the same program compiled by Turbo Pascal...Turbo Modula-2 is well documented...Turbo's librarian is excellent". - Micro Cornucopia #35

BGii (Backgrounder 2)

BGii adds a new dimension to your Z-System or CP/M 2.2 computer system by creating a "non-concurrent multitasking extension" to your operating system. This means that you can actually have two programs active in your machine, one or both "suspended", and one currently executing. You may then swap back and forth between tasks as you see fit. For example, you can suspend your telecommunications session with a remote computer to compose a message with your full-screen editor. Or suspend your spreadsheet to look up information in your database. This is very handy in an office environment, where constant interruption of your work is to be expected. It's a significant enhancement to Z-System and an enormous enhancement to CP/M.

BGii adds much more than this swap capability. There's a background print spooler, keyboard "macro key" generator, built-in calculator, screen dump, the capability of cutting and pasting text between programs, and a host of other features.

For best results, we recommend BGii be used only on systems with hard disk or RAMdisk.

JetFind

A string search utility is indispensible for people who have built up a large collection of documents. Think of how difficult it could be to find the document to "Mr. Smith" in your collection of 500 files. Unless you have a string search utility, the only option is to examine them manually, one by one.

JetFind is a powerful string search utility which works under any CP/M-compatible operating system. It can search for strings in text files of all sorts - straight ASCII, WordStar, library (.LBR) file members, "squeezed" files, and "crunched" files. JetFind is very smart and very fast, faster than any other string searcher on the market or in the public domain (we know, we tested them).

Software Update Service

We were suprised when sales of our Software Update Service (SUS) subscriptions far exceeded expectations. SUS is intended for our customers who don't have easy access to our Z-Node network of remote access systems. At least nine times per year, we mail a disk of software collected from Z-Node Central to you. This covers non-proprietary programs and files discussed in our Z-NEWS newsletter. You can subscribe for one year, six months, or purchase individual SUS disks.

There's More

We couldn't fit all Echelon has to offer on a single page (you can see how small this typeface is already!). We haven't begun to talk about the many additional software packages and publications we offer. Send in the coupon below and just check the "Requesting Catalog" box for more information.

Item	Name	Price	
1	ZCPR3 Core Installation Package	$49.00	(3 disks)
2	ZCPR3 Utilities Package	$89.00	(10 disks)
5	Z-Com (Auto-Install Complete Z-System)	$119.00	(5 disks) *
6	Z-Com "Bare Minimum"	$69.95	(1 disks)
10	BGii Backgrounder 2	$75.00	(2 disks)
12	PUBLIC ZRDOS Plus (by itself)	$59.50	(1 disk)
13	Kaypro Z-System Bootable Disk	$89.95	(3 disks)
14	Morrow MD3 Z-System Bootable Disk	$69.95	(2 disks)
16	QUICK-TASK Realtime Executive	$249.00	(3 disks)
17	DateStamper file time/date stamping	$49.95	(1 disk)
18	Software Update Service	$85.00	(1 yr sub)
20	ZAS/ZLINK Macro Assembler and Linker	$69.00	(1 disk)
21	ZDM Debugger for 8080/Z80/ HD64180 CPU's	$50.00	(1 disk)
22	Translators for Assembler Source code	$51.00	(1 disk)
23	REVAS3/4 Disassembler	$90.00	(1 disk)
24	Special Items 20 through 23	$169.00	(4 disks)
25	DSD-80 Full Screen Debugger	$129.95	(1 disk)
27	The Libraries.SYSLIB, Z3LIB, and VLIB	$99.00	(8 disks)
28	Graphics and Windows Libraries	$49.00	(1 disk)
29	Special Items 27, 28, and 82	$149.00	(9 disks)
30	Z80 Turbo Modula-2 Language System	$89.95	(1 disk)
40	Input/Output Recorder IOP (I/OR)	$39.95	(1 disk)
41	Background Printer IOP (BPrinter)	$39.95	(1 disk)
44	NuKey Key Redefiner IOP	$39.95	(1 disk)
45	Special Items 40 through 44	$89.95	(3 disks)
60	DISCAT Disk cataloging system	$39.99	(1 disk)
61	TERM3 Communications System	$99.00	(6 disks)
64	Z-Msg Message Handling System	$99.00	(1 disk)
66	JetFind String Search Utility	$49.95	(1 disk)
81	ZCPR3: The Manual bound, 350 pages	$19.95	
82	ZCPR3: The Libraries 310 pages	$29.95	
83	Z-NEWS Newsletter, 1 yr subscription	$24.00	
84	ZCPR3 and IOPs 50 pages	$9.95	
85	ZRDOS Programmer's Manual 35 pages	$8.95	
88	Z-System User's Guide 80 page tutorial	$14.95	

* Includes ZCPR3: The Manual

Echelon, Inc.

885 N. San Antonio Road, Los Altos, CA 94022 USA
415/948-3820 (order line and tech support)
Telex 4931646

NAME _____

ADDRESS _____

_____

TELEPHONE _____ DISK FORMAT _____

☐ REQUESTING CATALOG

ORDER FORM

Payment to be made by:
- ☐ Cash
- ☐ Check
- ☐ Money Order
- ☐ UPS COD
- ☐ Mastercard/Visa:

_____

Exp. Date _____

California residents add 7% sales tax.
Add $4.00 shipping/handling in North America, actual cost elsewhere.

ITEM	PRICE
_____	_____
_____	_____
_____	_____
Subtotal	_____
Sales Tax	_____
Shipping/Handling	_____
Total	_____

Editor's Page

Industrial Electronics

As I have previously stated, I firmly believe that the employment and business opportunities of the future are in using microcomputers and embedded microprocessors in areas other than personal and business computers. Several people have told me that 90% of the CPUs and MPUs are going into commercial products other than computers. Another report states that they expect to sell 250 million microcontrollers this year, and 750 million by 1990.

I see a lot of help wanted ads for people experienced in industrial type applications, but few ads for people to work on personal or business computers. For example, Nyland Associates (Route 2 Box 352, Nebo, NC 28761 phone (704)652-1801), publishes *The Nayland Letter*, 'The employment newsletter for the C software engineering and programming community', and in a recent issue they state 'The greatest demand seems to be in the areas of microprocessor control systems, PC to mainframe connect technology, local area networking as well as other areas of computer technology. Those engineers expected to receive premium compensation will be working with real time control and monitoring systems, drivers, compiler systems, robotics, diagnostics, artificial intelligence, manufacturing, office automation systems, voice and data communication systems, firmware, graphics, desktop publishing, and superconductivity.' Contact Mark Tokay at Nayland for information on their services.

The automotive industry is going for electronics in a big way, and the Society of Automotive Engineers (SAE) is working on the development of a standardized automotive communications bus. One of the contenders is the Inte-Bosch Car Area Network, but the SAE Vehicular Networks for Multiplexing and Data Communications and the Truck and Bus Control and Communications Network subcommittees are both working on specifications — the markets opened by the use of high speed data buses in vehicles will increase the need for people with real time control and communications experience.

Some of the experience requirements in current help wanted ads are: "Communications Networking, Database Management, Factory Automation, Motion Control, Interfacing With Communications, Real Time Control of Factory Workstations, Emmbedded Processors, Real Time Distributed Control Network Using Digital Closed Loop Control Techniques, Signal Filtering, Prior Experience with 8051 Microprocessors, Real Time O/S Design and Implementation, Compiler Design, Microprocessor Based Industrial and Vehicular Instrumentation/Controls, ASSEMBLY Language Programming."

Several people have told me that they can't fill their job openings for people who can program in assembly language for real time embedded controllers, but that there are lots of graduates who know about spreadsheets, the IBM-PC, and a High Level Language (they have no job openings for these people).

An RBBS for TCJ Subscribers

After I announced the end of the TCJ BBS because of the problems with our long rural phone line, Richard Jacobson offered to open an area of his Remote Access System (Z-Node #15) to TCJ subscribers. His description of the system is in this issue, and we will be monitoring your use and messages to decide what files need to be placed in the TCJ section.

A New CP/M Author

Well, really not new to CP/M users, but rather, new to TCJ. Bob Blum, who formerly edited a CP/M column for two other magazines while they were supporting CP/M, has joined the TCJ crew. See his first introductory column in this issue, and be sure to respond with the questions and comments we need in order to tailor his column to your needs.

Being Friendly Ain't Easy

In issue #28 (Programming Style — User Interfacing and Interaction) I discussed the need for better user interfacing, and this made me aware of how much I have neglected this in my own programs. I use the excuse that these are simple programs written for our own internal use only, and that we know how they work, but I'll have to admit that I have problems using programs which I wrote a year ago. I keep a folder or notebook for each program with commented code and notes, but there is not enough commenting or notes on the user interface — and we shouldn't have to hunt for the notes when we want to use the program.

It is easy to say that software should facilitate use by the novice without getting in the way of the expert, but deciding exactly what steps need to be taken in order to accomplish this is more difficult than it first appears — even for short trivial programs. I'm spending much more time designing the interface than I am on the program itself, but once I have established the basic interface I'll be able to use it as a starting point for other programs.

What we develop will not be exactly what you want, and will not be suitable for widely distributed commercial programs. It is custom designed to fill our needs, and will be constantly revised as our needs change, and as we refine the interface structure. I'll publish what we are doing, along with some of our ideas, and discuss some of the trade-offs that should be considered, as a starting point for your comments and improvements.

The HELL With Being Compatible
I Want What I Want!

While I use commercial products for handling database and word processing, and the usual languages, assemblers, and compilers, almost everything else involves custom programs written for our internal use only. I don't sell these programs, and they are not intended for wide distribution, which gives me a lot of freedom, because my programs don't have to run on anyone else's system!

One of the things which I want to implement on my Z80 system is the use of UNIX® style pipes to pass the output from one program to the input of another program, as discussed in Lacobie's article *Better Software Filter Design — Writing Pipeable User Friendly Programs* in issue #29. I'd also like to add output redirection, which is sending output from a program to a device other than the default which is usually the console; and input redirection, which is taking input for a program from something other than the keyboard (these are also discussed in Lacobie's article).

These features are available with PC/MS-DOS, but there is no reason why we can't implement them under CP/M. One of the things which I like about the CP/M systems that I am using is that the operating system is entirely on disk (some systems have part of the O/S in ROM). Another feature is that I have the BIOS source code.

With the O/S entirely disk based, I can use a number of custom modifications for different applications, and the BIOS is easy to modify because I have the source code. The CCP and BDOS sections are a little more intimidating, but after studying the output from C.C. Software's CP/M 2.2 Source Code Generator, and reviewing Hilton's New-Dos series, I feel ready to tackle the entire operating system (if you are serious about the CP/M O/S, you must get C.C's SCG for CP/M 2.2).

I'm not going to write a huge *do-everything* system, and I don't intend to replace ZCPR3.3, which I'll continue to use for normal operations — what I plan is a series of special O/Ss for non-conventional uses. For these special applications I'll strip out every part of the CCP, BIOS, and BDOS which isn't being used, and add any features which I want for that application.

The reason I'll strip the O/S is to save space and speed the operation. For an extreme example, a data gathering station may not even have an attached console or disk drive, and the O/S may be ROM'ed. In this case it might be easier to start from scratch rather than modifying CP/M, but there are other in-between cases where it will be easier to start with an existing system. I call this "BIOS Bashing." My dictionary defines bash as: *1 to strike with a crushing or smashing blow. 2 a crushing blow. 3 a lively party; a wildly good time.* The definition you choose may depend on the outcome, but I consider BIOS Bashing to be the heart of computing, because that's where the real action starts.

The stripped down O/S will free up RAM space for features such as pipes, redirection, and installable device drivers (something else I'm stealing from PC/MS-DOS). Where I need special O/S features which can be called from a program which runs under conventional CP/M or ZCPR3.3, I'm even thinking about reading in the special O/S, block-moving it up into the top of high memory, running the special section, and then using a short bootstrap loader to reload the regular system. In this case I may have to write any passed parameters out to a temporary disk file, and read them back in. This may sound nuts, but why have all that RAM taken up by unused portions of the O/S while running special applications which could make good use of the additional RAM? We overwrite the CCP while we are not using it, why not overwrite the BIOS and BDOS while we are not using them? Only specially designed custom programs will operate under these conditions, but that's why we program, right?

I like CP/M and the Z80, but I am facing the problem of the 64K memory limitation. While the HD64180 is an improvement, I think that I'll have to go to Hawthorne's 68000 system with its large continuously addressable RAM and position independent programs in order to satisfy myself. In the meantime, I'm bashing CP/M in order to develop my skills and decide what I want in a system.

A lot of my CP/M development work is being moved to my Morrow S-100 Z80 systems which provide 24-bit extended addressing and additional 64K banks of RAM — I just have to dig into the manuals and figure out how to use the banks for programs and systems.

Even though much of what I do will be rather strange and not fully applicable to other systems, I'll publish some of it in order to give you something to think about. If you want to hear more about it, you'll have to let me know by responding with your comments and telling us what you are doing.

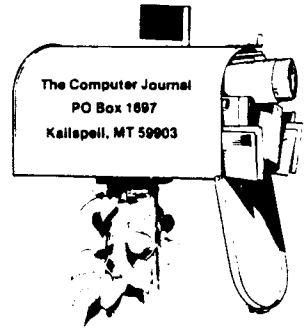
Z80 or 8080?

Digital Research made the smart move of supplying their ASM assembler with their CP/M operating system, but it was written for the 8080 CPU which was in use then and now we are (almost) all using systems with the much better Z80 CPU — but too many people are still using their old ASM 8080 assembler because it came with the system.

We're doing our eight bit systems a disservice by continuing to use the 8080 code (I admit to using it for short quick & dirty programs) because we are comfortable with it, and it's past time to switch over to Z80 assemblers. One of the reasons for staying with ASM is that it came free when assemblers were high priced, but there are now excellent low priced Z80 assemblers such as SLR's Z80ASM (plus their new linker which uses bank switched or disk based overlays) and Echelon's revised ZAS. These superior products are so much better than ASM, that anyone doing any serious work should switch over.

TCJ will stress the use of Z80 code, and hope that any ASM diehards out there will follow along. The few 8080 and 8085 users out there will have to convert the code (Can anybody suggest a public domain converter?) ■

Reader's Feedback



An Avid Reader

I never write letters to magazines in any form, but your editorial #29 struck a responsive chord, and I thought you'd be interested in the reasons why at least **one** of your readers bought the whole back run of TCJ (as of Micro-C, when I first descovered it a few years ago, dito Micro/Systems) — and it relates to your editorial. First, I never program, because I can't — I can't make my Kaypro IV do anything because I don't know how.

I don't know any language, really — am just now struggling thru some basic books on CP/M (Waite & Cortesi), and look forward to learning Z80 assembler, Turbo Pascal® and BDS C® , so it fascinates me to read about how you operate, ditto Dave Thompson's editorial comments in Micro-C, even if I'm not there yet.

Frankly, I know my little Z80 CP/M world is on its way out, so I want to preserve as much of the literature surrounding it as possible — for when I do get around to learning it all. I don't especially look forward to more "User Friendly", for I think I've got lots to learn just from my little 8-bit system, and that's really what it's all about, isn't it?

Anyway, I do love going thru all this stuff that's really much over my head. But what you say about "us" providing our own Technical Support — by sharing our knowledge — **really** appeals to me: it's one big factor that attracted me (a long-time college professor of English) into computer-owning (I don't really dare say into computing....)

So — I have no gripe with Perfect Writer — I don't really aspire to own the latest or fanciest — but I do look forward to completing the initial course of study I've embarked on — and your free-wheeling editorials really help share the flavor of what it's all about. Also, it's a secure feeling knowing I have all this reference material (including Dr. Dobbs & Computer Language) for my future perusal and learning.

So, probably I'm not a *typical* reader, but certainly an avid one. Again, many thanks.

B.T.

PS: Just read Bill Kibler's article mentioning school teachers. At least I can type, but all the rest of his characterization fits. Also, let me add that I do have the two Bell Labs UNIX manuals and really aspire to be able to comprehend (if not necessarily to actually write) large systems based on this O/S.

B.T.

Apple User

I am currently using a PCPI Applicard, modified for 8MHz operation and running on an Apple IIe. Peripherals include a 3Mb ramdisk, 10 Mb Winchester, Applied Engineering 3.6 Mhz accelerator card (for 65C02 on motherboard), Anchor Lighting 2400 modem, two 8" DS/DD drives, and an Appe Imagewriter I printer. I have also recently purchased a Kaypro 10-83, but have not done much hacking with it, other than to order an Advent Turbo-ROM. I am convinced that the Apple/PCPI combination offers the optimum tradeoff of performance vs. cost of system expansion. A wide variety of peripherals are available for the Apple bus, and tend to be orders of magnitude cheaper than comparable products for single board computers and/or Kaypro's.

I have done a fair amount of contract programming for the Apple CP/M environment (device drivers, mostly). Early in 1987 I developed a piece of software for scheduling the operation and signal routing of 24 VCRs in an industrial setting. The development language was Turbo Modula-2, and I probably was the first individual to use it for such a large project — due to the kindness of Dave McCord and Echelon, Inc. in providing me with an advance copy. If you are interested, this might be a good subject for an article. Keep up the good work.

S.H.

Editor: I keep looking at my Apple II+ for interfacing because it is so easy (and cheap!) to get the interface cards. How many others are still using Apple IIs, and are you interested in an article on using them for interfacing to controllers?

CP/M User

I really like your magazine. I have an Ampro Little Board Plus with a 20 Mb hard drive and Z System (ZCRP3.3/ZR-DOS).

Naturally I would like to see more High Performance Enhanced CP/M articles, and also software upgrades (if there ever are any), coverage of Z280, and HD64180 systems. These are powerful machines and with Z System they do things that IBM still cannot do.

If you are planning to go PC, just send my check back, I am not that desperate yet!

L.C.

Editor: The great news regarding CP/M software upgrades is that MicorPro has released the CP/M Version of WordStar, which we are using to write and edit this issue. It's a tremendous improvement, and all CP/M users should support their effort!

An Open Reply to Jerry Nelson

It was very refreshing to read Jerry Nelson's open letter in TCJ issue #28.

His view of the world as a person who simply wants to get a job done comes as a breath of cold clean air to us, and I hope many others.

(Continued on page 40)

A Double Density Floppy Disk Controller Algorithm for the CP/M Operating System

by Donald C. Kirkpatrick

Have you been looking longingly at those new 1.2 Megabyte floppy disk drives, wishing you could upgrade your system? When I started building my CP/M system from scratch, one thing was certain, it would have to read standard 8" exchange format disks. But I always knew that some day I would want to upgrade to a double density format. When I finally made my move, I set two goals: increase the storage per disk as much as possible and improve the system throughput with concurrent processing. When you decide to make your move to the higher density drives, the ideas, design considerations and compromises, problems, and algorithms given here should help ease your transition. Believe me, the performance increase is well worth the time invested.

The ideas presented here are of particular interest to those of you who are upgrading your 5¼" systems. With two exceptions, the newer 5", 1.2 Megabyte drives (for example, the Panasonic JU-475) are electrically identical to the older double-sided 8" drives. The two exceptions are: the number of tracks on the 1.2 Megabyte drives has been increased from 77 to 80 and there is now a motor-on control. If either of these differences change any result, that will be noted at the appropriate time.

My system has two eight inch, single sided disk drives which can be used as either single density or double density. When work began, the single density system had a single 128 byte disk buffer with no overlapping processing; the total storage capacity (directory plus files) being 243K bytes per floppy disk. When work was completed, the total increase in throughput was approximately 100%, with 90% due to the larger disk buffers and concurrent processing, and 10% due to a predictive read-ahead scheme. The total disk space available to files and the directory was increased to 684K. This performance increase did not come for free however. The total operating system size increased by 3K, from 7K to 10K, with 2K of the increase due to new read and write disk buffers.

Introduction to CP/M's Disk Organization

The CP/M operating system is a single user system designed to be run on an 8080 or any of its descendants. The CP/M system is described in detail by many books and publications, so only the important file system specifics are presented here. For further information, consult one of the publications listed in the bibliography.

CP/M performs all disk input/output in 128 byte logical units called records. Occasionally, when describing a single density floppy disk system, the terms *sector* and *record* have been used interchangeably because the physical disk sector size is also 128 bytes. On a double density system, the sector size is never 128 bytes and the two names are not interchangeable. I will always use the term record when referring to 128 byte data blocks that programs read and write. The term sector is reserved to refer to the smallest data unit written or read on a disk.

CP/M uses the name *allocation unit* to describe the minimum incremental *chunk* of space allocated to a file. A single density exchange-format disk is divided into 243 allocation units of 1K each. Two allocation units are reserved for the directory. The space remaining can be divided into one 241K file or 241 files of 1K (if the directory could hold 241 entries). The double density system designed here changes the allocation unit size to 4K and the number of allocation units to 171 (4K × 171 = 684K).

The CP/M disk is organized so the operating system starts on track 0 and continues for as much space as required. On a single density system, two full tracks (8K) are normally required to hold the operating system. On a double density system, one track holds 9 sectors of 1K. The operating system growth, due to the change to double density, does not force the use of a second system track because 2K of the 10K space is uninitialized disk buffer. See Table 1 for a summary of data units relevant to the current discussion and Table 2 for a summary of disk drive characteristics.

The directory immediately follows the system track(s). Its size changed, but the minimum space is one allocation unit. All space after the directory is allocated to file storage. Each directory entry is 32 bytes long. Sixteen bytes form a table of allocation unit numbers owned by the file. The other sixteen bytes contain the name and other housekeeping items. Four directory entries fit into one 128 byte record.

The standard CP/M single density system reserves two allocation units (2K) for 64 directory entries. In the double density format used here, the directory size is increased to 128 entries since the disk size is slightly more than double. Because I chose an allocation unit size of 4K, the directory will fit into one allocation unit. The directory entry allocation table contains unit numbers assigned to that file. When the highest allocation unit number on the disk is less than 256, each allocation unit number fits into one byte and each directory entry table holds sixteen allocation unit numbers. When the highest allocation unit number is greater than 255, it takes two bytes to store one number, and each directory entry table holds only eight allocation unit numbers. (Now you know why the allocation unit size was changed to 4K, the maximum allocation unit number will not exceed 255; not so with 1K or 2K units on a 684K disk.) When a file has less than sixteen/eight allocation units, the directory entry table will have some empty entries. Such an unused entry contains a 0 since allocation unit 0 is always dedicated to the directory, never a file.

When a disk is *full*, it will be for one of two reasons: either there is no free allocation unit to assign to a file requiring more space, or there is no free directory entry to store the file's name and its list of allocation units. Choose your directory size carefully; your disk is full if you run out of directory entries, even if there is free space on the disk.

The CP/M operating system is logically partitioned into three separate functional modules: the Console Command

Processor (CCP), the Basic Disk Operating System (BDOS), and the Basic Input Output System (BIOS). One of the strengths of CP/M is that all the hardware-specific input/output drivers are located in the BIOS. These three modules communicate through a standard set of subroutine calls. Since this is a hardware-specific upgrade project, only the BIOS need be modified. Prudent BIOS modifications result in complete upward compatibility.

Of the 17 BIOS subroutine calls, only seven relating to disk operations need be described here; only two of the seven require any significant changes. Parameters are passed to these subroutines through the processor registers. Any results are also returned through the registers.

- SELDSK — Specifies which disk is referenced in all future disk subroutine calls. This routine need not be altered to upgrade to double density.
- HOME — Move the head of the disk specified in the last SELDSK call to track 0. This routine need not be altered to upgrade to double density.
- SETTRK — Specifies which track is referenced in all future disk reads and writes. The standard Digital Research BDOS issues spurious SETTRK commands to track 0 and but never follows with any reads or writes. For this reason, it is better to actually seek the read/write head to the track required only when the read/write command finally arrives. This routine need not be altered to upgrade to double density.
- SETREC — Specifies which record is referenced in all future disk reads and writes. Note this is the logical record number. On a single density system, this is also the physical sector. On a double density system, this number must be converted by the BIOS to the physical sector and offset into the sector. While track numbers start with 0, sector numbers start with 1. This routine need not be altered to upgrade to double density.
- SETDMA — Specifies the memory source/destination address for all future reads and writes. Note the length is always 128 bytes. This routine need not be altered to double density.
- READ — Performs the read of the record specified in the previous calls to SELDSK, SETTRK, and SETREC into the memory destination specified in the last call to SETDMA.
- WRITE — Perform the write of the record specified in the previous calls to SELDSK, SETTRK, and SETREC from the memory source specified in the last call to SETDMA.

When a record and sector are both 128 bytes, no special consideration need be taken; just write the record to the disk. When a sector contains more than one record (always true on a double density disk), it may be necessary to preread the sector to retain valid, previously written data in the adjacent records. When the record is being written to a newly assigned (empty) allocation unit, there is no data in the adjacent records and no sector preread is required. When more than one record fits into a sector, it is more efficient to defer the write to disk until the sector is full. However, when the write is to a directory, the write must be completed immediately. Directory writes always require prereads. The BDOS passes the WRITE routine a parameter that helps the BIOS decide if a sector preread is required. This *write-type* parameter has the values and meanings:

- 0 — This is a normal write operation. This kind of write may require a sector preread and may be deferred.
- 1 — This is a write to a directory record. This kind of write may require a sector preread and must never be deferred.
- 2 — This write is the first write to an allocation unit that has just been assigned to a file. Since the allocation unit is empty, no sector preread is ever required. The write may be deferred.

How the BIOS uses this information is of vital importance. Ignoring this information leads to significant performance degradation; using it incorrectly results in lost data and files. More will be said about the type-0 and type-2 writes later.

