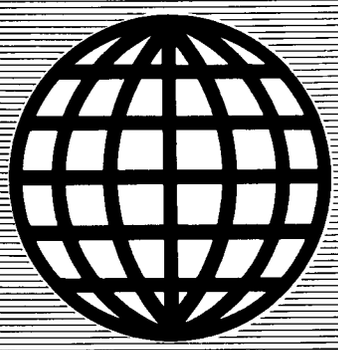


The COMPUTER JOURNAL®

Programming - User Support

Applications

Issue Number 33

$3.00

Data File Conversion

Writing a Filter to Convert Foreign Formats

Advanced CP/M

Z3PLUS & Relocation

SCSI for the S-100 Bus

Another Example of SCSI's Versatility

Use a Mouse on Any Hardware

Implementing the Mouse on a Z80 System

Systematic Elimination of MS-DOS Files

Part 2— Subdirectories and Extended DOS Services

ZCPR3 Corner

ARUNZ, Shells and WordStar 4.0

Data Base

A Data Base Primer

ISSN # 0748-9331

you see here.

THE AMPRO LITTLE BOARD™/286

Big power for smaller systems.
Little Board/286 is the newest member of our family of MS-DOS compatible Single Board Systems. It gives you the power of an AT in the cubic inches of a half height 5 1/4" disk drive. It requires no backplane. It's a complete AT-compatible system that's functionally equivalent to the 5-board system above. But, in less than 6% of the volume. It runs all AT software. And its low-power requirement means high reliability and great performance in harsh environments.

Ideal for embedded & dedicated applications. The low power and tiny form factor of Little Board/286 are perfect for embedded microcomputer applications: data acquisition, controllers, portable instruments, telecommunications, diskless workstations, POS terminals . . . virtually anywhere that small size and complete AT hardware and software compatibility are an advantage.

Compare features.
Both systems offer:

- 8 or 12MHz versions
- 512K or 1Mbyte on-board DRAM
- 80287 math co-processor option
- Full set of AT-compatible controllers
- 2 RS232C ports
- Parallel printer port
- Floppy disk controller
- EGA/CGA/Hercules/MDA video options
- AT-compatible bus expansion
- A wide range of expansion options
- IBM-compatible Award ROM BIOS

But only Little Board/286 offers:

- 5.75" x 8" form factor
- EGA/CGA/Hercules/MDA on a daughterboard with no increase in volume
- SCSI bus support for a wide variety of devices: Hard disk to bubble drives
- On-board 1Kbit serial EPROM. 512 bits available for OEMs
- Two byte-wide sockets for EPROM/RAM/NOVRAM expansion (usable as on-board solid-state disk)
- Single voltage operation (+5 VDC only)
- Less than 10W power consumption
- 0-70°C operating range

*AT is a Registered Trademark of IBM Corp.

Better answers for OEMs.
Little Board/286 is not only a smaller answer, it's a better answer . . . offering the packaging flexibility, reliability, low power consumption and I/O capabilities OEMs need . . . at a very attractive price. And like all Ampro Little Board products, Little Board/286 is available through representatives nationwide, and worldwide. For more information and the name of your nearest Rep, call us today at the number below. Or, write for Ampro Little Board/286 product literature.

408-734-2800
Fax: 408-734-2939 TLX: 4940302

AMPRO
COMPUTERS, INCORPORATED
1130 Mountain View/Alviso Road
Sunnyvale, CA 94089

Reps: **Australia**-61 3 720-3298; **Belgium**-32 87 46.90.12; **Canada**-(604) 438-0028; **Denmark**-45 3 66 20 20; **Finland**-358 0 585-322; **France**-331 4502-1800; **Germany, West**-49 89 611-6151; **Israel**-972-3 49-16-95; **Italy**-39 6 811-9406; **Japan**-81 3 257-2630; **Spain**-34 3 204-2099; **Sweden**-46 88 55-00-65; **Switzerland**-41 1 740-41-05; **United Kingdom**-44 2 964-35511; **USA**, contact AMPRO.

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

Contributing Editors
Joe Bartel
Bob Blum
Bill Kibler
Rick Lehrbaum
Bridger Mitchell
Jay Sage

The Lillipute Z-Node sysop has made his BBS systems available to the TCJ subscribers. Log in on both systems (312-649-1730 & 312-664-1730), and leave a message for SYSOP requesting TCJ access.

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 33

Columns

Editor's Page

Small Companies—part2

In the last issue, I mentioned that the small software companies were in trouble. That situation has worsened during the past two months, and it is not just the software companies. The hardware and publishing (book and magazine) portions are also feeling the heat. The past two years has seen a tremendous surge in PC clone sales, but now there are an expanding number of vendors battling over a shrinking market.

I have been told that *80Micro has published their last issue, after almost 10 years in the business. Byte* is in its fifth year of declining page counts. As the troubled companies cut their advertising expenditures, I expect more of the commercial magazines to fold or make major reductions in their size. Perhaps *Byte* is in a stronger position because of its gradual reduction; some of the other fat magazines may not survive a sudden reduction. I have repositioned TCJ as a user supported journal, with a minimum of advertising support from the vendors seriously interested in our market. We are small, we don't have fancy color, and we can only pay our authors a small honorarium; but we publish information that you won't find in the commercial magazines—and we are planning on expansion.

The book publishing field is being taken over by three or four large publishers and a few large bookstore chains. The chains prefer to deal with as few publishers as possible, and the large publishers prefer to deal with a few chains instead of a lot of little booksellers. They both prefer application books with a broad general interest so that they can load the shelves in every store with the same assortment—they want to market them like the mass distribution paper backs at the supermarket. Some of the chains will not even special order a book for you unless it is on their approved list (from one of their favorite vendors). This leaves no outlet for the small publisher with the specialized technical books we need. Someone should develop a centralized distribution center where we can order advanced books. The C User's Group (P.O. Box 97, McPherson, KS 67460) carries a selection of about 100 C and UNIX related titles which are available by mail order. Suport them by ordering any C books from them! We need another place to order other books, anyone out there interested?

S-100 is Not Dead

You don't hear much about those massive S-100 dinosaurs with 8" drives, the ones with the 'boat anchor' power supplies, but there are still a lot of them in heavy