Fundamental Blocking/Deblocking Considerations

Disk Sector and Buffer Size

My CP/M system had been used for several years and experience had shown the resource most often in short supply was disk space. I decided floppy disk space would be maximized, but that reduced the space available to application programs. *(Ed: The program space is reduced because of the space taken by the larger disk buffers).* Not to worry, if an application program required more memory, the old smaller single density version could be used. The larger the physical sectors on the disk, the greater the storage capacity, because there are fewer inter-sector gaps to waste space. The sector size was chosen to be 1024 bytes. This size resulted in nine sectors per track, 76 tracks per disk (for files and the directory), and produced the maximum possible capacity (684K) for a single sided, double density floppy disk. The operating system disk buffers and the sector had to be the same size because any disk read or write always transferred an entire sector.

Number of Disk Buffers

In general, increasing the number of disk buffers increases throughput. However, in this system, the buffer size is so large that too many buffers would have unreasonably increased the operating system size. I observed that most programs would read from one file while writing to a second file (for example, assemblers). If there were only one disk buffer, thrashing could be a problem. It seemed that two 1024 byte buffers, one dedicated to reading and one dedicated to writing, provided the best trade-off between thrashing and space.

One way to increase performance is to maximize the overlapping of the input/output with program execution. For a write, if the file already exists, a preread of a sector will be required if the sector containing the record is not already in memory. Rather than wait for this preread, a third holding buffer of 128 bytes stores the write record while the sector preread occurs. This permits the system to copy a record into the holding buffer, start the preread, and return to the calling program. When the interrupt signals the completion of the sector preread, the holding buffer is copied into the write buffer. The holding buffer protects the data. The calling program may start to change it before the preread is completed.

Predictive Read-Ahead

When a program requests a disk read, the system cannot return to the calling program until the read is completed. Thus, if the data is not in a disk buffer, the program must wait until the disk access is complete. If it is possible to predict what sector will be required next, then the read could be started before the calling program requests it. This allows disk accesses and processing to overlap and raises the throughput. The problem is how to predict the next required sector.

CP/M's file allocation structure does not force contiguous file allocation. An allocation unit contains four sectors. Those four sectors must be contiguous. For a specific file, allocation units do not have to be contiguous, but they usually are. By its construction, the directory must be contiguous. Therefore, the following algorithm was used to predict when and where the next disk read was to be:

- When the last record of the current read buffer is read by the calling program, increment the sector number (and the track number if required) then read this sector into the read buffer.

Notice this algorithm assumes a file will be accessed serially. CP/M supports random access files, and this algorithm may actually degrade performance for randomly accessed files. This risk is acceptable because programs that perform random access are very rare. Notice a predictive read-ahead may fetch a sector that is never used.

Concurrent Processing

Concurrent processing in this context means the floppy disk controller and the microprocessor are simultaneously performing useful work.

The original single density system had almost no concurrent processing. A simple foreground/background scheme was implemented to support concurrent processing. All applications programs run in the background and floppy disk controller routines run in the foreground. All foreground routines are invoked by an interrupt from the floppy disk controller. When an interrupt occurs, the processor interrogates the floppy disk controller and determines which foreground task to run. If there is useful work for the floppy disk controller to perform, the command string is sent to the controller and the task starts. While the controller completes the task, the background program continues execution.

Since several foreground tasks might be invoked by one interrupt, there is a prioritized foreground queue of fixed length. Since the maximum number of items in the queue, three, is known a priori, it is not possible for the queue to overflow. The three tasks are: preread sector, copy holding buffer to write buffer, and write sector. Other tasks in the queue (but not at the same time) include: recalibrate drive, check density, retry last read/write, and read sector. The queue server is very simple; it only checks a set of flags to decide which tasks to invoke.

Foreground/background communication is accomplished via a set of flags located in memory (see Table 3). My system uses a Z80 and fortunately the Z80 has atomic bit set and reset instructions; without these instructions, the critical section problems would have to be solved differently. When a background task needs a disk operation performed, it just sets the appropriate flags and uses the floppy disk controller to generate an interrupt. The foreground finds the flags set and performs each operation. As an operation is complete, the floppy disk controller generates another interrupt and the foreground routine clears the flag, thus informing the background that the operation is complete.

Error Processing

The calling program expects the BIOS to return a flag indicating if a read or write is successful. Because writes occur in the foreground and may be deferred, the write routine always returns success. The entire burden of an unsuccessful operation and error recovery falls upon the BIOS. Most soft errors are caused by the read/write head not being correctly located over the track on the floppy disk. This type of error can be corrected by stepping off the bad track and re-seeking the track. This step on/off is performed automatically by the BIOS. If, after 15 retries, the operation is still unsuccessful, the BIOS reports the error directly to the system console and prints the status bytes provided by the floppy disk controller. This allows the operator to determine the severity of the fault and what corrective action is required.

Determining the Density

The method of automatically determining the density of a floppy disk is quite simple. The floppy disk controller monitors the disk drive and informs the operating system when a new disk is inserted. The operating system then instructs the controller to report back with the first sector ID it can find on the disk in double density mode. If the ID read is successful, then the disk is double density; if the read fails, then the disk must be single density.

Keeping Track of Unallocated Writes

One key performance increase comes from carefully keeping track of writes to a newly allocated unit. If this is done correctly, unnecessary sector prereads will be eliminated. Remember the write-type parameter passed by the BDOS to the write routine? A 4K allocation unit contains 32 records. It is only on the first write to the first record of this unit that the BDOS will flag the write as a type-2 write (first write to new unit). All other 31 writes will be flagged as type-0 writes. It is entirely the responsibility of the BIOS to remember this is still a newly allocated unit. Remember, blocking/deblocking is the job of the BIOS; if writes are not deferred, then sector prereads could very well be required.

This is the algorithm used: Save the write-type parameter sent when the allocation unit number changes. On subsequent writes to the same unit, if the record number is one greater than the previous record, leave the saved parameter alone. When the next write record number is not exactly one greater but is in the same allocation unit, the saved write-type parameter is reset to type 0. This way, if the first write is unallocated, it will stay so as long as possible. The test for an unallocated write is a 2 in the saved write-type parameter rather than the currently passed value.

The Read Algorithm

The processing of a read request is fairly straightforward when compared to a write request. The steps in the algorithm are as follows:

- Step 1: Switch to a local stack. There is no way of knowing how much stack space is available from the caller. Better safe than sorry.
- Step 2: Wait for any disk operation in progress to complete. If a read is needed, the read routine will be building a floppy disk controller command string. In the case of a retry, the string may be executed up to 15 times. Even as we speak, a retry may be in progress. It is just too risky to continue until all previous operations are complete.
- Step 3: Check for the drive door closed and a disk inserted. This must be postponed until after step 2 because a command string must be sent to the floppy disk controller.

There must be a disk in the drive because the read routine needs to know if the read is single or double density. If the disk in the drive is single density:

• Step 4: Build the floppy disk command string, set the read flag, and start the foreground task.

• Step 5: Wait for the foreground to complete. Note the foreground clears the read flag when the read is complete.

• Step 6: Transfer the record from the 128 byte temporary holding buffer to the DMA destination. If your DMA hardware allows, the read could transfer directly to the final destination; my DMA controller can't.

• Step 7: Restore the stack and return to the caller.

But if the disk in the drive is double density:

• Step 4: Find out if the record is already resident in the write buffer or the read buffer. Skip to step 7 if the record is already present.

• Step 5: Build the floppy disk controller command string, set the read flag, then start the foreground read.

• Step 6: Wait for the foreground to finish. Note the foreground clears the read flag when the read is complete.

• Step 7: Transfer the record to the calling program.

• Step 8: If the requested record is the last record in the read buffer, then construct the floppy disk controller command string and start the predictive read-ahead to refill the read buffer.

• Step 9: Restore the stack and return to the caller.

The floppy disk controller command string depends upon the sector and track, the kind of operation, and the kind of floppy disk controller you have. The string must always be built before the flags are set just in case an interrupt (door closed for example) happens. But the command string must be built after all previous disk operations are complete, lest the command in progress be overwritten.

The Write Algorithm

The write request is complicated by the need for prereading a sector when the write is into a preexisting file. The BDOS simplifies this task by informing the BIOS when a write is to an empty allocation unit. A sector preread is not required on sequential writes to a previously unallocated unit. After the first write, keeping track of whether or not a preread may be required is the responsibility of the BIOS.

In general, any write may be postponed until a more convenient time, for instance, when the write buffer is full. However, special directory writes occur which can never be postponed. Once the foreground write operation is invoked, the BIOS returns to the calling program to allow maximum concurrency. The write algorithm is as follows:

• Step 1: Switch to a local stack. There is no way of knowing how much stack space is available from the caller. Better safe than sorry.

• Step 2: Wait for any disk operation in progress to complete. If a write is needed, the write routine will be building a floppy disk controller command string. The previous disk command string may still be executing.

• Step 3: Check for the drive door closed and a disk inserted. This must be postponed until after step 2 because a program string must be sent to the floppy disk controller.

There must be a disk in the drive because the write routine needs to know if the write is single or double density. If the disk in the drive is single density:

• Step 4: Transfer the data to the 128 byte holding buffer.

• Step 5: Construct the floppy disk controller command string, set the write flag, then start the foreground write. Note the foreground clears the write flag when the write is complete.

• Step 6: Restore the stack, and exit to the calling program. No need to wait for the operation to complete, the foreground program will do everything. The calling program may be able to use this time effectively.

If the write is to a double density disk, the record written may be in the write buffer, the read buffer, or out on the disk:

• Step 4: Find out if the sector that contains the record to be written is already in memory. Apply the unallocated unit algorithm to update the unallocated unit write-type flag. If the sector containing the record to be written is in the write buffer, transfer the record and go to step 8.

• Step 5: Since the write buffer does not contain the record, if a write is pending, start the foreground write and wait for it to complete. Build the floppy disk controller command string.

• Step 6: If the record is in the read buffer, transfer the read buffer to the write buffer, mark the read buffer empty, transfer the record to the write buffer. Go to step 8.

• Step 7: The sector containing the record to be written is not in memory. Determine if a sector preread is necessary by checking the unallocated unit write-type flag. If no preread is necessary, go to step 8. Otherwise, transfer the record to the 128 byte holding buffer and set the read and copy flags.

• Step 8: If this is a directory write, set the write flag, reset the write pending flag, and start the foreground write. If this is not a directory write, set the write pending flag. Note the foreground clears the write flag when the write is complete.

• Step 9: If the read or write flag is set, start the foreground task by initiating a seek to the desired track.

• Step 10: Restore stack and exit to caller.

Table 1
A Summary of Some Basic CPM Data Units

Unit	Description
Record	The basic disk input/output unit that all CPM programs use. Always 128 bytes, the size was determined by the creators of CPM.
Sector	The amount of data that is always transferred between memory and the floppy disk. The size is determined by hardware considerations. On a single density system, the size is usually 128 bytes. On a double density system, the size can be anything from 256 bytes to 16K bytes, always a power of 2.
Allocation Unit	The minimum amount of data that can be assigned or "given" to a file. The size is determined by the disk capacity and the size of the directory. On a single density system, usually 1K. On a double density system, usually a multiple of 2K.
Directory Entry	The space consumed by one entry in the directory. Always 32 bytes. If a file has more allocation units than will fit in one directory entry table, then another directory entry is assigned to that file. Four directory entries fit in one record assigned to the directory.
Directory Entry Allocation Table	Always a sixteen byte field inside a directory entry, it contains the allocation units assigned to the file that owns the directory entry. If the maximum allocation unit number is less than 256, the table contains 16 entries; greater than 255 and the table contains 8 entries.

Table 2
Drive Characteristics

	Single Density	Double Density
Record Size (Bytes)	128	128
Tracks	77	77
Sectors Per Track	26	9
Records Per Sector	1	8
Records Per Track	26	72
Record Capacity (128 Bytes)	2002	5544
Drive Capacity (Kilobyte)	250.25K	693K
Tracks Reserved for CPM	2	1
File and Directory Capacity	243K	684K
Allocation Unit Size (Bytes)	1024	4096
Allocation Unit Size (Records)	8	32
Allocation Units	243	171
Tracks for Directory and Files	75	76
Directory Allocation Units	2	1
Directory Records	16	32
Directory Entries (32 Bytes)	64	128
Directory Capacity	2K	4K
File Storage Capacity	241K	680K

For a double sided disk, all the per track and per disk figures are doubled.

Foreground Tasks

All the foreground tasks are interrupt driven. The necessary command strings are constructed in background, the appropriate flags are set, and the background task generates an interrupt by forcing the floppy disk controller to seek to the required track. It is of vital importance that the above items be done in this order. Chaos would result if the flags were set and an interrupt occurred (for some external reason) before the command string was built.

The specific interrupts that cause a switch to foreground processing are:

- Disk drive door being opened; the drive becomes not ready.
- Disk drive door being closed; the drive becomes ready.
- Previous read or write has been completed.
- Previous seek operation has been completed.

When an interrupt occurs, the floppy disk controller is interrogated and the reason is determined. The appropriate subroutine is then called to handle the interrupt.

After every interrupt, all status flags are checked for any pending tasks that need to be initiated. The steps the foreground program executes are:

- Step 1: Get the floppy disk controller status bytes to determine the reason for the interrupt. If the reason is the successful completion of a read or write command, clear the retry flag and the read or write flag as appropriate. If the reason is the unsuccessful completion of a read or write command, set the retry flag.
- Step 2: If a drive has just become ready, set the recalibrate flag, start the recalibrate operation, and go to step 10.
- Step 3: If a drive has just become not ready, mark any buffers containing that drive's data as empty and go to step 10.
- Step 4: Check for the recalibrate flag. If so, reset the recalibrate flag, set density check flag, start density check, and go to step 10.
- Step 5: Check for the density check flag. If so, clear the flag, save the density check result, and go to step 10.
- Step 6: Check for the retry flag. If so, step the drive, send the command to the floppy disk controller again, and go to step 10.
- Step 7: Check for the read flag. If so, start the read operation and go to step 10.
- Step 8: Check for the copy flag. If so, copy 128 bytes from the holding buffer to the write buffer.
- Step 9: If the write flag is set, start the write operation.
- Step 10: Return to background mode.

Problems To Be Avoided

All sorts of problems can arise if the read buffer and the write buffer ever contain the same sector. This would have the same effect as two files on the disk using the same sector. Unpredictable and unrepeatable results would follow. Therefore, great pains are taken to assure this never happens. If you are not careful, there are several scenarios where this could occur. For example, on a predictive read-ahead, the write buffer could contain the next sector beyond the current read buffer sector. All these potential problems are solved by assigning higher priority to the write buffer. Every read must check the write buffer first to be sure the sectors are not the same. Every write must check the read buffer and if it contains the desired sector, transfer its contents to the write buffer, then mark the read buffer empty.

The major problem encountered during the system testing phase turned out to be an unanticipated critical section problem. The old single density system never had a critical section because concurrent processing was not attempted. The critical section in this implementation occurred when both the foreground task and background task attempted to interface with the floppy disk controller. Once the problem was identified, it was rather easy to solve. Now, when the background wants to communicate with the floppy disk controller, it first turns off interrupts, then polls the main status register of the floppy disk controller. The contents of the register indicate whether the controller is busy or idle. If busy, the background re-enables interrupts and waits. If idle, the background completes the task at hand and then enables interrupts. This guarantees that the foreground and background never talk to the controller at the same time.

Another problem discovered in testing concerned exception handling. The problem occurred when an attempt was made to read on a disk drive that did not have a disk in it. The old single density system reported the error to the console and waited for a disk to be inserted; the double density system was to do the same.

Table 3
Foreground/Background Flags

Flag	Meaning When Set (all flags are set when true)
Read	Read data to buffer from disk. Set by background; cleared by foreground.
Write	Write data from buffer to disk. Set by background; cleared by foreground.
Pending	Data in the write buffer has not yet been flushed to the disk. Set and cleared by background.
Retry	Unsuccessful read or write operation detected. Retry in progress. Set and cleared by foreground.
Recalibrate	Recalibrate of disk drive in progress. Set and cleared by foreground.
Density	Density check in progress. Set and cleared by foreground.
Copy	The data in the 128 byte holding buffer must be copied into the write buffer before it is sent to the floppy disk. Set by background and cleared by foreground.

The problem was that the background routine returned to the caller as soon as the disk was inserted. It should have waited for the density check. Under these conditions, the caller thought a single density disk was in the drive when it really was double density or vice versa. The solution required restructuring of the door check routine.

When I first tested the new double density routines, I found some programs losing data from the end of their write files. This was due to these programs not closing their output files. Rather, they simply wrote the last record and then performed a warm boot. The final write buffer was never being flushed to the disk. This was solved by having the warm boot code check for a deferred write and flushing the buffer if needed. This cleared up the problem.

Related to the no disk problem above is a fundamental problem that does not seem to have a solution. The scenario is this: Suppose a drive has a single density disk in it and the BDOS knows about this disk. If the door is opened and the disk is traded for a double density disk, the BIOS knows this and performs all the required actions to read the disk. But somewhere inside the BDOS is a data field that cannot be reached. The next time the BDOS tries to read the disk, it uses the wrong sector and track numbers. It thinks there are 26 records per track when there really are 72 records per track. This problem appears to have no solution other than to perform a software reset (Control-C warm boot) of the system. The bottom line is, if CP/M thinks it knows what density disk is in the drive, then CP/M had better be right.

Results and Conclusions

Operating System Size — The total space consumed by the operating system grew from 7K to 10K, an increase of 3K or a 43% increase. The bulk of the increase was due to the addition of the two 1024 byte disk buffers.

Floppy Disk Storage Capacity — The space available for file storage on one floppy grew from 241K to 680k or 182%. This disk format is a non-standard format, but interchangeability is not an issue. If a floppy disk needs to be interchanged with another CP/M system, the single density format is used. The single density format is an internationally recognized standard that every eight inch disk system must support.

Fragmentation — The nature of the CP/M directory system results in fragmentation or wasted space. Because the size of the allocation unit increased from 1K to 4K, increased fragmentation would be expected. This conclusion was substantiated by actual measurements. A single density disk containing 31 files required 189K of space. When this disk was converted to a double density disk, the space allocated increased to 256K, due entirely to fragmentation. One would have predicted the fragmentation would be 2K × 31 or 62K additional, but the final result was 67K (256K − 189K). This difference was caused by the large number of small files (less than 1K) on the disk.

Speed — The file handling capabilities of the system were greatly enhanced. Two benchmarks were run, a strictly disk to disk copy test timing and an assembler test timing. The results are outlined in the table below:

	Single Density	Double Density Predictive Read-Ahead Off	Double Density Predictive Read-Ahead On	
Copy	4:30	2:38	2:20	
Assembly	1:45	1:00	0:55	Min:Sec

The assembler test was the assembly of this BIOS portion of the operating system. The file copy was the duplication of a disk on one drive to a blank disk in the second drive. There were 31 files on the disk and the total space filled on the disk was 189K. These figures show the new double density system was about twice as fast, with 10% of the improvement due to the predictive read-ahead.

Compatibility — The new dual density system is now being used regularly. M80/L80, WordStar, SuperCalc, ZSID, and DESPOOL run successfully and no compatibility problems occur.

Two programs had to be re-written, but it was known beforehand this would have to be done. These two were the disk format program and the system loader program. The format program had to be modified to format double density disks. The system loader program, which formerly loaded the operating system onto tracks 0 and 1, was modified to load the system onto track 0 only. Fortunately, both programs were small.

Bibliography

The following books and publications have proven to be of exceptional value in providing information about CP/M or about a floppy disk controller that is particularly good at concurrent processing, the NEC 765.

Cortesi, David E. Inside CP/M: A Guide for Users and Programmers. Holt, Rinehart and Winston, New York, 1982.

CP/M 2 Alteration Guide. Digital Research. Pacific Grove, California. 1979.

Hogan, Thom. Osborn CP/M Users Guide. OSBORN/McGraw-Hill. Berkeley, California, 1981.

Murtha, Stephen M. and Mitchell Waite. CP/M Primer. Howard W. Sams & Co. Inc. Indianapolis, Indiana. 1980.

SA800 Series Diskette Storage Drive Double Density Design Guide. Shugart Associates. Sunnyvale, California. 1977.

Weiner, Richard and Gregory York. A Single/Double Density Floppy Disk Controller Using the PD765. NEC Microcomputers, Inc. Wellesley, Masschusetts. 1980. ■

Implementing ZCPR3 IOP Support for Ampro

Featuring NuKey, a Keyboard Re-definition IOP

by Rick Swenton

If you are the proud owner of an Ampro Z80 "Little Board" you are fortunate to have one of the finest complements of hardware and software available for a very modest price. Like any well designed piece of hardware, performance is dependent on well designed software. Ampro, in its infinite wisdom, chose to implement support for Richard Conn's ZCPR3 replacement Console Command Processor (CCP). When they created their Basic Input/Output System (BIOS), they built-in the required routines to initialize the special reserved memory locations needed in a ZCPR3 system. At the time the BIOS was written and subsequently revised, ZCPR3 I/O Packages (IOP) were not widely used. This was because IOPs were not really understood by the masses and IOP software was not readily available. Many users did not feel that they needed the features that an IOP provided such as console capture to disk, redirection of console/printer/modem I/O or operating devices in tandem. Richard Conn's original concept of an IOP was to remove the I/O routines from the BIOS and place them into the IOP segment. In this way, many different IOP's could exist and be loaded on demand. Each would provide you with custom specialized I/O tasks. Because IOP's were not very popular, Ampro chose not to allocate space for an IOP in the reserved memory set-up by the BIOS at cold boot time.

Things chance very fast in the 8-bit world. Suddenly, IOP's are becoming very popular. Echelon, the official source of ZCPR3 and a full line of Z-System products, now has three full-featured IOP's available. If you use Ampro's standard BIOS, you can not run any of them without modifying the BIOS. Users of the Micromint SB-180 were running around left and right using IOP's and rubbing my nose in it. Micromint had chosen to provide IOP support in their BIOS. But then the Ampro had been out in the market before the SB-180.

In this article I will show you how to modify the Ampro BIOS to support IOP's. I will also provide an overview of Echelon's NuKey IOP, which is an advanced function key re-definition program that allows you to redefine your function and cursor keys on the CRT terminal.

Changing the BIOS

I am using Ampro's BIOS version 3.8, but the changes are fairly generic and also apply to the earlier versions although you may have problems finding some of the locations in the older versions. Of course, to make the changes, you will need a copy of the BIOS source code - BIOS38.ASM - or whatever version you have. The BIOS source code is available from AMPRO.

In the current Ampro ZCPR3 implementation, the following areas are initialized by the BIOS (addresses are in Hex, earlier BIOS versions did not implement all of these):

```
FFD0 - ZCPR3 External Stack
FF00 - Multiple Command Line Buffer
FE00 - Environment Descriptor
FDD0 - ZCPR3 External FCB
FD80 - ZCPR3 Message Buffers
FD00 - Shell Stack
FC00 - Named Directory Area
FA00 - Flow Control Package
F200 - Resident Command Package
```

We will locate our IOP segment 1.5k below the RCP:

```
EC00 - Input/Output Package
```

So, the first change to make is to define the address of the IOP. Find the IOP and IOPS equates and change their values as follows:

```
Old Code:

IOP     EQU     00000H          ; Redirectable I/O Package
IOPS    EQU     0               ; size in 128-byte blocks

New Code:

IOP     EQU     0EC00H          ; Redirectable I/O Package
IOPS    EQU     12              ; size in 128-byte blocks
```

Next, we need to change the BIOS jump table to re-direct certain jumps to the IOP. The jump table is at the start of the BIOS, just after the ORG BIOS statement:

```
Old Code:

        JMP     BOOT            ; Cold start
WBOOTE: JMP     WBOOT           ; Warm start
        JMP     CONST           ; Console status
        JMP     CONIN           ; Console character in
        JMP     CONNOUT         ; Console character out
        JMP     LIST            ; List character out
        JMP     PUNCH           ; Punch character out
        JMP     READER          ; Reader character in
        JMP     HOME            ; Seek to home position
        JMP     SELDSK          ; Select disk
        JMP     SETTRK          ; Set track number
        JMP     SETSEC          ; Set sector number
        JMP     SETDMA          ; Set DMA address
        JMP     READ            ; Read disk
        JMP     WRITE           ; Write disk
        JMP     LISTST          ; Return list status
        JMP     SECTRAN         ; Sector translate

New Code:

        JMP     BOOT            ; Cold start
WBOOTE: JMP     WBOOT           ; Warm start
        JMP     IOP+12          ; Console status
        JMP     IOP+15          ; Console character in
        JMP     IOP+18          ; Console character out
        JMP     IOP+21          ; List character out
        JMP     IOP+24          ; Punch character out
        JMP     IOP+27          ; Reader character in
        JMP     HOME            ; Seek to home position
        JMP     SELDSK          ; Select disk
        JMP     SETTRK          ; Set track number
        JMP     SETSEC          ; Set sector number
        JMP     SETDMA          ; Set DMA address
        JMP     READ            ; Read disk
        JMP     WRITE           ; Write disk
        JMP     IOP+30          ; Return list status

        JMP     SECTRAN         ; Sector translate
```

Now find the label HDCODE which is just before the Cold Boot Entry:

```
HDCODE  EQU     $               ; End of hard disk code
```

Between this statement and the Cold Boot code, add the following code:

New Code:

```
;
;       This is the Auxiliary Jump Table for the extended IOP
;
IOPRET:
        JMP     CONST
        JMP     CONIN
        JMP     CONOUT
        JMP     LIST
        JMP     PUNCH
        JMP     READER
        JMP     LISTST
```

Be sure you place this code before the label BOOT: because all code after BOOT: is overlayed by system storage after a Cold Boot. We need to have IOPRET resident all the time.

The next set of changes are to the ZBOOT routine.

Old Code:

```
ZBOOT:  LXI     B,3             ; Set up the ZCPR3 command line
        LXI     H,CMDSET        ; . pointers
        LXI     D,Z3CL          ;
        DW      LDIR80          ;

        LXI     B,10            ; Move the automatic command to
        LXI     H,AUTOCMD       ; . the ZCPR3 command line
        LXI     D,Z3CL+3        ;
        DW      LDIR80          ;

        LXI     B,11            ; Move the initial path descriptor
        LXI     H,PATH          ; . to the proper location
        LXI     D,EXPATH        ;
        DW      LDIR80          ;

        MVI     A,0FFH
        STA     Z3WHL           ; Turn the wheel byte on

        LXI     H,ENV           ; Move the environment to
        LXI     D,Z3ENV         ; the proper location
        LXI     B,ENVEND-ENV
        DW      LDIR80

        ENDIF                   ; ZCPR3 init

        XRA     A               ;
        STA     CDISK           ; Indicate disk 0 selected
        STA     HSTACT          ; Set host buffer inactive
        STA     UNACNT          ; Clear unalloc count
        STA     HSTSID          ; Assume side zero
        JMP     GOCPM           ; Initialize & jump to CP/M

LOGMSG: DB      CR,LF,LF
```

New Code:

```
ZBOOT:  LXI     B,3             ; Set up the ZCPR3 command line
        LXI     H,CMDSET        ; . pointers
        LXI     D,Z3CL          ;
        DW      LDIR80          ;

        LXI     B,10            ; Move the automatic command to
        LXI     H,AUTOCMD       ; . the ZCPR3 command line
        LXI     D,Z3CL+3        ;
        DW      LDIR80          ;

        LXI     B,11            ; Move the initial path descriptor
        LXI     H,PATH          ; . to the proper location
        LXI     D,EXPATH        ;
        DW      LDIR80          ;

        MVI     A,0FFH
        STA     Z3WHL           ; Turn the wheel byte on

        LXI     H,ENV           ; Move the environment to
        LXI     D,Z3ENV         ; the proper location
        LXI     B,ENVEND-ENV
        DW      LDIR80

        LXI     H,IOPENT        ; Move the dummy IOP
        LXI     D,IOP           ; to the proper location
        LXI     B,IOPLEN+2      ; also copy the 2 bytes after IOPEND
        DW      LDIR80
        LXI     H,IOPRET        ; Change the CBOOT vector to point
        SHLD    BIOS+1          ; to the IOPRET table

        ENDIF                   ; ZCPR3 init

        XRA     A               ;
        STA     CDISK           ; Indicate disk 0 selected
        STA     HSTACT          ; Set host buffer inactive
        STA     UNACNT          ; Clear unalloc count
        STA     HSTSID          ; Assume side zero
        JMP     GOCPM           ; Initialize & jump to CP/M

;
;       This is the dummy IOP loaded on cold boot
;
IOPENT:
        JMP     IOPEND          ; return 'not implemented'
        JMP     IOPEND          ; return 'not implemented'
        JMP     IOPEND          ; return 'not implemented'
        JMP     IOPEND          ; return 'not implemented'
        JMP     IOPRET          ; CONST
        JMP     IOPRET+3        ; CONIN
        JMP     IOPRET+6        ; CONOUT
        JMP     IOPRET+9        ; LIST
        JMP     IOPRET+12       ; PUNCH
        JMP     IOPRET+15       ; READER
        JMP     IOPRET+18       ; LISTST
        JMP     IOPEND          ; return 'not implemented'
        JMP     IOPEND          ; return 'not implemented'
        JMP     IOPEND          ; return 'not implemented'
        JMP     IOPEND          ; return 'not implemented'
        JMP     IOPEND          ; return 'not implemented'
        DB      'Z3IOP'
        DB      'DUMMY  '

IOPLEN  EQU     $-IOPENT
IOPEND  EQU     IOP+IOPLEN

        XRA     A               ; this byte is at IOPEND
        RET

LOGMSG: DB      CR,LF,LF
```

One more minor change is to be made. On the last page of the listing, find the two labels RESERVE. The second one will be zero. The first should be set to 0EC00H as shown below:

Old Code:

```
        IF      ZCPR3
RESERVE EQU     0FD00H          ; (0FD00H if ZCPR3)
        ENDIF
        IF      NOT ZCPR3
RESERVE EQU     00000H          ; (00000H if no ZCPR3)
        ENDIF
```

New Code:

```
        IF      ZCPR3
RESERVE EQU     0EC00H          ; (0EC00H if ZCPR3)
        ENDIF
        IF      NOT ZCPR3
RESERVE EQU     00000H          ; (00000H if no ZCPR3)
        ENDIF
```

That completes the BIOS changes. You can assemble the BIOS source and check for errors.

IOP Theory

The original concept of an IOP was to remove all console, printer, reader and punch device drivers from the BIOS and place them in the IOP. In this way, separate IOP's could be created for special applications which would otherwise require modifications to the BIOS. For example, you could have an IOP which has data sent to the console port and the modem port at the same time so that two users could be attached to the computer. You could make changes to your BIOS to do this but I am sure you would not want these two devices in parallel all the time. You would then need two separate BIOS's on two separate boot disks. However, with the IOP approach, the console and modem drivers are in the IOP. You could create a standard IOP and a custom IOP with the console and modem in parallel. Each could be loaded on demand thereby transforming standard I/O to a custom one and back again. A side benefit is the decrease in size of the BIOS after the

device drivers are removed. You do lose the space savings to the 1.5k IOP reserved memory but you are allowed an unlimited number of 1.5k IOP segments. This adds great versatility to your BIOS.

Recently, Joe Wright, a prominent member of the Z-System community proposed an extension to the basic IOP concept. Joe's proposal was to leave the device drivers in the BIOS and to have the IOP act as a "front-end" or "pre-processor" to BIOS device driver calls. With this method, the Jump Table at the beginning of the BIOS was modified to direct console, printer, reader and punch calls to the IOP instead of to their respective routines within the BIOS. When a call is made to the BIOS for one of these I/O routines, the BIOS first passes the call on to the IOP. The IOP will perform some pre-processing and then pass the call on to the originally intended routine in the BIOS. This is how Joe Wright's NuKey works. NuKey is an IOP which performs console keyboard key re-definitions. With NuKey, you could program your function keys to be translated into a string of 1 to 15 characters. Let's say that you use NuKey to translate your CRT's function key F1 to the string "DIR<return>." Let's also say that your CRT sends ESCAPE-A when the F1 key is pressed. When NuKey is running, all console input requests are directed through the NuKey IOP first. NuKey is monitoring all characters coming from your keyboard. Suddenly NuKey sees ESCAPE-A come in. Instead of passing those two characters to the BIOS CONIN routine, NuKey sends the string DIR<return> to the BIOS and your screen displays a disk directory because you pressed the F1 function key.

For any IOP to pass the intercepted call back to the BIOS, there has to be an internal jump table placed somewhere in the BIOS. The IOP will know the starting address of this table and just add multiples of three to this address to get where it wants to go.

During the cold boot, the BIOS must initialize the IOP area with a "DUMMY" IOP. This is needed because the Jump Table at the start of the BIOS was modified to pass some calls to the IOP. There had better be something there in the IOP area to handle these calls. At cold boot, we load a primitive IOP which simply passes all calls back to the BIOS through the internal jump table we placed in the BIOS.

If we were to load any IOP into its reserved memory, it would overlay the DUMMY IOP. Then the newly loaded IOP would handle all BIOS requests for I/O. If at some time we no longer wish to have this IOP active, the only way to de-activate it is to load another IOP over it. If we do not want any IOP to be active, we load the DUMMY IOP into the IOP reserved memory. With the DUMMY IOP loaded, the BIOS acts like a standard one.

To create a DUMMY.IOP file, boot your system and do not load any IOP so that the BIOS generated DUMMY IOP is resident in the IOP memory area. Invoke DDT or equivalent and enter the following commands:

```
A0>DDT<ret>
-MEC00,F1FF,100          (This moves the DUMMY IOP from EC00 to 100)
-^C                      (Get out of DDT with control-C)
A0>SAVE 6 DUMMY.IOP      (Save 6 pages to the disk file DUMMY.IOP)
```

The above sequence created a copy of the BIOS generated DUMMY IOP which resided in the IOP reserved memory area into a disk file.

It should be noted here that if you make any changes to your BIOS which cause it to change size or change the location of the internal jump table, you will have to create a new copy of the DUMMY IOP into the DUMMY.IOP disk file. Also, any previously created IOP files such as the ones created with NuKey will most probably crash if you attempt to use them on a BIOS which changed size. You will have to use NUKEY.COM to create new ones, which is a very simple procedure.

The NuKey.IOP

Now that you have modified your BIOS to support IOP's, you can get on with the task of making your terminal's keyboard a friendly place to work. If you are like most computer users, you regularly run several keyboard intensive programs such as word processors, spreadsheets, database managers and even Z-System (ZCPR3). If your applications are like most applications, the keyboard and special function key definitions are different between applications and even when they work, you wish they could be the way YOU wanted them. You also wish that the definitions would remain the same from program to program. Enter NuKey.

Earlier, I mentioned that Joe Wright was a prominent member of the Z-System community. Joe is much more than that. Joe is an experienced Z-System "Power Programmer." When ZCPR3 first came out, we were all very impressed. Joe Wright's Z-System programs are so tightly integrated into the "Z" environment and command full control of the "Z" features that he renews our excitement all over again. NuKey is no exception.

The Basics

NuKey is a keychanger IOP. A keychanger, sometimes called a keyboard redefinition program, can be described as a key translator. When the keychanger is running, it places itself between

the keyboard data and the data's destination, usually the system BIOS. Without the keychanger, the keyboard data is processed by the BIOS. With the keychanger, the keychanger intercepts the keyboard data and examines it. It looks to see if it should pass this data along to the BIOS without modifying it or to translate the data into some pre-determined different data. Normally, the characters such as the letters of the alphabet, numerals and punctuation will not be altered. But you may decide to have the keychanger translate your keyboard's special function keys into strings of characters to perform system commands. For example, your F1 through F6 keys could be programmed to perform the functions DIR<ret>, LDIR<ret>, VFILER<ret>, VMENU<ret>, NULU<ret> and EDIT<ret> respectively. Notice that the carriage return shown as "<ret>" was also programmed into the string sent out by the function keys. You could have also programmed your F1 through F4 keys to generate the control functions ^E, ^X, ^S, and ^D for Wordstar compatible cursor movement. Under NuKey, you can have as many different keychanger IOP's as you desire. For example, my SYS.IOP system keychanger translates my keyboard's "ERASE" key into the string "CLS" which is the Z-System command to clear my screen. When I run my word processor, I load WP.IOP which translates by keyboard's "ERASE" key into ^G (control-G) which is the Wordstar compatible erase character function. When I use the modem program, the MODEM.IOP allows me to send my name and password to sign-on to a remote computer with the touch of two keys.

To make things confusing, NuKey is more than a keychanger. NuKey is a "string changer." This means that NuKey can not only translate single character keys into a string of one to 500 characters but it can also translate keys which themselves generate from one to 15 characters into another string of one to 500 characters. Those of us who have CRT terminals which generate more than one character when the function and cursor keys are pressed can really appreciate this feature of NuKey. Almost any character or string of characters generated by a keyboard can be translated into another string of characters by NuKey.

Another very powerful feature of NuKey is the "extend" key. The extend key can almost be thought of as a special keyboard "shift" key. Using the extend key, you can assign a translation to any key on your keyboard, but the translation will only take place if the extend key was pressed first. For example, you can use NuKey to allow a control key, such as control-S, to be passed unchanged but be translated to some other character or string only if the extend key was pressed before the control-S was pressed. This allows you to keep any or all keyboard keys unchanged yet still have a complete set of definitions assigned to them. The translation to the special definitions will only take place if the extend key is typed first.

The Details

NuKey requires a Z80 compatible microprocessor. This includes the 64180 and the NSC-800. NuKey also requires that you be running ZCPR3 and that your BIOS supports the implementation of IOP's as described in the Echelon publication "ZCPR3 and IOPs."If you followed my BIOS modifications in the first part of this article, then your BIOS does conform to this requirement.

NuKey also works with Echelon's "Z-Com" auto-install Z-System package as well as their bootable Z-System disks for Kaypro and Morrow, and Micromint's SB-180.

As distributed, NuKey is in COM file form. NUKEY.COM can create an unlimited number of personally customized IOP's for your every application.

NUKEY.COM is a ZCPR3 utility but does not have to be "installed" on most systems. The owners manual does state that NUKEY.COM may lock-up some systems. If this happens on your system, just install NUKEY.COM like you do most other ZCPR3 utilities with the Z3INS program.

When you invoke NUKEY, select "M" from the Main Menu to create a new IOP. When you are asked for a filename, press RETURN and the default file NUKEY.IOP will be created.

NUKEY.IOP is a ZCPR3 segment loaded with the LDR utility. Just type "LDR NUKEY.IOP". Now you need to select your DEFine and EXTend keys.

Defining Your Keys

When you are asked to select your DEFine key you should select a key which is not frequently used. Remember that whatever key you select will no longer be available for use under any other program except NuKey. A good choice for this character is the back-slash "\" or the tilde "~". I prefer the tilde because on my keyboard it requires the shift key to be pressed at the same time. This minimizes accidental pressing of the DEFine key.

Next, you are asked to select the EXTend key. Here again you should select a key which is not frequently used. However, whatever key you select will remain available to your other software.

After you select your DEFine and EXTend keys, NuKey is ready to run. All "virgin" copies of NUKEY.IOP (or whatever you name them) created with NUKEY.COM will ask you to select these two keys the very first time the IOP is run. After they are selected, they are stored within the IOP and are never asked for again. If you need to change them, you must create a new "virgin" copy of the IOP and start from scratch.

To assign new definitions to the keys, press the DEFine key. If the EXTend function will be used with this key, press the EXTend key next. Then, press the key you wish to re-define. Now, enter the character or characters that you wish this key to generate. The DEL, RUB or backspace keys can be used to correct your typing mistakes while entering the characters. If you want a carriage return at the end of the character string, you must enter it explicitly. It will appear as ^M on the screen. Press the DEFine key again to end the definition.

To delete an old definition, press the DEFine key. If the EXTend function was used with this key, press the EXTend key next. Now enter the character whose definition you wish to delete. The DEL or RUB keys will delete that definition.

To display the currently assigned definitions, press the DEFine key twice. A list of definitions will appear on the screen.

The assigning of new definitions, deleting of old definitions and viewing of current definitions can be performed at any time, even while you are running an application such as a word processor or spreadsheet. There is a minor drawback to this, and I personally feel it is very minor. In order to see the definitions, you have to display them on the screen. The application program does not know that NuKey is writing information to the screen and NuKey may be writing over some data being displayed by your application. NuKey is only writing over the data displayed on the screen and not over the data in the application program's buffers. This means that if you view, add or delete NuKey definitions while you are running an application program, you may have to make your application program re-write the screen after you are

done with NuKey.

Saving Definitions to Disk

There are two ways to save NuKey definitions to disk. You can invoke NUKEY.COM by just typing NUKEY<return>. This will bring-up NuKey in the menu mode. Simply follow the menu choices. The alternate method is to invoke NUKEY.COM and pass the parameters from the command line. Type NUKEY S NUKEY.IOP<return> to save the current NuKey definitions into the file called NUKEY.IOP. As with any ZCPR3 utility, the command NUKEY // will display a short help screen.

Restoring Definitions From Disk

The ZCPR3 loader program LDR.COM is used to load previously saved IOP files into memory. Just type LDR NUKEY.IOP<return>.

Deactivating NuKey Completely

As I stated in the first part of this article, in order to deactivate an IOP, a new IOP must be loaded on top of the old one. If you do not want any IOP to be running, you need to load a "dummy" IOP. This "dummy" IOP is just a series of JUMP instructions which are loaded into the IOP memory segment. They make the IOP look just like it did right after cold boot when your BIOS initialized the IOP area. You can create a DUMMY.IOP file using a debugger like DDT.

Immediately after a cold boot and before any IOP is loaded, the IOP memory segment has a dummy IOP in place. Use the ZCPR3 SHOW utility to determine the address of your IOP. (If you are using an AMPRO BIOS modified as shown in this article, then that address is EC00H.) Using the debugger, enter the following commands:

```
A0>DDT<ret>
-MEC00 F1FF 100
-G0
A0>SAVE 6 DUMMY.IOP
```

This will move the 1.5K dummy IOP image residing in memory between EC00H and F1FFH down to 100H through 6FFH. After exiting the debugger six 256 byte pages are saved to the filename DUMMY.IOP.

If you perform any changes to your BIOS which will change the location of the internal BIOS jump table, you will have to create new copies of your NUKEY.IOPs and the DUMMY.IOP. If you attempt to load a previously created IOP after BIOS changes which caused the BIOS to change size, the IOP will almost certainly cause the system to hang.

Customizing NuKey

NuKey has three built-in features which can be customized by the user. For nearly all applications, NuKey will run "right out-of-the-box" and customization will not be necessary. But for those times you need them, having the ability to change these three parameters can be a life saver.

The first parameter is called FUNDEL. Some terminals transmit their function keys as fast as their current baud rate will permit while other terminals deliberately insert a time delay between the function key characters. By changing the value of FUNDEL, NuKey can accommodate inter-character delays from 0 to 254 milliseconds. The default setting is 35 milliseconds.

The second parameter is called CTLDEL. This parameter is used to tell NuKey how many false console status "deceptions" to provide. This feature is used to "fake-out" certain application programs. Let's say you define a particular function key to send a complete command string and filename to your word processor. As soon as the word processor loads, it flushes keyboard input and some of your commands or some characters from the filename will disappear. Setting the CTLDEL parameter to some number other than zero will cause NuKey to respond "false" to a console status request after sending a carriage return so that the word processor will "think" that it has successfully flushed the keyboard buffer and no characters from the re-defined function key will be lost. The default setting is 8 deceptions. The range is from 0 to 254.

The third and final parameter is called KEYDEL. Some applications programs flush the keyboard after every character. Just as the above CTLDEL function provides for false console status after a carriage return, KEYDEL provides for false console status between individual characters. The default setting is 2 deceptions and can be changed from 0 to 254.

I personally have experimented with these values but can honestly say that the default values worked just fine on my Heath H19 and H29 terminals.

In Conclusion

Many Ampro Little Board computer owners have approached me with their dilemma of the BIOS not supporting IOP segments. Some have purchased one or more of the three Echelon IOP products only to find out that they can not be run on their systems. With the information provided in this article, a person experienced in assembly language programming and BIOS integration (MOVCPM, SYSGEN, DDT, etc.) should have no trouble implementing IOP support for their Ampro BIOS. The required source code to the BIOS is is available from Ampro Computers for a modest charge.

If you are uncomfortable making these changes yourself, just send me a bootable disk of the system you are currently running. Please indicate your system configuration (hard disk, floppy disks, BIOS options desired). I will return the disk with the new BIOS for IOP support and the DUMMY.IOP file. Please indicate whether the disk is 48 or 96 tpi.

Rick Swenton
19 Allen Street
Bristol, CT 06010

Editor's Note: In view of Rick's very generous offer, it would only be fair to include some money for his time and postage.

Now that my two main systems, the Ampro Little Board and the Heath H89 are running NuKey IOP, I don't know how I have lived without it for so long. Joe Wright's NuKey IOP program really makes your keyboard a pure joy to use!

Until now, Ampro Computer users could not take advantage of the IOP applications that other users were enjoying for some time. Now you can make those changes yourself, raise your level of understanding, and install a great keychanger on your system at the same time.

Exploring the fascinating world of Z-System encourages all kinds of learning activities by its users. The original BIOS authors and systems integrators of the late 70's and early 80's never imagined in their wildest dreams that a home computer hobbyist would be modifying and enhancing the BIOS himself! This kind of activity will probably not take place in the 16-bit world for a long time if at all. Only in our 8-bit CP/M compatible world could we have all the information at our finger tips to do whatever we want, whenever we want. Only in our 8-bit CP/M compatible world do we have support companies like Echelon, expert programmers like Richard Conn and Joe Wright, and the volunteer efforts of the Z-System user community, most especially the ZSIG organizers, Jay Sage, Bruce Morgen, Richard Jacobson and many, many others.

I wish to thank Joe Wright for his assistance in the implementation of the Ampro BIOS IOP support.

The following items are available from:

Echelon, Inc.
885 N. San Antonio Road
Los Altos, CA 94022
(415) 948-3820

ZCPR3: The Manual is the reference manual for ZCPR3 and its utilities, 351 pages, typeset, bound book. $24.00

Z-System Users' Guide is a "how-to" step-by-step tutorial and an excellent complement to "The Manual" above. Loose-leaf, 95 pages. $14.95

ZCPR3 and IOPs is the standard reference manual for implementing IOPs and operating the associated utilities. 50 page loose-leaf manual. $9.95

Three IOP programs are available:

NuKey is an advanced function key generator. $39.95

Input/Output Recorder (I/OR) redirects input/output to/from console-printer-disk file. (ZRDOS Plus required) $39.95

Print Spooler (B/Printer) background single file print spooler. (ZRDOS Plus required) $39.95

ALL THREE IOP Programs above are available for $89.95

(ZRDOS Plus, an improved replacement BDOS is $59.50 additional)

Prices are subject to change.

Sources for more information on ZCPR3 IOP's:

ECHELON, INC.
885 N. San Antonio Road
Los Altos, CA 94022
(415) 948-3820

Echelon publishes a booklet called 'ZCPR3 and IOPs' by Richard Conn. ($9.95) It is a 50 page loose-leaf bound manual which explains BIOS implementation of IOP support. This manual is a MUST for anyone seriously considering writing their own IOP's or customizing their BIOS. It is also a MUST for anyone serious about understanding IOP's.

Source for BIOS.ASM for the Little Board:

Ampro Computers
67 East Evelyn Ave.
Mountain View, CA 94041
(415) 962-0230

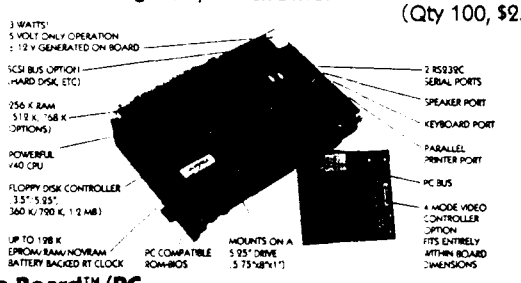
Ampro Computers offers a product identified as "1AS" which is called "Z80 BIOS and UTILITIES Source Code." I have a fairly old price list from 1985 which shows this product's cost as $79.00. It includes the BIOS.ASM file as well as the assembly language listings of all the Ampro Little Board Utilities.

Registered Trademarks: ZCPR3, I/OR, B/Printer, Z-System, NuKey, Echelon; Little Board, Ampro Computers; CP/M, DDT, Digital Research; Z80, Zilog; H89, H19, H29, Heath/Zenith. ■

Reliable, Cost Effective Solutions for Computerization

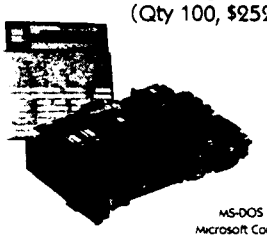
A Motherboard and 4 Expansion Cards in the Space of a Half-Height 5-1/4" Disk Drive!

from **$329**
(Qty 100, $252)



Little Board™/PC
World's smallest PC — and CMOS too!

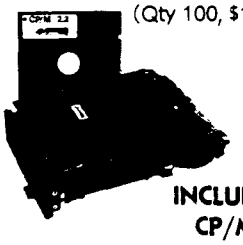
$329
(Qty 100, $252)



MS-DOS Microsoft Corp.

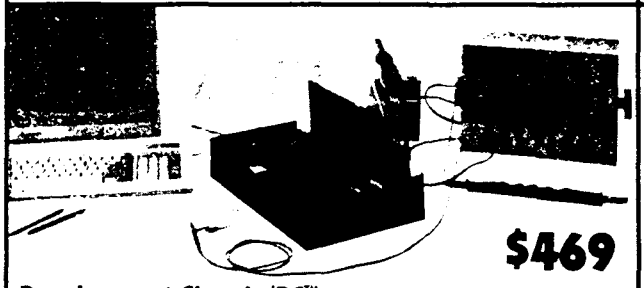
Little Board™/186
High performance single board MS-DOS system.

$159
(Qty 100, $124)



INCLUDES CP/M!™ DRI

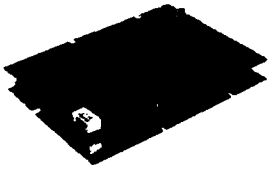
Little Board™
World's least expensive single board system.



$469

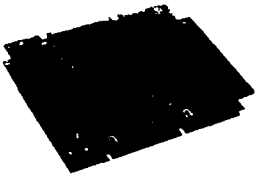
Development Chassis/PC™
"Known Good" PC bus project development environment for Little Board/PC (not included).

$99



Project Board/186™
Prototype adapter for 80186 based projects and products.

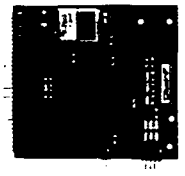
$89



Project Board/80™
Prototype adapter for Z80 based projects and products.

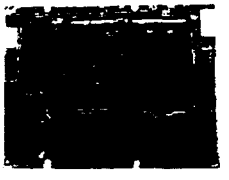
Z80 TM ZILOG

$159



CMOS Video Controller
4-mode CMOS video controller for Little Board/PC.

from **$179**

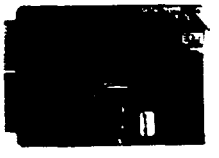


SCSI Memory Controller
SCSI controller for fixed and removable volatile and non-volatile semiconductor memory.

from **$149**

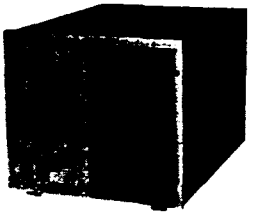
Expansion/186™
Multi-function expansion for Little Board/186. I/O, Serial, RAM, and Math Options.

$129



SCSI/IOP™
STD bus I/O expansion adapter for any SCSI host system.

from **$495**



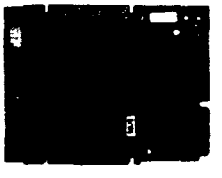
Bookshelf™
System Modules

- PC compatible Little Board/PC systems.
- Single and Multi-User Little Board/186 systems.
- Little Board/Plus CP/M systems.
- SCSI disk and tape expansion modules.
- Floppy drive expansion modules.

$395

Concurrent PC-DOS™ DRI
Multi-user, multi-tasking operating system for Little Board/186 supports up to four users.

$99



SCSI Z80 Host Adapter
SCSI host adapter for any Z80 system.
Plugs into Z80 socket.

Distributors • Argentina: Factorial, S.A. 41-0018 • **Australia:** Current Solutions (03) 227-5959 • **Brazil:** Computadores Compuleader (41) 262-4866 • **Canada:** Tri-M 604/438-0028 • **Denmark:** Danbit (03) 66 20 20 • **Finland:** Symmetric OY 358-0-585-322 • **France:** Egal Plus (1) 4502-1800 • **Germany, West:** IST-Elektronik Vertribes GmbH 089-61[illegible] • **Israel:** Alpha Terminals, Ltd. (03) 49-16-95 • **Spain:** Hardware & Software 204-2099 • **Sweden:** AB Akta (08) 54-20-20 • **UK:** Ambar Systems, Ltd. 0296 435511 • **USA:** Contact Ampro

AMPRO
COMPUTERS, INCORPORATED

67 East Evelyn Avenue • Post Office Box 390427, Mountain View, CA 94039 • (415) 962-0230 • TLX 49[illegible]

The ZCPR3 Corner

by Jay Sage

After the tremendous effort I put into producing my last column and ZCPR33 before leaving on summer vacation, I was really burned out. I hardly touched the computer or even thought about programming all summer. This was quite remarkable, since for months I had practically lived in front of that computer screen and thought I would have severe withdrawal symptoms if I didn't get my usual daily fix. I was glad the last issue of TCJ came out a bit behind schedule, because three weeks ago I really couldn't think of anything to write about for this column.

Then the bug struck again, as I knew it would eventually. My mind exploded with ideas, and, as in the past, I again have far more things to talk about than I will have the energy or time to get down on paper before this article has to be sent in.

Echelon and I

I practically fainted when I saw the byline on my last column, and I wonder if both Echelon and my real employer did too! The byline read: "Jay Sage, Echelon, Inc." I guess Art Carlson misinterpreted my statement that I had joined the Echelon team. In no way am I an employee or official representative of Echelon. In real life I am still a physicist at MIT doing research on special analog(!!) devices and circuits for image processing and neural-like computing. Digital computing is still essentially a hobby.

The members of 'The Echelon Team' are independent programmers who cooperate with Echelon and each other to advance 8-bit, CP/M-compatible computing. Other members of that team, to mention only a few, are Bridger Mitchell, author of DateStamper, BackGrounder, and JetFind; Joe Wright, author of the auto-install Z-System and the BIOSs of most of the recent popular 8-bit computers (Ampro, SB-180, On!); and Al Hawley, sysop of Z-Node #2 (the only one to sign up for a node before I did) and author of the REVAS disassembler and the new ZAS.

Supporting 8-Bit Software Developers

Though team members have no personal stake in Echelon as employees or owners, they do benefit to some extent from royalty payments for their software that is sold. Nevertheless, before turning to the technical material for this issue, I would like to make an unabashed plea to all of you to support Echelon and the small number of other companies still developing 8-bit software by purchasing their products. They are the only hope for the sustained vitality of our 8-bit world. If we don't buy the products they offer, the creative programmers who have not already done so will have no choice but to abandon Z80 programming in favor of the more lucrative IBM-compatible market. Check the ads in The Computer Journal, and support the companies that advertise there. Do not make regular use of illegitimate copies of their software; buy your own.

Unfortunately, no one is getting rich on 8-bit software. I did not keep a record of the time I spent on ZCPR33, so I cannot calculate accurately the effective hourly rate, but a rough estimate indicates that babysitting would probably have been a better way to make money. In fact, my greatest financial reward probably came from the money **not** spent on babysitters as a result of staying home programming instead of going out!

While reflecting on these issues, I thought I had a brilliant idea —put together a transportable computer and actually hire myself out as a babysitter. That way I could get paid not only for the commercial products like ZCPR33 but for all the public-domain programs as well. Unfortunately, on more careful consideration, I noted some serious defects in this scheme. First of all, how many people would hire me to sit night after night until 2 or 4 in the morning? And who would want a babysitter who wouldn't notice the house burning down until the power went out on his computer screen?

Basically, I think it is fair to say that all of us (even those, like Joe Wright, who depend on it for their livelihood) are in the 8-bit programming business because we love it. Even for the non-professionals, like me, there are reasons why some financial compensation is important. While we may derive enormous enjoyment and excitement from programming, our families do not, but if it brings in a little extra spending money that the entire family can share, they are much more tolerant of the long hours spent in front of the CRT or on the telephone helping users with problems.

New Commercial Z-System Software

As a transition from my 'soap box' remarks above, I would like to begin the technical discussion with a review of some exciting developments in commercial Z-System software.

WordStar Release 4

The most exciting development in a long time is the appearance from MicroPro of WordStar Release 4, CP/M Edition. As far as I can remember, this is the first new CP/M product from a major software house since Turbo Pascal version 3 came out several years ago, and it is the **only** product **ever** from a major vendor that supports Z-System. I am thrilled at the official recognition this bestows on Z-System.

For the season's kickoff meeting of the Boston Computer Society's CP/M Computers Group, we had representatives from MicroPro to introduce the new product. As excited as I was about Release 4, I was sure that this would be the end of the line, so I was quite surprised when the representatives talked about a release 5 for CP/M as well as for MS-DOS.

MicroPro speaks of itself now as the 'New' MicroPro, and, indeed, they sounded like a new MicroPro. They are extremely solicitous of user suggestions. Their upgrade policy is very generous (they'll happily accept the serial number from any older WordStar or NewWord), and the upgrade price of $89 for individuals and $79 for clubs is very attractive. Echelon's special offer at $195 for those who did not own WordStar or NewWord

before is also quite reasonable for such a high quality product. I personally have not made much use of WordStar in the past, preferring my roll-my-own, do-it-my-way PMATE text editor, but I have already placed my order for WS4, if for no other reason than to show my support to MicroPro and to encourage them to stick with us 8-bitters.

Frankly, I am also quite eager to explore WS4's new Z features. Since my copy has not arrived yet, I cannot give you a first-hand report, but what I have been hearing from others is extremely positive. It apparently knows about the command search path and named directories of ZCPR3 and can run as a true shell. The documentation included with WordStar 4 is extraordinary, the equivalent at least of the customization package that the 'old' MicroPro used to charge an extra $500 for! No longer will the hobbyists have to ferret out all the patch points for the program (though I am sure there will be plenty of areas, nevertheless, to keep us entertained).

ZAS/ZLINK

Another exciting development is release 3 of ZAS/ZLINK, Echelon's assembler/linker (and librarian) package. As many of you may know, most serious programmers — Echelon team members included — have had little but scorn for ZAS in the past. It was a strange and unreliable assembler.

But Echelon has now really made good on ZAS. They did it right this time. They did not go back to the original author and try once again to get him to fix it; instead they brought in the highly competent Al Hawley. Though I am sure it is still not perfect (what program is), it has correctly assembled all correct code that I have fed to it. And gone are its former irritating and unique idiosyncrasies, like square brackets instead of parentheses in arithmetic expressions. ZAS will now handle just about any code written in some semblance of standard assembly language. It supports a rich set of pseudo-ops, making it tolerant of common variants.

ZAS and ZLINK are also at long last honest Z-System tools, as befits an Echelon product. They recognize named-directory references for all files, and they communicate with the Z3 environment and message buffers. With an appropriate editor it is possible to build a code development system like that in Turbo Pascal. When ZAS encounters errors in assembly, it stores enough information about the first error in the environment that an editor can automatically locate the line with the error. Thus one can make an alias that bounces between the assembler and the editor in very convenient fashion. As an added bonus, the Echelon version of ZAS includes, as the one from Mitek always did, the option to generate in-line assembly code for Turbo Pascal.

If you have an older version of ZAS/ZLINK, you should definitely order the upgrade, priced at just $20 for the software alone or $30 with an updated manual. If for some reason you do not want to do that, you should at least destroy any copies of the older versions that you have lying around.

New ZCOM

Now I would like to move forward in time and talk about a product that is in the works (though with the delay between when I write these columns and when they reach readers, it may be on sale by the time you read this). This product is New ZCOM, or NZCOM, Joe Wright's utterly spectacular follow-on to ZCOM. This product will make manually installed Z-Systems obsolete, because manually installed systems will now offer less performance than NZCOM systems.

First some history. In the summer of 1984 Joe Wright was reflecting on the difficulty of converting stock CP/M 2.2 systems to ZCPR3 and wondering if an automatic method could not be developed. Rick Conn apparently opined that such a thing would not be possible. Joe did not ask me (we did not know each other at the time), but if he had, I would have told him the same thing — impossible! You know the old marines' saying: the difficult we do immediately; the impossible takes a little longer. Well, Joe didn't even take very long to do the impossible. In a matter of weeks he had a fully working version.

ZCOM had two drawbacks compared to a manually installed Z-System. First, it required an extra 0.5K of overhead. Secondly, and ultimately more seriously, it was not flexible. One had to accept a standard configuration. There was no choice of command processor options, number of named directories, size of RCP, and so on. Thus, ZCOM was the Z-System of choice only when a manual system could not be made, either for lack of skill or lack of BIOS source code.

Since the new computer I bought at work in 1984, a WaveMate Bullet, to my chagrin did not come with source for the BIOS, I tried briefly to figure out how to get ZCPR3 installed without BIOS modifications. I came up with an approach that might have worked, but before I got very far with its development, Echelon announced Z3-DOT-COM and, shortly thereafter, ZCOM. I bought them right away. After a short time, I figured out how they worked (and was amazed at Joe's cleverness). Then I began to implement the modifications I described in my last two columns, including ways to switch between different auto-install systems from within alias scripts and while inside shells.

Later, after I became acquainted with Joe, I told him about

my ideas for an enhanced ZCOM. It seemed, however, that he was much too busy at the time with other products to give any attention to ZCOM, so I decided that my next project after ZCPR33 would be a program that I tentatively called Dyna-Z. This would be a dynamic Z-System, an operating system whose configuration could be changed on the fly.

Dyna-Z would be useful in several ways. One could normally run a system with all the standard ZCPR3 modules except an IOP (input/output package), giving one a good set of ZCPR3 features for normal operations. When one wanted to make use of an IOP, like Joe Wright's superb NuKey keyboard redefiner program, a load alias would first check to see if an IOP space was available in the system. If so, NuKey would be loaded. If not, the alias would switch the operating system to one that did include an IOP and then load NuKey. One would sacrifice the 1.5K of TPA (transient program area — the memory available for a program to work in) only while one needed the benefits of NuKey.

On the other hand, when one invoked a command such as WordStar 4 or a spreadsheet that required a large TPA, an alias for that command would first switch to a Z-System with no IOP or RCP, increasing the TPA by 3.5K compared to the full Z-System. One could even drop the FCP (0.5K) and/or the NDR (0.5K typically). When the memory-hungry program was finished running, the remainder of the alias script would reload the standard version of Z-System. One would hardly know that the operating system had been different while WordStar or the spreadsheet was running.

Thus Dyna-Z would not only overcome the major disadvantage of ZCOM but would also overcome the only intrinsic disadvantage of ZCPR3 — the loss of TPA space. A minimum Z-System would use only 0.75K plus the autoinstall overhead (reduced to a mere 0.25K), a very small sacrifice for all the benefits that Z-System offered. And in a real pinch for TPA, one could even drop out of Z-System temporarily and run standard CP/M with its maximum possible TPA. Using SUBMIT, even this could be done with a script!

I was delighted this summer when Joe Wright called on the phone to discuss his plans for New ZCOM. It was the first I knew that he had taken up the project, and I found that he had already made great progress. Many of the features I had planned for Dyna-Z were already implemented, and Joe was eager to incorporate the rest. Our partnership was born! And it is a perfect partnership for me — Joe is doing all the work.

I would have been excited enough just to see the Dyna-Z features in NZCOM, but Joe has done much more than that. He has made the process of building a system of **your** choice about as easy as it could possibly be. You simply edit NZBAS.LIB, a text file describing the system configuration you want, and assemble all the individual components (CCP, DOS, RCP, FCP, etc.) to Microsoft-format REL files. NZCOM.COM then generates a system auto-loader program for you automatically. It will even allow you to clone an existing system, accomplishing automatically all the complex processes I described in my last two columns.

JetFind

Now I would like to switch back in time and describe a program that has been around for a while already but has not had the publicity I think it deserves. It suddenly struck me the other day just how often and in how many ways I use JetFind yet how few people probably are aware of its existence.

JetFind, by Bridger Mitchell, is basically a text finding program. It is something like Irv Hoff's publicly released FIND.COM, which can search through a file for a specified text pattern. But it goes orders of magnitude beyond that.

To begin with, I should explain that JetFind operates in either of two modes: interactive or command mode. In interactive mode, the program is invoked alone, with no command tail. It will then prompt the user sequentially for each piece of information it needs. The user can then live inside JetFind, performing one search after another until he issues an exit command. Alternatively, a single search operation can be carried out by including all the necessary information on the command line.

Now for the capabilities of JetFind. First of all, JetFind is not limited to searching the text for only a single pattern at a time. It can search for multiple patterns, and each one can be either a simple text string or a regular expression (a UNIX concept). Let's take a simple case first. Suppose you want to find all lines that contain either "Smith" or "Jones". In interactive mode you would enter the patterns one at a time in response to the prompt. Just hitting carriage return would end pattern input. In command mode, you would enter for the search pattern the following expression:

```
SMITH|JONES
```

The special character '|' represents 'or'. From command mode, of course, one cannot distinguish upper and lower case. To do that you must use interactive mode.

Now let's consider a more complex search that would make use of a regular expression. Suppose we want to find all labels in an assembly language program. We could use the following regular expression:

```
[A-Z][A-Z0-9]*:
```

The first term in brackets means a character from the set of letters ranging from 'A' to 'Z'. The second term in brackets is the set including the digits from '0' to '9' also, i.e., an alphanumeric character. The asterisk means that the previous character specification may occur any number of times, including zero times (a '+' would require at least one occurrence). Finally the colon on the end represents the ':' character

If labels have to be at the left margin, we could use the regular expression

```
^[A-Z][A-Z0-9]*:
```

A caret at the beginning of an expression indicates the beginning of a line. A mode control specification (explained later) can tell JetFind whether or not to ignore case. If other characters are allowed in labels, they could be listed inside the brackets as well.

There is not enough space here to give a complete description of JetFind's regular expression syntax. Suffice it to say that it can perform just about any search you would ever want to do.

A second major feature of JetFind is that it is not limited to searching single files; you can specify whole collections of files to search. You can give a list of ambiguous file specifications both for files to **include** in the search and files to **exclude** from the search. These files can all come from a single directory, or files from many directories can be included. For example, the file list

```
TEXT:*.D?C ~TEXT:A*.*
```

indicates all files in the named directory TEXT (yes, JetFind is fully ZCPR3-compatible) with a file type of D?C but not (that is

the meaning of the '~' prefix) files whose name begins with 'A'. Note the '?' in the file type. The intention here is to search all DOC files. By including the '?' for the middle letter, files of type DQC (squeezed DOC files) and DZC (crunched DOC files) will be included as well. JetFind automatically uncompresses both squeezed and crunched files as it searches.

Moreover, JetFind is not limited to searching individual files. It can even search through members of libraries. If the first file name in the list of files is an LBR file, then the rest of the list is taken as a specification of the members of that library to be included in or excluded from the search. As with individual files, these files can be unsqueezed or uncrunched on the fly.

As if this were not enough, JetFind has about a dozen mode control options that define how it performs the search and what it does with the text identified by the search. Here are descriptions of just a few.

```
C     This option just counts and displays the number of matches without
      showing the matching lines of text.

N     The lines containing matching text will be numbered in the listing
      to make it easy to find them with an editor.

Rmn   This specifies a display region ('m' and 'n' are each digits from 0
      to 9).  For each line containing a match to one of the patterns, the
      previous 'm' lines and following 'n' lines will also be included in
      the display to provide context.

I     Case will be ignored so that 'a' and 'A' will be considered to
      match.

B     Begin displaying the text as soon as the first match has been found.

V     Reverse the test and display only lines that do not match any of the
      specified patterns.

T     Type the file, extracting it from a library and/or uncompressing it
      as required.
```

With these and other options not listed above, JetFind can be made to perform many tasks besides searching, such as typing files, extracting files from libraries, splitting off parts of files, and displaying files in a directory or library.

We're not finished yet! JetFind also supports full input/output redirection. The output text that is shown on the screen can additionally be saved to a file, either in a new file or appended to an existing file. The set of patterns to search for can also come from a file. Thus we could have the command

```
JETFIND -WN <ASM:LABEL.EXP ZF*.Z80 >LABELS.LST
```

This would search through all the ZFILER source code (ZF*.Z80) for the regular expressions contained in the file LABEL.EXP in directory ASM. The search would require whole-word matches ('W') and include line numbers with the matching lines ('N'). The output would be displayed on the screen and written to a new file called LABELS.LST in the current directory.

One final comment. JetFind does its work at incredible speed. Bridger Mitchell is an absolute master at wringing performance out of the operating system, using all kinds of tricks to speed up file operations. Hence the 'Jet' in the name. JetFind is available from Echelon or Echelon dealers for just $49.

New ZSIG Programs

Now I would like to turn to some exciting new ZSIG programs that have been released or are under development at this time.

LLDR — Library Loader

Paul Pomerleau — already well known to the community as the author of such widely used programs as VERROR (visual error handler), BALIAS (an alias editor), AFIND (alias finder), and the commercial LZED (little Z editor) — has released LLDR, a version of LDR with library support.

This program completely replaces LDR, the standard module loader program used to load new ENV, Z3T, RCP, FCP, NDR, and IOP operating system code segments. It does everything LDR did but adds one new, very important feature. It can load the modules from a library.

ENV and Z3T modules in particular are very short (one or two records), and it was very inefficient use of disk and directory space to have them sitting around as individual files. Now all the system files can be collected together in a single LBR file (or several, if you prefer). LLDR's syntax can be expressed in general form as follows:

```
LLDR [library[.LBR],]<list of modules to load>
```

If the optional library specification is omitted, then it performs just like LDR. If all the system modules are placed in a file called, for example, SYS.LBR, then one might invoke LLDR (renamed to LDR) as follows:

```
LDR SYS,SYS.ENV,DIMVIDEO.Z3T,DEBUG.RCP,SYS.FCP,NUKEY.IOP
```

Besides the saving in file space and directory entries, there is another nice side benefit of using LLDR in this way — it is much faster. Since only one file (SYS.LBR) has to be opened by the operating system, there is only one directory search, and loading the entire collection of modules takes little more time than loading a single one did with the old LDR program. Thanks, Paul, for another nice program.

SALIAS — Screen Alias Editor

When I released VALIAS (visual alias editor) a couple of years ago, I wrote in the documentation that someone should please extend it to full-screen operation (it only supported insertion and deletion of complete lines). Paul Pomerleau's BALIAS allowed full WordStar-like editing of alias scripts, but it treated the entire multiple-command script as a single line. I much preferred the structured presentation of VALIAS, with each command on its own line. I suggested that VALIAS should be extended to automatically indent the lines to show the nesting of flow-control commands.

It has been a long wait, but finally the wait is over. Rob Friefeld, from the Los Angeles area (contact him on Al Hawley's Z-Node #2), has released SALIAS (screen alias editor), and what a beauty it is. You will no longer find VALIAS on any of my disks!

With SALIAS, alias scripts are displayed and edited rather as if they were WordStar text files. Each individual command is displayed on its own line, except that long lines can be continued on the next line by entering a line-continuation character (control-p followed by '+') at the beginning of the continuation line.

SALIAS works in two basic modes. One is the command mode, like that in VALIAS. In command mode, the status line at the bottom of the screen displays the following prompt:

```
CMD (Clear Edit Format Indent Load Mode Print Rename Save
     Undo eXit)
```

Entering 'C' will clear the script editing area. 'E' will enter full screen editing mode. 'F' will reformat the script, converting it to upper case and placing each command on its own line, even if the user entered lines containing multiple commands separated by semicolons. The 'I' command is similar except that it indents th

lines by three extra spaces for each level of IF nesting. Thus a script might appear as follows:

```
IF EQ $1 //
   OR NU $1
   ECHO SYNTAX IS ...
   ELSE
   IF NU $2
      COMMAND FOR ONE FILE SPEC
      ELSE
      COMMAND FOR TWO FILE SPECS
   FI
FI
```

This format makes it very easy to see the relationship between the flow tests and to detect missing FIs.

Entry of complex conditional aliases is greatly facilitated by the use of the tab key to indent the script as it is entered. The 'I' command will reindent the display according to the actual flow levels, even if the user made a mistake. The spaces (tabs) used to format the display are not part of the actual alias script. However, leading spaces entered by the user (to invoke extended command processor operation, for example) will be included in the script and are displayed in addition to those added by the indenter. The 'F' command will show the real contents of the script, automatically deleting the indentation tabs.

The 'L'oad command tells SALIAS to clear any existing script and to load an alias file for editing. If such an alias already exists, it will be read in. Otherwise it will be created. 'M' allows one to specify either a normal alias or a VALIAS-style recursive alias, one that clears the entire multiple command line buffer before it is run. In an earlier TCJ column I described how one would use this kind of alias for certain kinds of recursion.

The 'P'rint command will send the screen display of the alias script to the printer. 'R' will let one change the name assigned to the script being edited so that a script can be read in from one alias and written out to a new alias. 'S' saves the current script in a file with the current name. Undo will ignore any editing that has been performed on the alias and let the user start over with a fresh copy of what was read in from the file originally. 'X' will terminate operation of SALIAS (without any prompting).

SALIAS has an alternative mode of operation entirely from within the interactive edit mode. All of the functions that can be performed by commands in command mode, and some others as well, can be performed using control-character sequences directly from edit mode. These commands are as follows. In all cases, the first character (for example, ^K) is a control character. The second character can be a control character or a regular character in either upper or lower case.

```
^KS   Save the alias under the current file name.

^KD   Done editing -- save the file and then clear
      the edit buffer.

^KX   Exit from SALIAS after saving the file.

^KN   Assign a new name to the script.

^KQ   Quit without saving the script.

^KR   Read in another alias script and append it to
      the commands currently in the edit buffer. This
      is very convenient for combining scripts from
      multiple aliases.

^KF   Reformat the alias, listing each command on its
      own line, removing any flow-control indentation,
      and converting to upper case.

^KI   Indent the alias display to show flow control
      nesting.

^KU   Undo changes that have been made to the script.

^KP   Print the script.

^KM   Toggle the mode of the alias between normal and
      recursive.
```

The editing commands follow the familiar WordStar pattern. Even ^QS (move to beginning of line), ^QD (move to end of line), and ^QY (delete to end of line) are recognized. The special form ^QZ will zap (clear) the entire script so you can start over. ^R and ^C move to the first and last lines of the script, respectively. ^QF allows searching for strings, and ^QA allows search-and-replace. ^L will repeat the last search or search-and-replace. ^V toggles between insert and overtype modes.

SALIAS is available in two versions. The longer version has imbedded help information that can be called up using ^J. In the shorter version, the help information has been omitted, and ^J instead is used to toggle the cursor between the beginning and the end of the current line.

I should mention a few other features of SALIAS. There is an additional status line at the top of the screen. It shows the name and version number of the program, the type of alias (normal or recursive), the number of characters free, and the current name for the alias.

The free-character value is calculated by subtracting the number of characters presently in the script from the number of characters allowed in the multiple command line buffer. This computation is not infallible. There are some parameter expressions, such as $D, that take up less room when expanded, so it is possible that SALIAS will refuse to let you save an alias that it thinks it is too long when in fact it is not. More likely, however, is that you will save an alias that has few enough characters for SALIAS to accept it but will become too long when the parameters are expanded. And even if this does not happen, you can run into trouble when the alias itself appears in a multiple command line expression, and not-yet-executed commands have to be appended to the alias script. These subtleties aside, having the display of free characters is helpful.

To summarize, in my opinion SALIAS makes all previous alias editors obsolete. You should be sure to pick it up from your local neighborhood Z-Node. If you do not have one, then join NAOG/ZSIG and order a disk from them.

VLU — Visual Library Utility

VLU is a utility we have all been wishing for — a screen-oriented library management utility. Its author is Michal Carson of Oklahoma City, OK. I originally suggested a library shell, like ZFILER but working on the contents of a library, but Michal pointed out that there is really no need for the visual library utility to be a shell, since shell functions like macros performed on pointed-to files will generally not be applicable to library members. So he built VLU as a non-shell utility that interfaces closely with ZFILER.

VLU can be invoked in two ways. Without any file name specified on the command line, it tries to open a library whose name is the same as the file name stored in the Z3ENV system file 2. This system file is where ZFILER keeps the name of the file it is currently pointing to. If one has a ZFILER macro called, for example, 'V' with the simple script 'VLU', invoking this macro

while pointing to a library (or to another file with the same name as an LBR file in that directory), the library will be opened for work by VLU. Alternatively, one can specify the name of the file on the command line, in which case this name takes precedence over any name in system file 2.

Once VLU is running, the display contains two fields. The upper field is like that in ZFILER. It displays the names of the files in the currently logged directory. The lower field is similar in appearance but shows the names of the member files in an open library file. If no library file is currently open, this field is blank.

As in ZFILER, there is a file pointer that can be moved around using standard control characters or, if defined, cursor keys. In VLU, the escape character toggles the cursor between the upper (file) and lower (LBR member) fields of file names.

Many operations can be performed on either set of files. Files can be tagged and untagged, and two wildcard tagging functions are provided. 'GT' group tags all files, while 'W' allows a wildcard file specification to determine which files get tagged. Individual files or groups of tagged files can be viewed, crunched (but not squeezed), or uncompressed (either uncrunched or unsqueezed, as appropriate). The 'F' command will show the size of an individual file or library member at the cursor. For library members, the size is shown in records as well as kilobytes. Additionally, tagged individual files can be deleted or renamed, and tagged library members can be extracted, with or without decompression.

The special 'GB' or Group-Build function of VLU allows tagged individual files to be built into a new library. As I write this article, the details of this feature have not been fully worked out, but it will be possible for the user to indicate which of the tagged files should be crunched according to an automatic algorithm (which will crunch them only if that results in a smaller file) and which should be put into the library in their existing form.

Single individual files will accept the command 'O' to open a library with that name if it exists and the command 'C' to close the currently open library. They can also be renamed using the 'R' command.

There are several miscellaneous commands. '/' will toggle between a built-in help display and the file name display. 'X' is used to exit from VLU; 'J' is used to jump to a file; '*' is used to retag files that were tagged before some group operation was performed; and 'Q' refreshes the screen display.

Future versions of VLU will include some features not yet in the present one. A printing capability will be added to complement the 'view' functions. Presently, VLU cannot add files to an already existing library, though it allows the user to specify the number of elements to accommodate in the library directory so that another tool such as LPUT can be used to add more files later. VLU has an 'L' command to log into a new directory. By opening a library in one directory and then changing to another, one can extract files to a directory other than the one containing the library. At present, however, the 'L' command does not recognize a file mask as ZFILER does to restrict the files included in the display. Even in its initial release form VLU is a very welcome addition to the toolbox of Z utilities, and I extend thanks from all of us to Michal Carson.

Subject for Next Time

As I promised last time, this column has taken a less technical tack, though I feel that it has covered important and valuable material. For next time, however, I expect to return to a more detailed technical discussion. In the past few weeks, I took up the task of rewriting and expanding ZEX, the ZCPR3 in-memory batch execution utility. This has been my first detailed look at a program of this type — a resident system extension (RSX), a program that takes over and replaces functions of the operating system, in this case the BIOS. I have learned a great deal and have made what I think are some spectacular improvements to the way ZEX works and additions to what it can do. Next time I will share with you what I have been doing and what I have learned. ■

Z-Systems Computer Festival

December 5, 1987

The Connecicut CP/M Users' Group will present its first Z-Systems Computer Festival at Mather Hall, Trinity College, in Hartford, Connecticut from 1 to 6 p.m., on December 5.

Featured speaker will be noted author Jay Sage, who will speak on the uses and future of the ZCPR operating system for CP/M computers. There will also be workshops on Z-System, Public Domain and Free Software, and demonstrations of state of the art CP/M computer hardware and software.

Admission is free and open to the public.

For further information, contact:

Lee Bradley, President
Connecticut CP/M Users' Group
(203) 666-3139

32000 Hacker's Language

by Neil R. Koozer

Here's a follow-up to my previous comments alluding to my 32HL system (32000 Hacker's Language). Before getting down to the details of what I'm up to, we must consider why we want to use one of the fatter processors. If we're after speed, we may gain enough by fixing our Z80 software. For example, my Z80-to-32000 cross assembler performs 51 (fifty-one) times as fast as the commercial version from 2500AD. But how fast can the 32016 do the same task if I use high quality 32000 programming? The answer is the same speed as the Z80 if we assume equal clock speeds and disallow in-memory source code. My previously stated goal was to break the chicken-and-egg barrier, but that's not the whole story because there must be a reason to do it. Here are some of my reasons for wanting to use the 32000.

1. The set of generalized 32-bit registers and floating point registers, the 2-general-operand instruction set, and the unfragmented addresses make programming very easy and allows almost any software to epitomize the KISS principal. For example the statement

```
MULL var1,var2
```

does a double precision floating point multiply of var1 by var2. If the operands are in memory, then no registers are used. The statement

```
MOVST U
```

moves a string while translating characters and checking for a specified ending delimiter.

2. If I ever do some computing instead of building my computers, my primary interest is in scientific computations. My 32000 transcendental routines do the Savage in 3.02 sec with an error of 1.6e-9. Because of the compactness of the code, it's no bother at all to make complex numbers a built-in data type and to include complex transcendentals.

3. The linear address space allows all software development tools to be integrated into the operating system in a coherent way, allowing much more efficiency on the human side of development activities. It also allows in-memory editors, large arrays, etc.

When I first happened onto a windfall 32016 board, I only wanted to do some math, and I wrote a FORTH-83 system to serve as a driver. I originally wrote the Forth system because I thought it was too hard to write a normal compiler. The first time I ran the Byte sieve, it took 43 seconds. After doing all the reasonable things that I could to speed it up, it still took 9.6 seconds. I started letting many keywords compile native code, and then I realized I knew how to write a compiler if I could just get rid of the Forth syntax. All it needed was an expression parser and some BASIC-like keywords that would compile the run-time code. I first used a recursive descent parser, and the Sieve time dropped to about 4 seconds. That parser was too dumb; I couldn't easily remove the unnecessary pushes and pops or take advantage of some of the 32000 instructions. For example, a smart parser would translate

```
      IF int1 < int2 ...
into

      CMPD int1,int2
      BGE xxx           ;xxx = disp. to else clause
```

which doesn't even use any registers. I wrote a more flexible parser that produces code that cannot be improved with hand assembly (assuming fairly simple statements like those in the Sieve program). The compiled BASIC version of the sieve program occupies about 114 bytes and runs in 1.74 seconds (32016, 7.16 Mhz). I couldn't measure the compile time. This rendition of the Sieve uses a call to the BASIC STRING$() function to fill the array because in this system strings and byte arrays are synonymous.

At about this point my experimental system reached a dead end because it was not self generating, and the commercial cross assembler had too many problems. Before continuing I had to write a new cross assembler. Before I could do this I had to write a new Z80 assembler because all the Z80 assemblers that I had were too slow. I'm replacing the Forth system from the ground up, starting with a native 32000 assembler as the core. This assembler is to grow into an operating system and high level language compiler, but no matter what it becomes, it will always also be an assembler so that it can compile itself in one quick pass at any stage of its development.

The philosophy of usage of this system is different from that of most commercial compilers. It is not for the development of commercial products that are to be distributed in opaque form. It is for the user who wants to develop his own programs, expecially in applications where there is no such thing as the final version of the program. These might include scientific computations or laboratory research experiments. It is also for the user who would like to develop the system itself into a different kind of operating system or language compiler. It is expected that the compiler will always be in the computer and that the programs will always exist in source form. Programs will view the system as something like a module in Modula2 and will call system resources with normal function or procedure calls. Programs can be stored and loaded in compiled form as long as the system is not changed. If the system is changed, then any program to be loaded is compiled from its source. The user is completely free to repair or embellish any part of the system at any time. This is a vital feature because there is no such thing as a program that is good enough to be immune to shortcommings that demand correction.

Another philosophical point of this system has to do with memory usage. Because of a long history of cramped addressable memory, it has become a universal custom to use disks as memory emulators. We even have RAM disks that use memory to emulate the memory-emulator. Now that we have 68000 and 32000 processors, we can use the novel approch of using memory as memory and using the disks to store things.

Another point is clutter reduction. A long time ago I craved to get my fingers on a relocating assembler and linker, but now after some seasoning, I have found that the OBJ files and other intermediate files produced by compilers and assemblers constitute an obstacle course that impedes progress. It forces extra activities to keep track of the files. It requires constant editing of global lists. It requires extra commands to do the linking, etc. It requires more editing of batch files. It forces more disk changes due to full disks. It also slows down the actual compilation. My Z80-to-32000 cross assembler compiles an executable file twice as fast as the fastest linker that I have ever used, and many times as fast as the linkers that do enough things to be usable. It takes the same amount of time that the CP/M LOAD utility takes to translate an equivalent HEX file to a COM file. Thus, my assemblers and my 32HL system do not generate any intermediate files; they use INCLUDE to accomplish linking at the source level. My Z80 assembler retains the option to make a HEX file and retains the option to perform two passes. My cross assembler eliminates the capability to perform two passes or to make a HEX file. My 32HL system further eliminates the capability to make a PRN file (I'm hoping my proposed debugging scheme won't need the PRN file). I have found myself getting into the habit of not saving the COM file because it takes longer to type the commands to save the COM file on two floppies than it takes to assemble a new one later. Modula2 fixes some of the bad syntax of Pascal and allows some ways around the roadblocks, but the excruciatingly huge number of files to keep track of renders the thing unusable to me (one programmer per project). It's a good emulation of the cassette days when you had to keep track of files without a directory.

One of my ideas for the file system is to let the directory be compatible with the symbol table so that the expression parser can access disk files the same as other variables. The statement

```
A = B(I) + C
```

would not care if B is an array in memory or if it is a disk file. No opening or closing syntax is needed, no functions are needed, and none of that connect/disconnect stuff that Modula2 uses. The parser would check the types and compile appropriate code just as it does for any statement. A file is created by preceding any variable declaration or procedure declaration with the word FILE, for example

```
FILE REAL name1(0)
FILE PROC name2(REAL x y INT j).....
```

The file name1 is an array of real numbers that starts out with 0 length, while name2 becomes a compiled program stored on disk. To execute it type

```
name2(param, param, param)
```

The parentheses are optional to make it more like typing a conventional command line. Nothing like ARGC or ARGV will be needed. In C terminology, FILE is a class along with STATIC, AUTO, etc. The syntax of my language, shown in Figure 1, is a mix of BASIC, C, Modula2, & assembly.

Figure 1: HL32 syntax.

```
Any statement can be assembly or high-level.
Free form source--any number of statements can appear on a line.
Keywords are all-caps because of large number of mnemonics.
User defined identifiers must start with a lower case letter.
User defined identifiers arbitrary length, all characters
    significant.
Both assignment and equality comparison use '=' sign.
No punctuation is used between statements or after labels.
Variable definitions resemble those of C except no commas or
    semicolons used.
Conditional compilation like that of C.
Operator precedence like that of BASIC.
Complex numbers a built-in data type (with complex
    transcendentals).
Parameter lists are checked and resemble those of Modula2 except
    commas and semicolons are not used.
Control structures like Modula2, except each has a different
    ending word: IF..ENDIF, WHILE..WEND, FOR..NEXT, etc.
No braces or BEGIN..END needed for multiple statements in IF,
    etc.
Procedure RETURN like that of Modula2.
Procedures nest like those of Modula2.
Local variables can be static or auto.
Variable initialization can be done with a simple predicate in
    the definition or with an arbitrary number of separate
    statements.
```

To visualize the interactive aspect of the system, imagine starting with Turbo Pascal and expanding the command shell to include the operating system. Then add assembly statements to the language. Then add commands similar to those of a machine language debugger (move, display, disassemble, etc). Then add the ability to execute any Pascal statements or assembly statements from the keyboard. Then add the keywords STOP and CONT to the language. The compiled code for STOP is a call to the command loop (runtime regs are saved). When in the stopped condition, you can look at variables or registers, change variables or registers, call procedures, single-step the machine instructions of the program, etc. When you type CONT the program then continues. If you know the address to stop at, you can set a breakpoint from the keyboard instead of recompiling to insert the STOP. You can enter the breakpoint address symbolically if the program has a label at the appropriate place. When in the stopped condition you can also execute the ROM monitor, do commands, reenter 32HL and continue the stopped program. Any global variables or procedures that are created during the compilation of a source file or a keyboard line are available for use by subsequently compiled source files or keyboard lines. This thing is more interactive than Forth because at the keyboard we can execute

```
FOR j=0 TO 23 <statements> NEXT j
```

but in Forth you can't do that unless you use a colon definition to make that line a new keyword, then execute the keyword, then 'forget' the keyword.

On the implementation side, all pre-defined words, including DOS commands and library procedures are to be in the main state machine. Another state machine that handles operands will contain the library functions. User defined words go into a heap area, and an array of pointers is used to find the symbols with a binary search. At the end of a procedure, the symbol table is truncated back to the point where the procedure started. Therefore, the binary search is performed on clumps instead of the whole table at once. The source text buffering uses a method that allows the parser and state machine to run blind without checking for the end of the buffer. Because the intended usage of this thing demands rapid compilation, there will be no optimization and no

text substitution facility like #define in C.

I've always been mystified by the fact that some high level languages don't have certain high level features that are included in the most primative of assemblers. Until just recently, BASIC did not allow symbols for branch or call addresses, and C does not have symbolic constants that assemblers and other languages have. Text substitution is a perverse way to simulate symbolic constants (because of the huge time penalty). I was also shocked when I learned that C did not check the parameter list of the function definition when compiling a call to a function. In my first kludgy experiment in compilation of functions, I would not have dreamed of neglecting this kind of checking. Of course, you would have to draw and quarter me to make me do run-time checking.

Here's a summary of the current state of 32HL: It's a complete native assembler and assembles itself in one pass. It calls Z80 CP/M to get source text from a CP/M file. It has its own command loop and executes statements (assembly or otherwise) that are typed at the keyboard. It has character, string, and numeric I/O routines that are callable by client programs. I now have the disassembler integrated and STOP/CONT implemented, but not breakpoints or the high level words. The transcendentals and complex transcendentals are written and waiting to be installed. I'm now blacksmithing the expression parser to fit the new environment. When I get the expression parser in place, all the high-level keywords will fall in almost automatically.

Since I'm only a hobbyist running open loop, I welcome any comments about what I'm doing wrong or what would be neat to include.

Those interested in 32000 hacking should also contact Richard Rodman, 1923 Anderson Road, Falls Church VA 22043. He has started a small newsletter for 32000 hackers, which he currently gives free of charge. It is to be a forum on the creation of a public domain operating system, and a focal point for exchange of 32000 schematics, software, and information. He is starting an inventory of P.D. software that anyone can get for $8 a disk. The first volume contains my cross assembler, my Z80 assembler, and 32000 source code for transcendentals, complex math, complex transcendentals, a disassembler, and a math-intensive orbital game program in assembly. If anyone wants to see my old FORTH/BASIC system or my half-done 32HL system, I can send those for $8 each on CP/M SSSD 8" or on IBM 360k format (the IBM format for mailing only; no IBM code). ■

Neil R. Koozer
Kellogg Star Route Box 125
Oakland, Oregon 97462
(503)-459-3709

Multitasking

(Continued from page 38)

can be edited and reassembled with a larger queue. At 9600 baud, 960 characters can be sent each second. At low baud rates, fewer can be sent. Send small groups of characters so that the rate of sending will not exceed the rate of the hardware, causing a wait from the system.

A second common control action is for the program to do specific actions at regular time intervals. These actions might be taking a reading or creating some kind of a log record. The get time command gets the elapsed time from when the system started. The number has no meaning per se, but elapsed time can be measured by comparing the results of two different calls. Another way is to use a software timer with a flag, and check the flag to see if the correct amount of time has elapsed.

The routines presented here can be used as a starting point for multitasking process applications. They can help simplify what might otherwise have been a complex or unsovable problem. Even though the examples given use HTPL and K-OS ONE, it is possible to use the same techniques with other operating systems or for dedicated systems that don't have an operating system installed. Also, these techniques are a much lower cost solution than using a real time kernel.

■ ■ ■

The CP/M Corner

by Bob Blum

Starting a new column for a magazine that I have never written for is a little unnerving. For over four years I edited columns of this type for other magazines and as you might expect, in that length of time I became very comfortable doing them. The primary reason for my peace of mind, though, was not familiarity with the work but the responsiveness of the readers. I have always depended heavily on them to provide me with their feedback on how I am doing and most importantly to tell me what type of material was most valuable to them. These ideas, many times, became the subjects of columns. As you can see, my job wasn't very difficult, all I had to do was listen.

HELP.....

To get this column started on the right foot I need your help. The more quickly I get to know you and your interests the faster I can begin to cover the subjects most important to you. Drop me a note describing your interests, or maybe you have a nagging problem that seems to be insolvable. I may not be able to solve it myself, but I'm sure one of the other readers can. Have you run into a particularly good program lately? Tell me about it. No matter whether it came from the public domain or you purchased it commercially, personal recommendation is always the best reason to investigate a software package for potential review. And while you're in a giving mood don't forget to send along those special programs or subroutines that you worked so hard to develop. When space permits I will run the source code of the shorter ones as a part of this column but will offer all of them on disk; more on that subject in the next column.

If you still aren't moved to write do it as a favor to me just to impress my editor. After all, I'm the new kid on the block and need all the help I can get.

Don't Fence Me In

There are, of course, some guidelines that I must follow when deciding what material to work with. The border of my columnistic territory is the world of CP/M compatible 8-bit machines.

Rather than limiting the editorial content of this column to just the highly technical issues surrounding CP/M I hope to give equal time to new products as well. Many companies, mainly the smaller one and two programmer shops, continue to produce an amazing assortment of new software packages. But again, the direction I take will be based on your guidance. So, let me hear from you.

In the Queue

I always have a seemingly endless list of projects in one state of completion or another. Mine is a problem of priority, not one of too little to do. Nonetheless, as they are finished I will make the source code, if any, available on disk and space permitting run the listings here as well.

The projects that I am currently involved with include both hardware and software items. Here are a few:

I've spent some time with the latest 8-bit single board computer being produced by Micro Mint. This little barn burner, smaller than a mini-floppy disk drive, sports the HD64180, the newest 8 bit chip being produced by Hitachi. I'm told that this device is at least as powerful as an 8 MHZ Z80 while offering a much higher level of integration that includes I/O ports and other features that normally require many external devices.

Even thought DRI has placed CP/M Plus on the mature list and as such will no longer actively support it, I am continuing to port it to different computers. My most recent adventure is with the Intercontinental 8 bit S-100 master board; on board memory management and many other performance features make the ICM board a natural for CP/M Plus.

Among the top ranking features of CP/M Plus is its capability to automatically time and date stamp files at each access. This feature is especially important to me because I have the untidy habit of not religiously tracking my disk files during a heated development session. I wait for the project to conclude before trying to make good sense out of my files and then sometimes have trouble recalling the order in which I created them, and most importantly, which is the most current. Until Plu*Perfect Systems began to sell DateStamper there was no software available to add this desirable feature to CP/M 2.2. I've used DateStamper and have found it a most useful tool and safeguard.

If you're one of the many Z-80 computer owners that lack the hardware necessary to maintain time and date information then Ztime-1 may be for you. It is a complete real time battery driven clock board that is installed by inserting it in place of the Z80 chip of your computer's main logic board. The Z80 chip is then placed in the socket provided on the Ztime-1 board.

If you have ever run out of symbol table space during an assembly the folks at SLR Systems can help you. They have recently released the latest version of their now famous Z80ASM assembler. It has extra logic incorporated that will begin using tables on disk if memory fills to capacity.

Outside of porting CP/M Plus to different machines my favorite project is comparing CP/M compatible operating systems. I have successfully implemented three out of the four packages that I have been able to find. I think you will be just as excited by the results as I am.

I could go on and on, my list is far from empty, but at this point I think it's time to listen for a while. Please let me hear from you. ■

MDISK

Part 2: Software Drivers for Ampro RAM Disk

by Terry Hazen & Jim Cole

Introduction

In this second part of our series on the MDISK add-on 1 megabyte RAM disk for the AMPRO Z80 Little Board® , we will present the software drivers that are added to the AMPRO BIOS in order to enable the Little Board to use the extended RAM as a solid state disk drive, complete with a system track. As the driver routines include the modification of the CP/M® Disk Parameter Block, you may wish to review Thomas Hilton's article, "Disk Parameters" in Issue #27 of TCJ for more detailed information on the CP/M DPB.

There are several approaches to software RAM disk drivers. We have chosen a very direct approach, which is to add the driver routines directly to the AMPRO BIOS. This approach takes advantage of existing BIOS routines, creates the most compact and integrated code, and is not troubled by possible interactions with programs such as keyboard redefinition utilities that relocate themselves into high memory. In order to modify the BIOS, however, you must first have a copy of the BIOS source code, which is available directly from AMPRO, as you cannot add the new routines directly to the BIOS HEX file or to the operating system.

The AMPRO BIOS

MDISK requires the use of AMPRO BIOS version 3.0 or above, which offers much greater capabilities than the earlier BIOS versions. Although the AMPRO BIOS is written in 8080 assembly language, it actually uses many Z80 instructions. These instructions are entered as data words (DW) in order to fool ASM.COM into thinking that they are merely data so that it doesn't attempt to process them. Since the BIOS is really Z80 code, and since we use Z80 code in the MDISK driver routines, we have chosen to present the driver routines in this article entirely in Z80 code for clarity. The complete insert listings (in 8080 code) are available as MDISK.LBR on the TCJ disk. If you have the BIOS source code disk handy, you might want to bring it up on your word processor at this point and follow along.

MSIZE

One result of adding additional code to the BIOS is that it slightly increases the size of the BIOS. In order to compensate, we must reduce the size of the TPA by decreasing the value of the BIOS equate MSIZE by one. This amounts to a decrease of 1k in TPA size. For example, if you are presently using an all-floppy 60k BIOS, you would need to change MSIZE from 60 to 59 when you add the MDISK drivers, giving you a 59k MDISK system. You will also need to use ZMOV-CPM to generate a new operating system of the new size for use with your new BIOS.

The Basic MDISK Equates

The MDISK equates are all grouped together and centrally located for easy modification. They are inserted into the BIOS following the existing equate DELSEND, in the BIOS section "Other equates", as shown in Listing 1. Keep in mind that the values presented here are our own compromise design and not the only possible approach. You may desire to modify them for your specific application.

MDPORT is the port number of our MDISK control port. We have chosen to use port 30H. If you are using a different port, put that port number here.

CHIPS is the number of 8 chip 256k RAM chip sets that are installed (the maximum number is 4 sets, giving 1 megabyte total), and is used later in MDPARM to calculate the appropriate extent mask (EXM) and the disk size (DSM).

MDIRSIZ is the maximum number of directory entries allowed. We have chosen to allow a maximum of 128 directory entries no matter how many chip sets are installed in order to minimize the reserved directory size and maximize the net MDISK storage space. Since the MDISK RAM disk is primarily a workspace rather than permanent storage,

Listing 1.

```
DELSEND EQU     28
DDLSPT  EQU     40

;========== Insert modified code as shown below: ========================;
;                                                                        ;
; MDISK equates:                                                         ;
;                                                                        ;
;                                                                        ;
MDISK$VERS EQU  50              ; Software version                       ;
;                                                                        ;
MDPORT  EQU     030H            ; Control port for bank                  ;
;                               ;    switching                           ;
CHIPS   EQU     4               ; Number of chip sets installed (1-4)    ;
;                               ;    (each set is 8 256k chips)          ;
DELCHR  EQU     0E5H            ; Deleted directory character            ;
ZBANK   EQU     00000000B       ; Zero bank selection                    ;
MDISKLED EQU    10000000B       ; MDISK LED ON bit                       ;
;                                                                        ;
MDIRSIZ EQU     128             ; MDISK directory size                   ;
BLKSIZ  EQU     2048            ; MDISK allocation block size            ;
;                                                                        ;
MD$CKS  EQU     0               ; No directory checking required         ;
MD$ALV  EQU     60              ; Max 472 2k disk blocks                 ;
;                                                                        ;
;========== End of modified/inserted code ==============================;
;
```

workspace elbow room seems more important than a very large number of potential directory entries. MDIRSIZ is used later in MDPARAM to compute the maximum directory size (DRM). If you want to use MDISK as a battery backed-up RAM disk that is used for more permanent storage and you have the need for more than 128 files, you might wish to increase MDIRSIZ. In that case, you will also have to modify AL0 as described below.

BLKSIZ is the size, in bytes, of each allocation block. Blocks of 2k were chosen as a good compromise. Larger blocks waste too much RAM disk space and smaller blocks require more directory space.

MD$CKS is the directory check scratchpad area size, which is used to determine whether the media has been changed. We can set this equate to zero, as MDISK is a "non-removable media" device, hence needs no directory checking.

MD$ALV is the size of disk storage allocation information scratchpad area (ALV). This value is equal to the maximum number of data blocks allowed, and is computed as (DSM/8)+1. DSM is the maximum disk size and is computed in MDPARM. For simplicity, we set this equate to the maximum size needed, which is the size required by 4 chip sets, or 60 2k blocks.

Listing 2.

```
DPBASE:
       - the BIOS hard disk information is skipped in this listing -

        IF      NOT HARD$DISK

;========== Insert modified code as shown below: ======================;
;                                                                       ;
;       Insert DPHMD: as a replacement for the DPH for whichever        ;
;       drive letter you have selected for MDISK.  Drive F is shown     ;
;       here.                                                           ;
;                                                                       ;
DPHMD:                                                                  ;
        DEFW    0,0,0,0,DIRBUF,MDPARM,CSVMD,ALVMD                       ;
;                                                                       ;
;========== End of modified/inserted code ============================;
;                                                                       ;

        DEFW    0,0,0,0,DIRBUF,SPARM,CSVG,ALVG

          etc, up through drive P:

        DEFW    0,0,0,0,DIRBUF,SPARM,CSVP,ALVP
        ENDIF


;========== Insert modified code as shown below: ======================;
;                                                                       ;
MDPARM:                                                                 ;
        DEFW    128             ; SPT: Sectors/track                    ;
        DEFB    4               ; BSH: Block shift                      ;
        DEFB    15              ; BLM: Block mask                       ;
        DEFB    (4-CHIPS)/2     ; EXM: Extent mask                      ;
        DEFW    128*CHIPS-41    ; DSM: Disk size - 1                    ;
        DEFW    MDIRSIZ-1       ; DRM: Dir max - 1                      ;
        DEFB    11000000B       ; AL0: Dir bit map: for 128 entries     ;
        DEFB    00000000B       ; AL1: Dir bit map                      ;
        DEFW    0               ; CKS: Check size                       ;
        DEFW    1               ; OFF: Track offset, 1st track is system;
;                                                                       ;
;========== End of modified/inserted code ============================;
;

SPARM:                          ; Ampro single sided 48tpi
```

MDISK Disk Parameter Information

The next MDISK BIOS insert, shown in Listing 2, follows the AMPRO label DPBASE, which is where the parameters for each drive are listed. MDISK looks like a floppy disk to the BIOS, so we will operate in the "IF NOT HARD$DISK" area. We're assuming that we're using an all-floppy system and using drive "F", the first available non-floppy drive selection, as our MDISK drive. We must then modify the DPH for drive F. If you're using a different drive letter for MDISK, modify the DPH for that drive instead.

DPHMD is the MDISK Disk Parameter Header, and contains information that helps define the MDISK "disk drive". The first byte is the address of the logical-to-physical translation vector. MDISK doesn't use translation, so we set that byte to zero. The next three bytes are scratchpad areas. DIRBUF is the address of a 128 byte scratchpad area for directory operations, and is shared by all the drives. It is not modified. MDPARM is the address of the MDISK disk parameter block, which defines the characteristics of the MDISK RAM disk and is discussed in more detail below. CSVMD is the address for the scratchpad area used to check for changed disks. MD$CKS, which we have set to zero earlier, is the size of this area. ALVMD is the address of the scratchpad area for BDOS disk storage allocation information. MD$ALV, also set earlier, is the size of this area.

MDPARM contains 15 data bytes of disk information that describe the characteristics of the disk organization. The first byte, SPT, is the number of 128 byte records per track. We use 128, which gives us 16k tracks. The next two bytes are the block shift factor (BSH), and the block mask (BLM). Because our block size is 2k, it turns out that our BSH should be 4 and our BLM should be 15. The next two bytes are the extent mask (EXM), and the disk size – 1 (DSM). Given a block size of 2k, both of these values depend on the number of chip sets that we have installed. EXM is determined by the block size and the total number of blocks. If we install one or two chip sets, EXM will be 1. Either 3 or 4 chip sets will require an EXM of 0. DSM is the maximum data block number – 1, not counting the one 16k system track we're reserving for the operating system and directory. This value will vary according to the number of chip sets installed, from 87 for one chip set to 471 for all 4 chip sets. By earlier setting the CHIPS equate to the number of chip sets you will use, these calculations will be made automatically during assembly of the BIOS.

The Directory Maximum – 1 (DRM), is the total number of directory entries allowed – 1. Since we selected 128 as the maximum number of entries, DRM will be 127.

AL0 and AL1 are really a single 16 bit word representing a bit map showing the data blocks reserved for directory entries. Since each directory entry is 32 bits and we need space for 128 entries, we need 32

Listing 3.

```
PHYTAB:
        DEFB    1,000H, SSDD48, 0H ; Floppy drive A
        DEFB    1,011H, SSDD48, 0H ; Floppy drive B
        DEFB    1,022H, SSDD48, 0H ; Floppy drive C
        DEFB    1,033H, SSDD48, 0H ; Floppy drive D

        DEFB    2,044H, 0H, 0FFH ; Special floppy drive E

;========== Insert modified code as shown below: ======================;
;       Insert this section as a replacement for the driver statement  ;
;       for whichever drive letter you have selected for MDISK.        ;
;       Drive F is shown here.  Note that the label: "Hard disk" here  ;
;       is irrelevant.                                                 ;
;                                                                      ;
        DEFB    7               ; MDISK driver number                  ;
        DEFB    050H            ; High nibble: Drive "F" offset        ;
                                ;    from DPBASE (05)                  ;
                                ; Low nibble:  0H                      ;
        DEFB    4               ; MDISK type byte                      ;
        DEFB    0H              ;                                      ;
;                                                                      ;
;========== End of modified/inserted code ============================;
;

        DEFB    0,060H,0,0      ; Hard disk drive G

          etc, up through drive P:

        DEFB    0,0F0H,0,0      ; Hard disk drive P

DRVRADR:                        ; Driver table
        DEFW    SELERR          ; Driver 0 - select error
        DEFW    FLOPPY          ; Driver 1 - floppy
        DEFW    DISKE           ; Driver 2 - E-disk
         IF     HARD$DISK
        DEFW    DRVR3           ; Driver 3 - SCSI hard disk
         ENDIF
         IF     NOT HARD$DISK
        DEFW    SELERR          ; Driver 3 - not defined
         ENDIF
        DEFW    SELERR          ; Driver 4 - not defined
        DEFW    SELERR          ; Driver 5 - not defined
        DEFW    SELERR          ; Driver 6 - not defined

;========== Insert modified code as shown below: ======================;
;                                                                      ;
        DEFW    MDRIVER         ; Driver 7 - MISK driver               ;
;                                                                      ;
;========== End of modified/inserted code ============================;
;

PHYEND  EQU     $

;========== Insert modified code as shown below: ======================;
;                                                                      ;
; MDRIVER - MDISK driver routines                                      ;
;                                                                      ;
;  Entry:                                                              ;
;       If A = FFH then select MDISK                                   ;
;              00H then write MDISK                                    ;
;              01H then read MDISK                                     ;
;                                                                      ;
MDRIVER:                                                               ;
        INC     A                                                      ;
        JP      Z,SELMDISK      ; Go select MDISK                      ;
        DEC     A                                                      ;
        JP      Z,MWRITE        ; Write operation                      ;
        JP      MREAD           ; Read operation                       ;
;                                                                      ;
SELMDISK:                                                              ;
        JP      SELEND          ; Nothing else special here            ;
;                                                                      ;
; MDISK disk read and write routines                                   ;
;                                                                      ;
```

× 128 or 4k of directory space. This means that we need two 2k blocks, so we will set the highest two bits in AL0 to 1.

The final two words are the check size, CKS, and the system track offset, which is the number of reserved system tracks. Since we do no directory checking, CKS is zero. We have reserved one system track, so OFF is set to 1.

Physical Driver Table

The next BIOS insert, also shown in Listing 2, follows the AMPRO label PHYTAB, and contains the MDISK driver number, the default MDISK drive offset, and the MDISK type byte. AMPRO allows for up to 7 drivers, and is already using drivers 1 to 3. We have selected driver 7 for our MDISK driver in order to keep us out of the way of possible expansions.

The next byte contains the physical offset from DPBASE in the DPH table in the high 4 bits and the physical device address in the low 4 bits. Since we're using drive "F" as our MDISK drive letter, our offset is 5 (since A = 0). If you're using a different MDISK drive, this offset must be changed, and instead of modifying the listing for drive "F", you must modify the listing for the drive letter you have chosen. We have no physical device address, so the low bits are 0, giving us an offset of 050H.

The type byte contains information about the density, number of sides, the sector numbering, the track count, the block size, and the sector size. When you work all that out according to the information comments in the AMPRO BIOS, you get a type byte of 4, which describes a single sided, single density drive with a block size of 2k and a sector size of 128 bytes.

MDRIVER is the address for our MDISK driver number 7, which is shown in Listing 3. Whenever the MDISK drive is selected, the BIOS directs us to driver 7 for further processing.

MDISK Drivers

The MDISK read and write routines, MREAD and MWRITE, are very straightforward and function in similar ways. They begin by setting a one level local stack to make sure that the stack pointer will not get lost. PAGESEL translates the track and sector numbers to the bank number our hardware needs and turns on the MDISK drive LED. The data is then moved in 128 byte blocks using the OUT MDPORT instruction to select the desired bank and the Z80 LDIR instruc-

tion to move the data between the selected locations. When we are finished, we restore the stack and select the CP/M bank (ZBANK), which also turns off the MDISK drive LED on our way back to the calling routine.

MDISK Setup Routines

When the system is cold booted or reset, we need to initialize the MDISK directory area so that we can use it. We also don't want to lose our MDISK directory if one is already present, so we need to provide a routine that tests for the presence of an MDISK directory and will create one if it doesn't already exist. We insert this routine, shown in Listing 4, into the BIOS following the BIOS label NOT7 just after the sign-on message is displayed.

We need a way to tell whether we already have a disk directory, so we will create a "label" file when we boot the operating system. We will call our label file "-MDISKXX.RAM", where XX is the MDISK software version number, "50" in this case. By using a leading dash in the filename, we accomplish several things. This way, the label file usually ends up in the upper left corner of the directory listing if your directory utility gives you a sorted listing. We also make it compatible with many disk catalog utilities and with the very useful ZCPR3 named directory utility LDSK, which can create a named directory NDR file from all selected user areas that have label filenames beginning with a dash. LDSK will ignore the extension, "RAM", and create the named directory "MDISK50".

To do our entry test, the MDTEST routine first looks at the first MDISK directory entry for the label file "-MDISK50.RAM". If it is found, we will assume that a directory already exists and that we can tell the operator so and continue with the boot sequence without doing any more setup work. If it is not there, we assume we will have to create a new directory, which will wipe out any existing directory information.

To protect our directory from accidental deletion on cold boots, it is important not to accidentally erase the label file, so we will make it a read-only file. To prevent our label file from being inverteently copied by utilities like AC or ACOPY that have the option to copy files if they don't have the ARCHIVE bit in the filename set, we will also make it an ARCHIVED file. In Listing 4, you will see that this involves adding 080H to the proper filename character in each case.

Listing 3 Continued

```
;
MREAD:                                                                  ;
        LD      IX,00           ; Load IX with 0                        ;
        ADD     IX,SP           ; Add SP and IX to save SP in IX        ;
        LD      SP,MSTACK       ; local stack area                      ;
        CALL    PAGESEL         ; Select the correct bank of memory     ;
        LDIR                    ; Move it                               ;
        LD      A,ZBANK         ; return to CP/M bank                   ;
        OUT     (MDPORT),A                                              ;
        LD      HL,(DMAADR)     ; Point to destination of data          ;
        EX      DE,HL                                                   ;
        LD      HL,MDBUF        ; Point to source of data               ;
        LD      BC,128          ; 128 bytes of data to move             ;
        LDIR                    ; Move it                               ;
        XOR     A               ; Zero to accum                         ;
        LD      (ERFLAG),A      ; No errors                             ;
        LD      SP,IX           ; Restore SP to entry value             ;
        RET                     ; Done...                               ;
;                                                                       ;
MWRITE:                                                                 ;
        LD      IX,00           ; Load IX with 0                        ;
        ADD     IX,SP           ; Add SP and IX to save SP in IX        ;
        LD      SP,MSTACK       ; Local stack area                      ;
        LD      BC,128          ; Similar to read, except backwards     ;
        LD      DE,MDBUF        ;    use of buffers                     ;
        LD      HL,(DMAADR)                                             ;
        LDIR                    ; Move it                               ;
        CALL    PAGESEL                                                 ;
        EX      DE,HL                                                   ;
        LDIR                    ; Move it                               ;
        LD      A,ZBANK                                                 ;
        OUT     (MDPORT),A                                              ;
        XOR     A               ; Zero to accum                         ;
        LD      (ERFLAG),A      ; No errors                             ;
        LD      SP,IX           ; Restore to entry value                ;
        RET                     ; Done...                               ;
;                                                                       ;
PAGESEL:                                                                ;
        LD      HL,(SEKTRK)     ; 1/2 32k page = 1 track                ;
        LD      A,(SEKSEC)      ; Get sector                            ;
        RLA                     ; Setup sector value to compute bank #  ;
        LD      B,A             ; Save for later                        ;
        LD      A,L             ; Makes it almost too easy              ;
        ADD     A,4             ; Skip over the 4 16K tracks that       ;
                                ;    make up the 64k CP/M system        ;
        RRA                     ; Move low order track bit to high order;
        LD      C,A             ; Sector value to select addr in bank   ;
        LD      A,B                                                     ;
        RRA                                                             ;
        LD      B,A                                                     ;
        LD      A,C             ; Get bank number                       ;
        OR      MDISKLED        ; Turn on led                           ;
        OUT     (MDPORT),A      ; Select bank into lower 32K addresses  ;
;                               ; Each sector is 128 bytes              ;
        LD      H,0             ; Multiply by 128 to get                ;
        LD      L,B             ;    sector address                     ;
        ADD     HL,HL           ; *2                                    ;
        ADD     HL,HL           ; *4                                    ;
        ADD     HL,HL           ; *8                                    ;
        ADD     HL,HL           ; *16                                   ;
        ADD     HL,HL           ; *32                                   ;
        ADD     HL,HL           ; *64                                   ;
        ADD     HL,HL           ; *128                                  ;
        LD      BC,128          ; Number of bytes to move (1 sector)    ;
        LD      DE,MDBUF        ; Point to destination of data          ;
        RET                                                             ;
;                                                                       ;
        DW      0000H           ; Local Stack Area                      ;
MSTACK:                         ; Only 1 level required                 ;
;                                                                       ;
;========== End of modified/inserted code ==============================;
;
```

Listing 4.

```
NOT7:
        XOR     A               ; Clear 5380 registers (SCSI init)
        OUT     (NCRICR),A      ; .
        OUT     (NCRMR),A       ; .
        OUT     (NCRTCR),A      ; .
        OUT     (NCRSER),A      ; .
         ENDIF                  ; End of HD initialization

        XOR     A               ; Patch out old CP/M message
        LD      (CPMMSG),A
        LD      HL,LOGMSG       ; Display signon message
        CALL    PUTS
        LD      HL,Z80MSG
        CALL    PUTS

        LD      HL,DRIVE$TDATA  ; Init drive type data
        LD      DE,DRIVE$TYPES
        LD      BC,8
        DEFW    LDIR80

;========== Insert modified code as shown below: ======================;
;                                                                      ;
        LD      A,02            ; First MDISK bank, sys & dir track    ;
        OR      MDISKLED        ; Turn on led                          ;
        OUT     (MDPORT),A      ; Select bank                          ;
        LD      DE,MDENTRY      ; MDISK entry                          ;
        LD      C,32            ; Length of entry                      ;
        LD      HL,4000H        ; Point to first directory entry       ;
;                                                                      ;
MDTEST:                                                                ;
        LD      A,(DE)          ; Get first byte                       ;
        CP      (HL)            ; Compare with MDISK entry             ;
        JP      NZ,MDNSETUP     ; No match - fail                      ;
        INC     DE                                                     ;
        INC     HL                                                     ;
        DEC     C                                                      ;
        JP      Z,MDSETUP       ; All bytes match                      ;
        JP      MDTEST          ; Continue test                        ;
;                                                                      ;
MDSETUP:                                                               ;
        LD      HL,MDSUMSG      ; Inform user                          ;
;                                                                      ;
MDSUDONE:                                                              ;
        LD      A,ZBANK         ; Go to normal CP/M                    ;
        OUT     (MDPORT),A      ;    pages                             ;
;                                                                      ;
MDSUMSGP:                                                              ;
        CALL    PUTS            ; Send to screen                       ;
        JP      CPMBOOT         ; Go to normal boot sequence           ;
;                                                                      ;
MDNSETUP:                                                              ;
        LD      DE,4000H        ; Move MDISK directory entry into      ;
        LD      HL,MDENTRY      ;    MDISK directory area              ;
        LD      BC,32           ; Entry is 32 bytes long               ;
        LDIR                    ; And move it                          ;
;                                                                      ;
        LD      A,DELCHR        ; Get blank directory character        ;
        LD      (DE),A          ; Store first character                ;
        LD      H,D                                                    ;
        LD      L,E                                                    ;
        INC     DE                                                     ;
        LD      BC,(BLKSIZ*4)-32 ; 4 allocation blocks long            ;
        LDIR                    ; Move it                              ;
;                                                                      ;
        LD      HL,MDNSUMSG     ; Tell operator MDISK initial          ;
        JP      MDSUDONE        ;    directory is all set up           ;
;                                                                      ;
MDSUMSG:                                                               ;
        DEFB    '   MDISK directory exists',CR,LF,0                    ;
;                                                                      ;
MDNSUMSG:                                                              ;
        DEFB    '   MDISK directory cleared',CR,LF,0                   ;
```

To create our label file directory entry, we move the directory data contained in MDENTRY to the first MDISK directory location. When that has been completed, the rest of the directory area is filled with the blank directory character 0E5H. The operator is then notified that the directory has been cleared and we continue with the boot sequence.

We can now reset the system at any time (as long as we don't turn the power off!) without danger of losing our MDISK directory data and files, providing that we retain our label file in the MDISK directory. We can also safely boot normal non-MDISK systems, since they can't address the extended RAM area where the MDISK directory and files reside. When we later re-boot our MDISK system, the directory and files will still be there as long as the power has remained on.

Creating Additional Files At Boot Time

We can use the same method that we used to create our label file if we wish to also create other useful files at boot time. For example, if you use PluPerfect's DateStamper® utility, you can also create the required blank "!!!TIME&.DAT" file at boot time by adding the appropriate directory data information to MDENTRY and moving it to the second MDISK directory location. The insert listings in MDISK.LBR include a DateStamper equate in the first DELSEND insert that, if set to YES, will automatically create this file at boot time.

Log-on Message

Listing 5 shows the MDISK addition to the AMPRO log-on message. It is used to notify the operator that the system being booted is an MDISK system. Of course you may customize this message if you desire.

Buffers

Listing 6 shows the final BIOS insert. It contains MBUF, a 128 byte buffer used for MDISK data movement, and the values for CSVMD and ALVMD that were set in the first insert.

Driver Operation

The normal driver operating sequence is to perform a call or jump in order to pass control to the driver code in the global upper 32k of system memory. Then select the proper bank by outputting a byte to the control port. Move the data to or from the RAM disk, depending on whether a read or write operation is being performed. Then output another byte to

the control port to bring Bank 1 Page 1 back into place. Finally, return to the calling routine. This completes the sequence and returns you back to a normal C/PM configuration.

Creating Your New MDISK System

The modified BIOS we have created isn't useful until it has been assembled and combined with the rest of the operating system. Start by renaming your modified BIOS to MDBIOS.ASM, assemble it with ASM, and place a copy of it on a fresh system disk.

Create the system image of your old system as follows (assuming that you are creating an all-floppy 59k system):

```
A0>ZMOVCPM 59 *<RETURN>

CONSTRUCTING 59K CP/M vers 2.2
READY FOR "SYSGEN" OR
"SAVE 41 CPM59.COM"

A0>SAVE 41 OLD.SYS<RETURN>
```

Note that the CP/M size number, in this case, "59", must match the MSIZE value in your modified BIOS.

Now use DDT to create a new system image file, MDISK.SYS:

```
A0> DDT OLD.SYS
DDT VERS 2.2
NEXT     PC
2A00    0100
-IMDBIOS.HEX
-R3980
NEXT     PC
2A00    EE00
-GO

A0>SAVE 49 MDISK.SYS
```

Use one of the following read offset values, according to the size of your system:

MSIZE (k)	Offset
59	3980H
58	3D80H
57	4180H
56	4580H

Finally, use SYSGEN to install the new system on a formatted disk. If you run ZRDOS, you can install it at this point. Before you boot your new system, run CONFIG.COM to copy your current system configuration information from your current system disk to your new MDISK system disk.

Your Working MDISK System

If all has gone well, you should now have a new system disk that will boot an operating system with MDISK as drive "F". You can now run SYSGEN to copy the operating system to drive "F", and the AMPRO SWAP utility to swap drives

Listing 4 Continued

```
;                                                                    ;
MDENTRY:                                                             ;
        DEFB    00                      ; User 0                     ;
        DEFB    '-MDISK'                ; File name                  ;
        DEFB    MDISK$VERS/10+'0'       ; Version number             ;
        DEFB    MDISK$VERS MOD 10+'0'                                ;
        DEFB    'R'+080H,'A','M'+080H   ; Extension with R/O, ARC flags ;
        DEFB    00,00,00                ; Bit map:                   ;
        DEFB    00,00,00                                             ;
        DEFB    00,00                                                ;
        DEFB    00,00,00,00                                          ;
        DEFB    00,00,00,00                                          ;
        DEFB    00,00,00,00                                          ;
;                                                                    ;
CPMBOOT:                        ; Normal CP/M boot sequence          ;
;                                                                    ;
;========== End of modified/inserted code ============================;
;
```

Listing 5.

```
SCSI$ID$FOUND:
        DB      'x'
        ENDIF
        IF      HARD$DISK AND ARBITRATION
        DB      CR,LF,'(Arbitration Enabled)'
        ENDIF
        IF      HARD$DISK AND (NOT ARBITRATION)
        DB      CR,LF,'(Arbitration Disabled)'
        ENDIF

;========== Insert modified code as shown below: ======================;
;                                                                      ;
        DEFB    CR,LF,'(MDISK vers '                                   ;
        DEFB    MDISK$VERS/10+'0','.'                                  ;
        DEFB    MDISK$VERS MOD 10+'0'                                  ;
        DEFB    ' Enabled)'                                            ;
;                                                                      ;
;========== End of modified/inserted code ============================;
;
```

Listing 6.

```
DIRBUF: DEFS    128             ; DIRECTORY ACCESS BUFFER

;========== Insert modified code as shown below: ======================;
;                                                                      ;
MBUF:   DEFS    128             ; Memory buffer for MDISK              ;
                                                                       ;
CSVMD:  DEFS    MD$CKS                                                 ;
ALVMD:  DEFS    MD$ALV                                                 ;
;                                                                      ;
;========== End of modified/inserted code ============================;
;
;       *       *       *       *       *       *       *       *
;
;  Floppy drive directory check vector storage

        DEFS    128             ; DIRECTORY ACCESS BUFFER
```

"F" and "A". The ZCPR3 utility PATH can be used to change your file search path to accommodate your new system. The path should begin with the current drive and user, then move to drive "A", your logical MDISK drive, and if you keep all your utility files in the disk in physical drive "A" (which is now logical drive "F"), then that is where the path should end up. Your new path assignment might look like the following:

```
$0:  (current disk, user 0)
$15: (current disk, user 15)
A0:  (MDISK drive, user 0)
F0:  (Your physical drive A, user 0)
F15: (Your physical drive A, user 15)
```

which assumes that you only use MDISK user 0. Organize the new path to suit your own system and work habits. MDISK.LBR contains MDISK.COM, a utility that will take care of all this for you when it is included in your STARTUP alias.

Setting Up MDISK Using ALIAS Files

Now that you have a working RAM disk, you will need to copy the utility and working files you will be using from your utility disk to MDISK. You can use VALIAS and ACOPY to easily create powerful SETUPxxx utilities. VALIAS accepts multiple commands and creates an ALIAS which will execute those commands when run.

A simplified setup utility, SETUP, with a part of the VALIAS display shown below, assumes that the desired utility files have had the file attribute F2 previously set. If you are using ZRDOS, the utility SFA will set the file attributes, otherwise you can use a public domain utility such as DA21. SETUP uses the ACOPY "2" option to copy all the files on F0 that have F2 set to MDISK. The command line parameters $1 (the drive/user specification) and $2 (the filename specification) are used to copy any other files specified on the command line. The ACOPY "S" option instructs ACOPY to skip the copy if the destination file already exists. SETUP B:FILENAME.EXT would copy the desired utility files from drive F0 and "filename.ext" from drive B0. The last command in the ALIAS is to display a directory of MDISK:

```
1-->A:
2-->ACOPY F:ACOPY.COM /S
3-->ACOPY F:*.* /2S
4-->ACOPY $1$2 /S
5-->DIR
```

For more information on ALIAS use, see Jay Sage's "ZSIG Corner" column in issue #27 of TCJ.

Possible MDISK Enhancements

The extended RAM on the MDISK board can also be used for purposes other than a RAM disk. The control software is what turns these extended banks of RAM into a disk drive emulator. With the appropriate control software, the extended RAM could also be used as a printer buffer, multiuser work spaces, fast storage for real time data recording, work space for graphics routines, or for many other applications. Wouldn't it be nice, for example, to have your whole operating system buried on MDISK, giving you a TPA of 63k or more? The RAM disk software is the only application that has been developed at this time, as it is an application that is useful to almost everyone, but the extended RAM hardware needs only the additional control software to provide these other potentially powerful features. In the meantime, enjoy using the new speed of your Little Board/MDISK RAM disk workspace! ■

The MDISK user disk with the source files for the insert listings plus some utilities is available on an AMPRO DSDD format 5.25" disk for $10 postpaid in the U.S. & Canada.

Non-Preemptive Multitasking

by Joe Bartel, Hawthorne Technology

Multitasking, where one computer appears to be executing more than one program at the same time, is needed in many control applications. Usually a terminal or some kind of keyboard and display is communicating with the operator while a process is being monitored and/or controlled in the background. This is different from common programs being written for a PC where only one thing is happening at a time and one job is finished before the next job starts.

For every problem there is usually a simple solution and a complex solution. There are many multiuser, multitasking operating systems on the market today, of which Unix is just one example. There are also many small real-time kernels available on the market that implement preemptive multitasking without file management. If only some of the features of a full multitasking system are needed, then a special purpose program can do the job without many of the costs and problems that occur with a full generalized solution. By having a model of a nonpreemptive multitask system to follow, it is easier to conceptualize a solution and plan a program for a specific application.

How Preemptive Multitasking Works

In a normal or preemptive multitask system there is a scheduler module that takes over and takes control of the system away from one task and gives it to another task. The task swap can be in response to an interrupt from an external device or the result of a condition set by another task. The most common event is requesting I/O, which causes the task to wait until the request is filled. A task may also be swapped because the time allotted to it has expired and it is time for the next task to run awhile.

Programs start to get complicated when a task can be interrupted at any point instead of running to completion. First, any shared routines have to be reentrant. That means it must be possible for a second task to start executing the routine before the first task has finished its execution of the routine. Any data areas that are shared by two or more tasks must have locks and semaphores to prevent two tasks from trying to update the shared data at the same time and corrupting the data. Because the task swapping routine doesn't know anything about the task that is running it must save all the registers and status.

Most generalized multitasking systems have the ability to start new tasks and to delete old tasks. This requires sophisticated memory management to allocate and reclaim memory, and makes the control tables more complex because they can change in size. For many control multitask applications there is a fixed number of tasks and fixed memory allocation. The number of tasks is decided when the program is written and doesn't change when it runs.

The Value of Non Preemptive Multitasking

A nonpreemptive multitask system is one where the tasks cooperate with each other. A task runs for a while, long enough to accomplish something useful but not too long, and then it voluntarily gives up control of the system to the next ready task. Each task is also expected to be careful and not trash areas used by the other tasks in the system.

If a task cannot be swapped with another task without permission, then many things become much simpler. First you don't need the elaborate safeguards to prevent an update from being interrupted because the task will not be giving up control during the update process. Second you don't have to make shared routines reenterant because no other task will enter the routine until the using task is finished. Because a task knows it will be swapped out, the routine that gives up control can save any information that needs to be saved — the swap routine needs to save only the minimum.

For many applications, the use of nonpreemptive multitasking means that an existing single user/single task operating system can be used for character I/O and for disk file management. In our case this means the use of K-OS ONE, but the techniques will also work with CP/M and MS-DOS. A single task operating system is usually much less expensive, less complex, and smaller than a multiuser system. A single user system is also much more available.

In any multitask system it is important not to waste time that could be used by other tasks. There are two main areas where time gets wasted. One is waiting in a loop for a character from the console or for an I/O device to become ready. In this case the TESTIO command should be used to check for an available character and give up the machine to the next task if there is no character available. Another waste of time is a timing loop. To avoid this, the system time command should be used and if it is not yet time to act, then give up the system to the next task.

Implementation

Now that you are sold on the advantages of using a nonpreemptive multitask scheme how is it done? As simple as possible, that's how. Since everyone with K-OS ONE has an HTPL compiler, I will use that language to demonstrate the concept. The same principles can be used to multitask in almost any language but it will be more complex because more registers have to be saved and reloaded with each context switch. You need to find out which registers are used by the language you are using when you write the swap routine. The routines to start the system and to switch to the next task are prototypes that you change to fit your special application. The examples show how to use the task switching.

The program presented here is in several parts. There are three routines needed in your multitasking system. There are two HTPL routines, PROGRAM and NEXT, and one assembly language routine SWAPTSK. These are shown in Listing number 1. The versions here are for a system that has three tasks. The main program gets things started. NEXT changes to the next task.

SAMPLE CODE

```
;------- HTPL CODE FOR MULTITASK      (
;------- SWAP TASK  ( OLD NEW -- )
SWAPTSK MOVE.L  (A4)+,A1        ;NEW POINTER
        MOVE.L  (A4)+,A0        ;OLD POINTER
        MOVE.L  (A7)+,A3        ;RETURN
        MOVE.L  A4,(A0)+        ;SAVE PARAM PNTR
        MOVE.L  A7,(A0)         ;SAVE STACK PNTR
        MOVE.L  (A1)+,A4        ;NEW PARAM PNTR
        MOVE.L  (A1),A7         ;NEW STACK PNTR
        JMP     (A3)            ;RETURN TO CALLER
;-------------------------------------------------


( HTPL NONPREEMPTIVE MULTITASK )

( these must be contiguous and in order )
long tsk0p tsk0r tsk1p tsk1r tsk2p tsk2r tsk3p tsk3r ;
long tskpn tskid ;

program
  512 allocate 512 + !tsk1p
  512 allocate 508 + #task1 !4 +4 !tsk1r
  512 allocate 512 + !tsk2p
  512 allocate 508 + #task2 !4 +4 !tsk2r
  512 alloacte 512 + !tsk3p
  512 allocate 508 + #task3 !4 +4 !tsk3r
  #tsk1p !tskpn    1 !tskid
  #tsk0p @tskpn swaptsk ( start system )
  return
  end

proc next
  @tskpn #tsk0p swaptsk    ( save old task )
  @tskid +1 dup !tskid
    if 3 > then 1 !tskid #tsk0p !tskpn end
  @tskpn 8 + !tskpn
  #tsk0p @tskpn swaptsk    ( start new task )
  end

( ---- user interface ---- )
proc task1
  repeat
    repeat next chkchr until
    upcase case
      [ 'A' ]  ( action for command A here )
      [ 'B' ]  ( action for command B here )
      [ 'C' ]  ( action for command C here )
      end
    false until
  end

( --- timed sequence task --- )
proc task2
  while true do  next
    gettime !now ( get current time )
    if @now @target >= then
      doaction ( do timed action )
      @target @interval !target end ( set new time target )
  end end

( --- generalized task --- )
proc task3
  ( any repeating loop goes here )
  end
```

SWAPTSK is used by next to change tasks.

The first part of the system is PROGRAM where the HTPL program will start executing. There needs to be a startup routine that gets each task ready to run. It will only execute once when the program starts. Its purpose is to initialize the tasks, the tables used to switch tasks, and start the first task. Because HTPL uses two stacks, space for the stacks must be allocated. The main program allocates space for the parameter stack for each task. Next the return stack space is allocated. The starting address for the task is placed at the top of the return stack that has been allocated. To start the process going the program calls SWAPTSK with a pointer to TSK0P where its own parameter stack pointer and its return pointer will be saved. The other pointer is to where it will find the parameter stack pointer and return stack pointer for the first task. When the return statement is executed then the first task starts.

The second part of the system is the routine that transfers control from one task to the next task. This routine, which we call simply NEXT, looks like an ordinary procedure call to the task giving up control. The difference is that it doesn't use any arguments and it doesn't return any results. The NEXT routine uses SWAPTSK twice, once to get into the scheduler and once to get out of it. If the task stack is used for the scheduler routine then it would be possible to rewrite this to use SWAPTSK only once. If there are routines that use variables and are used by more than one task, then the values of the variables should also be swapped. It is important to realize that this is a generic prototype routine that needs to be customized for the specific set of tasks for which it will be used.

The final and smallest part is SWAPTSK (swap task), a routine that needs to be added to the runtime library HTPLRTL or linked in as an include file. The SWAPTSK is in assembler because it is much easier to do that way. The first job of the SWAPTSK routine is to save the state of an HTPL routine. In this context, only the parameter stack pointer and the return stack pointers need to be saved. The routine takes two parameters. First is a pointer to a place to save the old task parameter pointer and return pointer. Second is a pointer to a place to get the parameter stack and return pointer for the new task. The SWAPTSK routine returns to the routine that called it. This leaves the return address of the routine that called NEXT as the return address on the old return stack. Because all the tasks are written together in a single HTPL program all the other registers are either the same or do not need to be saved because all computations are done on the stack.

Using the Multitasker

Each task in the multitasking system needs to call NEXT at regular intervals to let the other tasks have some time to run. Most programs have a major wait loop where they get a character or wait for time to pass. If there is no wait loop, then any place in the main loop of a routine will do. It is also important to be aware of approximately how much time is used for each routine. Try to keep the time between calls to NEXT short and uniform. In many cases it is neccessary to fine tune the tasks that are running by changing where NEXT is called to get a smoother running system.

The check character procedure (chckchr) is an example of how a procedure that returns a variable number of arguments is used.

repeat next chckchr until

If there is a character available, the character and a true indicator are returned. If there is no character available then a false is returned. This causes the loop to continue until a character is available which it returns on the stack. This allows processing to occur in the background while waiting for the operator to input a character. After the input character is processed, then the wait loop can be entered to wait for the next character.

In most cases there is no need to wait to output characters. The BIOS on the HT-68k has a 256 character output queue. If this doesn't become full then there is no wait for output because the operating system merely places characters in the output queue. If there are several tasks running and a lot of characters need to be sent out, then a small number should be sent each time the task sending the characters is active. This allows the other tasks to run while K-OS ONE sends the characters from the output queue under interrupt control. If a larger buffer is needed then the BIOS

(Continued on page 28)

Software Timers for the 68000

By Joe Bartel, Hawthorne Technology

In most process control projects, and many interactive programs, there is a need to sequence and time operations. These are some techniques I have developed over the years and have converted into 68000 assembler code. A similar method will work with almost any other processor.

/m/mFor most projects there is a trade off between more software and more hardware to accomplish a given task. Using more software means the unit cost will be lower, but it also means there may be more developement costs involved. Software needs less board space than hardware does and it uses much less power. The purpose of a standard software item is to provide the same convenience that an LSI part like a UART or PIA does. You can use them and you don't have to design the parts each time. Also by reusing the same parts you can get more done with less effort and fewer bugs.

If the time interval is very short, less than a millisecond, then it is hard to use software instead of another hardware timer. If the time interval is longer than 100 msec and the resolution is 10 msec then there is no reason to use hardware. With a software timer you can have as many timed intervals as you want, but you only pay for one hardware timer. The internal timer on the 68681 used in the HT68k is an example of this.

When doing timing I like to divide the servicing of the timer interrupt from the actual action that is performed. The goal is to spend as little time as possible inside the interrupt service routine. This lessens the chance of losing an interrupt and makes more time available for the time critical services like serial input devices. To do this I use flags to indicate that an operation is to be performed. A non interrupt loop tests the flags and does the operations indicated.

```
IN LINE CODE:
INCT1   MOVE.W    COUNT1(A5),D0   ;TEST OLD
        BEQ.S     INCT2           ;IGNORE IF Z
        SUBQ.W    #1,COUNT1(A5)   ;DEC COUNTER
        BNE.S     INCT2           ;NO ACTION IF NZ
        ST        FLAG1+1(A5)     ;SET FLAG IF Z
        MOVE.W    LOAD1(A5),COUNT1(A5)    ;RELOAD COUNTER
INCT2   RTS

SUBROUTINE
        LEA       COUNTER1(A5),A0  ;POINT TO STRUCTURE
        BSR       SOFTCOUNT
        LEA       COUNTER2(A5),A0  ;#2
        BSR       SOFTCOUNT
        LEA       COUNTER3(A5),A0  ;#3
        BSR       SOFTCOUNT
        RTI
```

A soft counter is defined as a data structure in memory which has a counter that is counted down each time a timer interrupt occurs. There is a reload value to reload the count after it is counted down, and a flag that is set when the count has been counted down to zero. This is a flexible arrangement that can be used for one shot timers, continuous timers, or other timer types.

First, clear the flag to indicate that the timer is about to start. Then if it is to be continuous, set the reload value. If the time is to be a one shot timer, then the reload value should be zero. To start the timer, set the count to a nonzero value. The timer will then count down on each time tic until it reaches zero. At that time it will set the flag. To find out if the time has expired you check the flag at convenient intervals. Once the soft timer has been set and is runniing it will continue in the background almost like a hardware timer. For long time intervals in the millisecond or second region this is a very low cost way to have many timers. The single hardware timer on the ht68k can be used in this manner to provide a variety of timed intervals.

A common method of using the timer flags is to have a single very large loop for the main program. The loop is a series of tests of the various flags that have been defined. If a flag is set, the action routine that it was timing is done, and the flag is cleared. For physical events the accuracy will be good enough in most cases. As more timers are added, then the accuracy of the timing for each event may be less then when there are few timers. One approach to this part of the problem is to have the most time critical event flags at the start of the flag test loop. Instead of testing the next flag, a routine that takes a long time to execute can go back to the top of the test loop. This would insure that only one long routine gets executed each time through the scan loop. The same loop can also respond to flags set as the result of other kinds of interrupts or events.

An example of using a software timer is in a communication program. When a reply is expected the count value can be set to the maximum amount allowed, the flag cleared, and the reload value set to zero. The flag is tested and if any interval causes it to time out then the flag will be set. This can be used to abort a call if there has been no activity for a certain time period.

An example of using a periodic counter is a logging or reporting system. The counter and the reload value are set to the log interval. The flag is checked very often. When the flag is found set, a log record is generated and recorded. After the logging is done the timer flag is cleared and the wait for the next time proceeds.

A series of sequential events can be controlled by a single timer if a state variable is used, or a series of times can be used. When the flag for the first timer is set, the action for that flag is done. Then the flag for that event is cleared, and the timer for the next event in the sequence is started. Eventually the time for the first event is started and the cycle starts all over again. In this case each timer is acting like a one shot that triggers the next one shot in the series.

This has been a short discussion of how to time various events with only a single hardware timer. The techniques will work on any processor but are especially well adapted to the 68xxx family. With this type of programming it is possible to trade the faster speed of the 68000 for less complex, less expensive hardware. Also by having a simple generic timer routine that can be inserted like a hardware block it is possible to create larger, more complex programs without greatly increasing the time needed to debug the routines involved. By having the action part of the routine separated from the timer part, it is also easier to get less overall variation in timing, and almost eliminate time drift. ■

(Continued from page 5)

We share his view of what is needed; that is to say a Z-System running 64180 based card or cards for industrial control. Our company has, we think, done that and more. We have built a two-card set for the STE (IEEE P1000 standard) bus. The base card has CPU, math processor, Real Time Clock, and bus interface. The second card has ¼M or 1M DRAM, FDC, SCSI and Centronics i/f.

The main card is designed for use as a lower-cost delivery vehicle for installation in applications. The two card set has a mode in which it will run all SB180 software, including stuff which does direct I/O (not even via the BIOS!) The only non-SB180 compatible things are the Centronics i/f and lack of 8" disk support.

The only way in which we have failed Mr. Nelson is that we are using the wrong bus! The STE bus is an asynchronous bus using two-part connectors and with a number of advantages over STD. However there are more STD users outside England than STE users.

We have long considered an STD version of our system; the changeover would not be too difficult as STD cards are larger, and the STD bus simpler than STE. Does anyone else out there want to encourage us to do so? We would be interested to hear from potential users or distributors.

In addition to Z-Systems our cards also run a control BASIC, with programs ROMable, so non-Z-System people at least have an alternative.

David G. Collier, MA
J.B. Designs & Technology Ltd.
15, Market Street
Cirencester,
Glos. GL7 2PB
ENGLAND

Needs Northstar Help

I have and use an Ampro Little Board, two Heath H89s with hard sector, and three Northstar Horizons with hard sector. I would like to see anything on the Horizon, as I have little documentation on them.

Maybe you could provide a good neighbor helper list like Echelon has with addresses of individuals willing to share some expertise on some of the antiques that are still strong. For a specific, one of the Horizons had a fatal HD crash. It is gathering dust because I don't know how to format and initialize an RD7 Northstar hard disk environment.

G.N.

Editor: We have had several requests for Northstar help, and G.N.'s suggestion regarding a helper reference list (SMUG also has a similar list) would be part of our helping each other by sharing our knowledge. If you are willing to have your name and area of expertise either published in TCJ or available in the office on a 'on request only' basis, send us the information and we'll act as a clearinghouse.

If you have specific questions, send us the questions with permission to publish with your name and address for direct reply from our readers.

HTPL & Turbo C

HTPL is a great idea since I program in MOD2, F83 Forth, and 80186 Assembly (X16B). I got Hawthornes 8086 Assembler — nice, simple, elegant.

I'm looking forward to issue #30, especially more about the Tiny Giant 68000. I've got the K-OS I, but not the board (yet).

I received Turbo C® version 1.0, and the Integrated Environment serves the wants of Turbo Pascal Users' interface — a real improvement. Paid $59 dollars for it. The surprise was inside the front cover: $295 for source code to the Run Time Library. Adding the $150 toolbox for Turbo C, the compiler/environment is a little over $500, the price of Aztec C commercial version. Ouch! But Turbo C is good for a beginning C programmer like myself.

R.S.

Miscellaneous Reader Comments

I'm using a Sanyo MBC555 series with various add-in boards, and have a Taiwan clone that's hardly used. What I'd like to see in TCJ is do-it-yourself stuff, 680X0, 32XXX, TI DSP, RISC, etc.

I'm engulfed in MS-DOS stuff in all the other magazines, but unfortunately much of it is superficial.

I'm interested in S-100, SCSI, Z80 upgrades, ZSystem, TurboDOS, Hardware Interfacing, RBBS Systems (ZNode SYSOP).

I use Osbornes, Xerox 820s, a Morrow MD-11, and SB180 w/ETS I/O expansion, and am currently playing with a National Single board 32332. I like your 8-bit OS articles and your embedded processor articles especially. The series on SCSI was particularly helpful. I would like to see more coverage of National's 32000 series.

I'd like to see articles on the 68000, 8748, Robotics Controllers, C, and Forth.

I'm using Tandy M100, Sharp PC1600, SB180/ZRDOS, and Macintosh. Continue your ZRDOS articles and industrial applications articles. If you can get any pocket computer or Z280 articles, I'll read them. I think a nice project would be a HD64180 replacement for the M100 8085, a drop-in-the-case board with M100 features would be so nice.

I use software but hardware is my main concern. I like articles on the hardware including particular address location of devices (if a standard or how to access them if not) and any problem areas in interfacing.

Please encourage Mr. Rose for more S-100 projects. How about a SCSI S-100 design?

I understand that a 1.2Meg 5/o" floppy can be controlled via an 8" disk controller. How? ■

A Jay Sage Fan

Have a Kaypro 4-84 to which I have added a 20 Mb hard drive (Advanced Design Eng.) and personalized ZCPR3 system. My wife and I recently purchased a 6/10 MHz AT compatible clone for use with our Arabian horse breeding business. While I am in the process of learning something about MS-DOS, my first love, and greatest interest, is in the 8-bit world. My interests are those of a hobbyist, so please don't forget about those of us out here who fiddle with 'puters just for fun and personal satisfaction.

R.B.

PS: Tell Jay Sage I really appreciate his column!

An S-100 User

I am a 36 year old Electrical Engineer with an MS and a BS. I run the following machines at home:

(Continued on page 46)

Lillipute Z-Node

A Remote Access System for TCJ Subscribers

by Richard Jacobson

Lillipute Z-Node is the new Remote Access System (RAS) for The Computer Journal. This arrangement had its birth a little over a month ago when Art Carlson told me he could no longer run the TCJ RAS in Kalispell, Montana because of difficulties with the rural phone lines. At the time, I was on vacation in Anchor Bay, California, population three-hundred and fifty, and couldn't really tell if the crackle was coming from Art's end or mine. It occurred to me that Lillipute Z-Node has been blessed with a fine, central location in Chicago. So I suggested to Art that we open a TCJ directory on Lillipute and give TCJ subscribers access to the new area plus three others already open to NAOG/ZSIG members. Art thought it was a great idea. With a little bit of luck the TCJ area should be up and running by the time you read this and TCJ subscribers will have access to four directories on each of the two computers that run Lillipute Z-Node.

Lillipute Z-Node is a dial-up computer system, a 144 megabyte database for sophisticated users of the Z-System and CP/M. It runs on two Ampro Little Board Plus computers. Each Ampro is housed in an Integrand cabinet, along with a 96 TPI floppy disk drive, a 48 TPI floppy disk drive, and a 72 meg full-height hard drive. The computers are hooked up to a pair of U.S. Robotics Courier 2400 modems. The operating system consists of ZCPR33, ZRDOS Public Plus, and Ampro BIOS 3.8. Messages are handled by Echelon, Inc.'s and Tim Gary's Z-Msg. Both systems use BYE510 to establish and maintain the remote linkup. KMD22 handles the file transfers. The hardware and software have undergone substantial evolution over the two and one-half years of Lillipute's operation.

The system went on line for the first time in March 1985 as Z-Node #15, one of over 80 Z-Nodes around the world dedicated to users of the Z-System (ZCPR3, ZCPR33, and ZRDOS) and CP/M. It started its life as a single, 9 megabyte Kaypro 10, accessible at only 300 or 1200 baud. An enthusiastic and supportive membership has made it possible to expand to a 144 megabyte, two-computer, 300/1200/2400 baud, remote system offering a wide variety of technical advice and public domain software.

Lillipute Z-Node can be reached at 312-649-1730 (System 1) or 312-664-1730 (System 2). Both systems are accessible by PC-Pursuit. I encourage TCJ subscribers to call System 1 and System 2. Leave a message to "Sysop" (that's me) requesting special TCJ access. I will check your name against Art's subscription list. If you show up on that list, you will be given entry to the public domain software download directories on your next call.

When you log in for the first time, you will be asked for your name and for a password. It is best to use your first and last names, no middle initial. Also, make a note of your password for subsequent calls. You must log in separately for each system and leave a message for SYSOP requesting TCJ access.

TCJ subscribers will have access to four named directories, YOUNEED, ZSIG, SB180, and TCJ. The YOUNEED directory holds the minimum utility toolset required to take full advantage of Lillipute Z-Node. ZSIG contains all the programs approved by the ZSIG software librarian, Jay Sage, and his software committee. SB180 has programs, documentation, and bug fixes relating to the SB180, an HD64180-based computer sold by MicroMint. It will also contain all updates for Malcom Kemp's brilliant new operating system for the SB180, called XBIOS.

TCJ (the ZCPR3 named directory, not the magazine) will contain files uploaded by Art Carlson, including listings from TCJ articles. With your assistance, Art Carlson and I will be able to determine what direction to take with the TCJ named directory. We will need your feedback to accomplish the task.

An enormous amount of software is available on Lillipute. Turbo Pascal material takes up about 4 megabytes. Several megabytes are occupied by the modem programs Mex and Imp, along with their overlays. Far more space is devoted to ZCPR3, ZCPR33, and supporting documentation than to any other single group of files. One of the new sections is devoted to Modula-2, with almost a megabyte devoted to Wirth's "successor" to Pascal. Two of the most popular sections are WORDPRO and PRINTER, which, taken together, offer a significant support facility for Z-System and CP/M users whose interests are primarily in word processing.

Systems 1 and 2 have different flavors and different audiences. The group that hangs out on System 1 tends to be less technical in orientation than the group on System 2. System 2 has no "applications" programs. It is devoted entirely to operating system software, including 4 megabytes of software to install ZCPR3 or ZCPR33 on virtually any 8-bit computer. System 2 also has a named directory called OPSYS, devoted to non-ZCPR operating systems as well as BDOS replacements other than ZRDOS (which is not public domain). System 1 has the dBASE files, the Turbo Pascal programs, and the writer's aids. It also tends to attract more vigorous debate than System 2.

Should the TCJ subscribers' directories whet your appetite, you can opt for full membership, with access to the entire 144 megabytes on both systems. For full membership, send a check for $35 payable to Richard Jacobson, accompanied by pertinent information—name, address, work and home phone numbers—to:

Richard Jacobson
Lillipute Z-Node
1709 N. North Park Avenue
Chicago, Illinois 60614.

After I receive your membership fee and application, and after your initial login, I will set you for access to the entire 144 megabytes of software on both systems. ■

Back Issues Available:

Issue Number 1:
- RS-232 Interface Part One
- Telecomputing with the Apple II
- Beginner's Column: Getting Started
- Build an "Epram"

Issue Number 2:
- File Transfer Programs for CP/M
- RS-232 Interface Part Two
- Build Hardware Print Spooler: Part 1
- Review of Floppy Disk Formats
- Sending Morse Code with an Apple II
- Beginner's Column: Basic Concepts and Formulas

Issue Number 3:
- Add an 8087 Math Chip to Your Dual Processor Board
- Build an A/D Converter for the Apple II
- Modems for Micros
- The CP/M Operating System
- Build Hardware Print Spooler: Part 2

Issue Number 4:
- Optronics, Part 1: Detecting, Generating, and Using Light in Electronics
- Multi-User: An Introduction
- Making the CP/M User Function More Useful
- Build Hardware Print Spooler: Part 3
- Beginner's Column: Power Supply Design

Issue Number 8:
- Build VIC-20 EPROM Programmer
- Multi-User: CP/Net
- Build High Resolution S-100 Graphics Board: Part 3
- System Integration, Part 3: CP/M 3.0
- Linear Optimization with Micros

Issue Number 14:
- Hardware Tricks
- Controlling the Hayes Micromodem II from Assembly Language, Part 1
- S-100 8 to 16 Bit RAM Conversion
- Time-Frequency Domain Analysis
- BASE: Part Two
- Interfacing Tips and Troubles: Interfacing the Sinclair Computers, Part 2

Issue Number 15:
- Interfacing the 6522 to the Apple II
- Interfacing Tips & Troubles: Building a Poor-Man's Logic Analyzer
- Controlling the Hayes Micromodem II From Assembly Language, Part 2
- The State of the Industry
- Lowering Power Consumption in 8" Floppy Disk Drives
- BASE: Part Three

Issue Number 16:
- Debugging 8087 Code
- Using the Apple Game Port
- BASE: Part Four
- Using the S-100 Bus and the 68008 CPU
- Interfacing Tips & Troubles: Build a "Jellybean" Logic-to-RS232 Converter

Issue Number 17:
- Poor Man's Distributed Processing
- BASE: Part Five
- FAX-64: Facsimile Pictures on a Micro
- The Computer Corner

Interfacing Tips & Troubles: Memory Mapped I/O on the ZX81

Issue Number 18:
- Parallel Interface for Apple II Game Port
- The Hacker's MAC: A Letter from Lee Felsenstein
- S-100 Graphics Screen Dump
- The LS-100 Disk Simulator Kit
- BASE: Part Six
- Interfacing Tips & Troubles: Communicating with Telephone Tone Control, Part 1
- The Computer Corner

Issue Number 19:
- Using The Extensibility of Forth
- Extended CBIOS
- A $500 Superbrain Computer
- BASE: Part Seven
- Interfacing Tips & Troubles: Communicating with Telephone Tone Control, Part 2
- Multitasking and Windows with CP/M: A Review of MTBASIC
- The Computer Corner

Issue Number 20:
- Designing an 8035 SBC
- Using Apple Graphics from CP/M: Turbo Pascal Controls Apple Graphics
- Soldering and Other Strange Tales
- Build a S-100 Floppy Disk Controller: WD2797 Controller for CP/M 68K
- The Computer Corner

Issue Number 21:
- Extending Turbo Pascal: Customize with Procedures and Functions
- Unsoldering: The Arcane Art
- Analog Data Acquisition and Control: Connecting Your Computer to the Real World
- Programming the 8035 SBC
- The Computer Corner

Issue Number 22:
- NEW-DOS: Write Your Own Operating System
- Variability in the BDS C Standard Library
- The SCSI Interface: Introductory Column
- Using Turbo Pascal ISAM Files
- The AMPRO Little Board Column
- The Computer Corner

Issue Number 23:
- C Column: Flow Control & Program Structure
- The Z Column: Getting Started with Directories & User Areas
- The SCSI Interface: Introduction to SCSI
- NEW-DOS: The Console Command Processor
- Editing The CP/M Operating System
- INDEXER: Turbo Pascal Program to Create Index
- The AMPRO Little Board Column
- The Computer Corner

Issue Number 24:
- Selecting and Building a System
- The SCSI Interface: SCSI Command Protocol
- Introduction to Assembly Code for CP/M
- The C Column: Software Text Filters
- AMPRO 186 Column: Installing MS-DOS Software
- The Z Column
- NEW-DOS: The CCP Internal Commands
- ZTIME-1: A Realtime Clock for the AMPRO Z-80 Little Board
- The Computer Corner

Issue Number 25:
- Repairing & Modifying Printed Circuits
- Z-Com vs Hacker Version of Z-System
- Exploring Single Linked Lists in C
- Adding Serial Port to Ampro Little Board
- Building a SCSI Adapter
- New-DOS: CCP Internal Commands
- Ampro '186: Networking with SuperDUO
- ZSIG Column
- The Computer Corner

Issue Number 26:
- Bus Systems: Selecting a System Bus
- Using the SB180 Real Time Clock
- The SCSI Interface: Software for the SCSI Adapter
- Inside AMPRO Computers
- NEW-DOS: The CCP Commands Continued
- ZSIG Corner
- Affordable C Compilers
- Concurrent Multitasking: A Review of DoubleDOS
- The Computer Corner

Issue Number 27:
- 68000 TinyGiant: Hawthorne's Low Cost 16-bit SBC and Operating System
- The Art of Source Code Generation: Disassembling Z-80 Software
- Feedback Control System Analysis: Using Root Locus Analysis and Feedback Loop Compensation
- The C Column: A Graphics Primitive Package
- The Hitachi HD64180: New Life for 8-bit Systems
- ZSIG Corner: Command Line Generators and Aliases
- A Tutor Program for Forth: Writing a Forth Tutor in Forth
- Disk Parameters: Modifying The CP/M Disk Parameter Block for Foreign Disk Formats
- The Computer Corner

Issue Number 28:

- Starting Your Own BBS: What it takes to run a BBS.
- Build an A/D Converter for the Ampro L.B.: A low cost one chip A/D converter.
- The Hitachi HD64180: Part 2, Setting the wait states & RAM refresh, using the PRT, and DMA.
- Using SCSI for Real Time Control: Separating the memory & I/O buses.
- An Open Letter to STD-Bus Manufacturers: Getting an industrial control job done.
- Programming Style: User interfacing and interaction.
- Patching Turbo Pascal: Using disassembled Z80 source code to modify TP.
- Choosing a Language for Machine Control: The advantages of a compiled RPN Forth like language.

Issue Number 29:

- Better Software Filter Design: Writing pipable user friendly programs.
- MDISK: Adding a 1 Meg RAM disk to Ampro L.B., part one.
- Using the Hitachi HD64180: Embedded processor design.
- 68000: Why use a nes OS and the 68000?
- Detecting the 8087 Math Chip: Temperature sensitive software.
- Floppy Disk Track Structure: A look at disk control information & data capacity.
- The ZCPR3 Corner: Announcing ZCPR33 plus Z-COM Customization.
- The Computer Corner.

TCJ ORDER FORM

Subscriptions		U.S.	Canada	Surface Foreign	Total
6 issues per year ☐ New ☐ Renewal	1 year	$16.00	$22.00	$24.00	
	2 years	$28.00	$42.00		
Back Issues		$3.50 ea.	$3.50 ea.	$4.75 ea.	
Six or more #'s		$3.00 ea.	$3.00 ea	$4.25 ea.	
				Total Enclosed	

All funds must be in U.S. dollars on a U.S. bank.

☐ Check enclosed ☐ VISA ☐ MasterCard Card # _____

Expiration date_____ Signature _____

Name _____

Address _____

City_____ State_____ ZIP _____

The Computer Journal

190 Sullivan Crossroad, Columbia Falls, MT 59912 Phone (406) 257-9119

News

Software Development Conference

The Editors of *Computer Language, AI Expert, Unix Review* and *Database Programming & Design* recently announced completion of the agenda for Software Development '88, a three day conference which will take place February 17-19, 1988, in San Francisco, at the Ramada Renaissance Hotel off Union Square.

The conference is being organized to provide practical information to professional software developers. Six to eight hundred developers are expected to participate.

As many as sixty speakers will present ninety lectures and workshops over the three days. Among those confirmed as speakers are: Bill Gates, Dick Gabriel, Ester Dyson, P.J. Plauger, Gary Kildall, Jack Purdum, Alan Kay, Charles Moore, John Warnock and Philippe Kahn. Topics ranging from Artificial Intelligence and Design Methodologies to Languages, Data Base Design and Graphics will be covered.

In addition to lectures and technical workshops, there will be an exhibit by industry suppliers. Those interested in attending or exhibiting should contact KoAnn Tingley, Conference Coordinator, Seminar Division, Miller Freeman Publications, 500 Howard Street, San Francisco, CA 94105. Phone: (415) 995-2426. FAX: (415) 543-0256.

Hyperspace DOS

Hyperspace Z-System is a CP/M 2.2 compatible disk operating system, for HD64180 and Z280 compatible systems, which includes ZCPR 3.3, ZRDOS 2.0, and sample BIOS for 64180 machines.

The Hyperspace Z-System features a large 57.25K free memory space even with a full featured ZCPR 3.3 implementation with RAM disk and hard disk support. The system may optionally be reconfigured to provide 61.25K of free memory for applications.

The large free memory space is provided by placing the BIOS and ZRDOS modules outside of the normal 64K logical address space, and many programs, such as dBase II® , Wordstar 4.0® , and Turbo Pascal® , have been run perfectly under Hyperspace.

Hyperspace is available, including the sample BIOS, for $195, and OEM terms are available. Contact Echelon, Inc., 885 North San Antonio Road, Los Altos, CA 94022, phone (415) 948-3820.

Programming/Interfacing Catalog

Group Technology, Ltd., has announced their Fall 1987 catalog of their new and current books and products, and it is directed particularly to those who wish to gain or teach hands-on experience in interfacing external devices to microcomputers.

New hardware items include the DAB-1, Data Acquisition/Output Board for the TRS-80 Color Computer, which turns the CoCo into an economical controller and data acquisition system which is excellent for learning, teaching, and implementing real world control applications. Some current applications include digitizing and analyzing a torsion beam, driving a telescope for automatic selection and tracking of a celestial body, controlling the iris of a particle beam accelerator, and temperature monitoring and control.

A new addition to the scientific software is SEQS(PC), Simultaneous Equation Solver, for the MS-DOS personal computers, which provides fast, versatile, and easy-to-use capability for solving nonlinear or linear algebraic equations.

New books on microcomputer interfacing, and assembly language programming supplement older standbys and include special emphasis on artificial intelligence, expert systems, and Forth language.

Science instructors, practicing scientists, microcomputer hobbyists, and anyone interested in improving professional affairs will find the catalog a valuable resource. The catalog is available at no charge from Group Technology, Ltd., 6925 Dogwood Road, Baltimore, MD 21207-2606, phone (301) 298-5716,

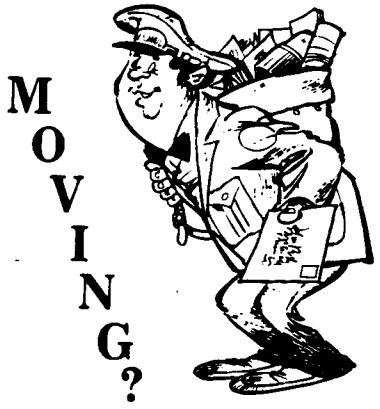
---

Logic Simulation Program

BV Engineering has just released a digital logic simulation program for IBM PC compatibles and the Macintosh computer. After describing a logic circuit and sequence of binary input signals to the program, LSP will compute the resulting binary output signals at any (or all) nodes of the circuit at the specified times.

LSP contains built in models for combinatorial gates such as AND, OR, NAND, etc., sequential devices such as D, JK, and toggle FLIP-FLOPS, as well as TRI- STATE devices. LSP handles all time scheduling and accurately propagates the input and computed output results through the design regardless of the complexity or nesting of feedback loops. The output of LSP is a timing diagram, or truth table, showing the binary states of each selected signal as a function of time. There are many other features, and LSP is completely compatible with BV Engineering's PCPLOT and PDP for plotting and graphics.

Contact Peter G. Boucher at BV Engineering, 2200 Business Way, Suite #207, Riverside, CA 92501, phone (714) 781-0252.



Make certain that TCJ follows you to your new address. Send both old and new address along with your expiration number that appears on your mailing label to:

THE COMPUTER JOURNAL
190 Sullivan Crossroad
Columbia Falls, MT 59912

If you move and don't notify us, TCJ is not responsible for copies you miss. Please allow six weeks notice. Thanks.

Books of Interest

68000 Assembly Language Programming
Second Edition
by Leventhal, Hawkins, Kane, and Cramer
Published by Osborne McGraw-Hill
1986, 6½ × 9", 484 pages

When learning about a new CPU, the first book I buy is Leventhal's assembly language volume, because I prefer a book which concentrates on the CPU rather than one which spends most of the space on operating system specific interfacing details for some chosen system — first I want to learn the CPU, after that I'll learn the operating system.

This book is similar to the Z-80 volume, but the design has been improved making it easier to read, and there is more discussion and explanation with the programming examples. This second edition includes information on the 68010 and 68020.

The contents are as follows: • Section I: Fundamental Concepts — Introduction to Assembly Language Programming; Assemblers; MC68000 Machine Architecture.

• Section II: Introductory Problems — Beginning Programs; Simple Program Loops; Character Coded Data; Code Conversion; Arithmetic Problems; Tables and Lists.

• Section III: Advanced Topics — Parameter Passing Techniques; Subroutines; Advanced MC68020 Addressing and Instructions; Connecting to Peripherals; Exception Processing; Interrupts and Other Exceptions.

• Section IV: Software Development — Problem Definition; Program Design; Documentation; Debugging; Testing.

• Section V: MC68000 Instruction Set — Descriptions of Individual MC68000 Instructions.

• Section VI: Appendices — Alphabetic Listings of Instructions; Numeric Listings of Instructions.

If I were limited to one book on the 68000 this one would be my choice, because I can sit down with it and an assembler and work my way thru the many examples. As they say on page 53 "The only way to learn assembly language is to work with it." A very small sample of the example problems are: byte disassembly, table lookup, word assembly, normalize a binary number, replace leading zeros with blanks, pattern match, hexadecimal to ASCII, decimal to seven-segment, BCD to binary, binary number to ASCII string, add entry to list, check an ordered list, remove an element from a queue, factorial of a number (reentrant and recursive), scaled indexes....

This Fall I intend to sit down with this book and Hawthorne's 68000 Tiny Giant, and learn all about 68000 assembly language.

Suggested Reading

The following books, which are in our queue for review, appear worthwhile. More complete reviews will follow, but this preliminary listing is to let you know that they are available in case you have immediate needs for them.

Your reviews and/or comments on specialized technical books are welcome, along with suggestions on useful books which should be inculded in future reviews. We intend to limit our reviews to advanced or specialized limited distribution works, and not to duplicate what is being already covered in the mass market magazines.

File Formats
For Popular PC Software
by Jeff Walden
Published by Wiley Press
ISBN 0-471-83671-0

Listed as 'A Programmer's Reference' this book is part of Wiley's IBM PC Series. It includes information on the disk file structure for Lotus 1-2-3, Symphony, Ability, dBase, Data Interchange Format (DIF), MultiMate, MultiPlan and the SYLK File Format, IBM Plans+, SuperCalc3 and the Super Data Interchange, VisiCalc, WordStar v2.2-3.31, and WordStar 2000./1/r I found the DIF very helpful when converting a mail list received on 9 track tape, and I wish it had information on the Condor file structure. I understand that there is a second volume out with information on additional programs.

Computer Numerical Control
byJoseph Pusztai & Michael Save
Published by Reston (Div of P-H)
ISBN 0-8359-0924-7

This book is apparently intended for those involved with numeric machining, and includes a lot of information on milling, drilling, turning, calculating cutter centerline distances, etc. The information looks complete, and this will be part of my permanent library.

Engineering Foundations of Robotics
by Francis N-Nagy & Andras Siegler
Published by Prentice-Hall International
ISBN 0-13-278805-5

A high-level book with an overview of robotics stressing the underlying mathematical techniques and principles relating to the kinematic models of robot manipulators. Definitely not hobby level, it includes topics such as *Description of a Wedge by Transformation Matrices, Transformations Along the Kinematic Chain, Gripper Positioning by Euler Angles for Roll-Yaw-Roll Geometry,* and *Servo Systems for Robot Control.* There is a lot which I don't understand, but this book is on my required reading list.

Multiple Processor Systems
for Real-Time Applications
by Burt H. Liebowitz & John H. Carson
Published by Prentice-Hall
ISBN 0-13-605114-6

The purpose of this book is clearly stated in the preface which says in part, "This book presents a pragmatic approach to the design and construction of a class of computing system we call the *locally distributed multiple processor system (MPS).* These systems consist of closely located independent CPU-memory pairs linked together by a communication subsystem." Another book for my required reading list.

Writing MS-DOS Device Drivers
by Robert S. Lai — The Waite Group
Published by Addison-Wesley
ISBN 0-201-13185-4

A high-level book with a lot of very good information about a very important subject. The book is well written, and well organized, contains useful working programs as examples (disk available for $10), and the necessary information on BIOS interrupts.

How to Write a Useable User Manual
by Edmond H. Weiss
Published by ISI Press
ISBN 0-89495-052-5

Writing is an art, and writing user manuals is a very special art! If you want to write a decent manual, you have to know more than just how to program, and this book has the needed information on the often neglected aspects of writing a user manual.

Data Compression, Second Edition
Techniques, Applications, Hardware, & Software Considerations
by Gilbert Held
Published by John Wiley & Sons
ISBN 0-471-91280-8

Data compression is very important for data communications, and this book provides information on how to select the optimum method for different applications.

Memory Resident Programming on the IBM PC
by Thomas A. Wadlow
Published by Addison-Wesley
ISBN 0-201-18595-4

Covers how to write your own TSR (Terminate and Stay Resident) programs, and includes a very good section on the IBM ROM BIOS Services. I've always wanted my own special TSR monitor/debugger, and this book shows how. According to the back cover, a disk is available from Addison-Wesley. Looks like a good combination with the device driver book above.

C Programmer's Guide to Serial Communications
by Joe Campbell
Published by Howard W. Sams
ISBN 0-672-22584-0

A large book (655 pages!) which answers most of the questions I had about baud rates, UARTs, CRC, XMODEM, interrupts, and much more. My only complaint about this book is that it wasn't here three years ago when I started digging out the facts about serial communications. Now, I'll read it cover to cover to fill in the gaps in what I picked up elsewhere.

Classified

WANTED — Dietary and Nutritional software and/or books, papers, and resource information, for use on an IBM PC in a public institution. Low cost or public domain software with source code preferred, as they have an experience programmer available to modify and customize the programs. The Computer Journal, Box T, 190 Sullivan, Columbia Falls, MT 59912, phone (406) 257-9119,

WANTED — S-100 Boards and/or documentation. Contribute the S-100 material which you are no longer using to the TCJ S-100 resource archives. Send material to The Computer Journal, 190 Sullivan, Columbia Falls, MT 59912, phone (406) 257-9119

WANTED — WordStar Version 3.X for generic MS-DOS, which supports ASCII terminals. Now that you've upgraded to Version 4.0, get rid of your old no-longer-used version. Contact The Computer Journal, 190 Sullivan, Columbia Falls, MT 59912, phone(406) 257-9119

Reader's Feedback

(Continued from page 40)

• PCXT 8MHz clone with EGA "Wonder", RLL disk controller, and 2 Meg board.
• S-100 bus Z80 system with 8" drives.
• Single board wire wrapped Z80 system with 8" drives.
• S-100 bus 64180 system with 128K RAM, 8" drives, and 24 Meg 14° Winchester.

I want to see someone write a non-preemptive task switching CP/M system, so you can have the editor and compiler in RAM always, and just switch between them as needed. I'd write it, but I don't have the time.

Anyway, I'm glad you folks are still publishing for the CP/M audience. But don't get dreary about it, like S-100 Journal.

B.D.

Editor: What we publish will depend on what we hear from our readers. Send us a list of the subject matter for 4 or 5 articles which you would like to see published — or better yet send us your article to publish. Knowledge is for sharing, but sharing means that you have to participate! ■

(Continued from page 48)

skill needed to actually perform the tasks requested of him.

The end result of the whole affair is a lot of people feeling taken for money that was very hard to come by. The system is causing almost as many problems as it is solving. Several people are ready to go back to the old system because they have had so many problems. Overall they would have little chance of surviving the computer's problem, if it wasn't for my friend practically giving his time away to them. For me it points up so many negative aspects of where the computer industry is at present.

Up until two or three years ago the industry either used computers within themselves or you had big companies supplying you with a guru computer whiz. The success of the PC sellers in convincing users that anyone can use them, and especially program them, is just now being felt. The previous example is a good picture of the so called blind leading the blind. For me doing business with companies it presents a special problem. Before a company should buy any system, their projected problems, choices and solutions need to be fully outlined. I find most businesses buying then asking later "will it do what we need done?" When the topic of training or using consultants comes up, the bottom line is cost, and that usually means the lowest bidder. Now when we talk about lowest bidders, I usually remind people that the shuttle booster rockets are made by the lowest bidder, and we all know what happened to the astronauts in that case. Industrial computers can kill too, but typically they will just destroy your data and eat up your money.

Summing this all up before my battery dies (I am writing this 15 miles from any power on my Z171) is that few people understand the need for education. In studying how school systems work, I can now see how little we as a nation consider education in anything we do. School children are typically taught less than they really need every day in America, and then our businesses continue on by asking untrained workers to do jobs requiring high skill levels. For me to make a living in training, our country and our business leaders will first need to become trained before they start appreciating training. Over the years I have seen how important training is and yet our businesses are down playing it more and more all the time.

Well at least there is one place you can still get good training on how to do things and that is The Computer Journal. Hopefully next time around I will start with some training on how to bring up 68K HTPL. So till next month keep on computing (that is if the batteries hold out long enough). ■

Registered Trademarks

It is easy to get in the habit of using company trademarks as generic terms, but these registered trademarks are the property of the respective companies. It is important to acknowledge these trademarks as their property to avoid their losing the rights and the term becoming public property. The following frequently used marks are acknowledged, and we apologize for any we have overlooked.

Apple II, II + , IIc, IIe, Macintosh, DOS 3.3, ProDOS; Apple Computer Company. CP/M, DDT, ASM, STAT, PIP; Digital Research. MBASIC; Microsoft. Wordstar; MicroPro International Corp. IBM-PC, XT, and AT; IBM Corporation. Z-80, Zilog. MT-BASIC, Softaid, Inc. Turbo Pascal, Borland International.

Where these terms (and others) are used in The Computer Journal they are acknowledged to be the property of the respective companies even if not specifically mentioned in each occurence.

Advertiser's Index

TCJ is User Supported

If You Don't Contribute Anything....

....Then Don't Expect Anything

THE COMPUTER CORNER

A Column by Bill Kibler

Well, summer is coming to an end and with it was the seasonal jaunt to Bend Oregon for Micro Cornucopia's SOG (Semi Official Get-together). This year most of The Computer Journal's staff was present and we all had considerable fun talking and planning those next articles. The entire affair had a different feeling to it this year, not so festive but a bit more serious. After the dinner on Saturday night everybody got into talking and planning. I went to bed at 3 a.m.but some never made it at all.

The friends at Hawthorne were there, Joe gave two talks and Marla gave one, both very well received. They have sold plenty of 68K boards and operating systems. They are not getting rich, but then they never planned on it. I had the feeling that everyone felt the success of the HTPL and HT68K were going just about as expected, and this next year should see some major changes in their favor. One person I was talking to had a STD bus system running an 8086, but the overall performance was rather poor so they have switched to a 68K CPU board.

This change in design makes me feel the reason for the low feelings at the SOG, were feeling that the PC Clones had just about destroyed the computer industry. I think many peoplehave resigned themselves to using the inferior product to do jobs more ideally suited to other systems. A few people who don't really need to stick to the Intel product line, have changed over to the 68K line and are much more optimistic about their future. I know personally I have little desire to deal with the PC Clone market, and feel that it has been invaded by suede shoe used car salesmen.

If you have been following my columns for the last year, you will know that I was in a master's program. No more, it has been completed. This means two things, I can complete all those projects I put aside (and you have been waiting for me to complete) and I now know enough NOT to become a computer teacher in the school system. I did get what I wanted out of the program, which was learning how to teach computer sciences, as well as learning to hate the PCs completely. We used a lot of PC programs in the course, but not once did I feel those programs were better (or faster) than those I had used on CP/M, or the ST's GEMDOS.

Hopefully by the time you read this, Atari will have released the new MEGA ST and a more business like product will be on its way to changing Atari's image. I got into one discussion at the SOG just on that problem, the idea that Atari only makes game machines. Now don't get me wrong, the ST is one super great game machine, but let's see why. To be truly a good game machine you need speed, good graphics, ability to do animation, lots of memory (no paging please), and low cost. In several ways those items are available on plenty of machines, but to equal the standard design of the ST, a clone would cost almost twice as much. When looking at game ability, business use in many ways is rather simple or hardly taxing of the system. For industrial use the PC's operating system and to some extent the CPU design are not intended for multitasking real time event handling, like the 68K is.

While at the SOG Art and myself had several discussions with FORTH and STD bus users about writing articles for the Computer Journal. I have since talked with the person who changed to 68K CPU and he is going to try and explain why they switched. One reason why I like the STD bus is just that ability to change CPU cards while leaving all the rest of the cards intact. Now I know that some of the timing parameters were changed, but we will leave that for my friend to write about in detail. We had a number of discussions on FORTH in the STD bus environment and hopefully will get a few articles about that. These people felt they couldn't do it with any other language, and if we pester them, maybe we will get some articles giving us the insight into why it works so well.

Overall the SOG was not a high point of my trip. I enjoyed seeing the staff again and talking with friends (the high point was no masters work!). The HTPL group has promised me some new items, and so I got excited again about bringing up the HTPL on my S-100 system. We talked about having our own meeting next year if the SOG doesn't happen (dorms maybe unavailable, but Dave is looking for alternatives), somewhere around Portland and mostly aimed at the 68K user (if enough of you write we may have one before the SOG). There was considerable talk and I agree that TCJ could become the Z80, STD bus, and 68K users magazine. For me personally my future is cloudy and uncertain.

I have been trained to help people with the PC line of products, but I hate them. I had even more reason to dislike them after stopping at a friend's place on the way home. He has been helping a local co-op health clinic sort out their computer problems. They purchased a turn key computer system containing medical software. The people who had control of the money at the time did not buy under any contractual agreements. They basically got a system, software, a few hours of training, operators manuals, and a complex mess. They later discovered plenty of problems and hired my friend to help them out. They have since discovered how little they got for their money and how much more it may cost to keep the system running. The original plan was to use the consultant for help, but he is now charging $50 dollars an hour, which they can not afford, they also need to update the software package at close to $1000, and can't do that till they send someone to a training program which costs $400.

What this whole problem points out is the typical problems businesses are facing when they try to computerize. The first problem was allowing people who know nothing about computers to buy them. These people didn't use a contract that clearly stated what the system will do, what is included in the purchase, and what kind of support and for how long. The hidden cost might not have been listed, but experienced purchasers have learned to include at least one free update in their contracts. The agent or consultant was not checked for his track record, in fact this was the first and only such installation this person has done. The consultant really didn't have either the experience or

(Continued on page 47)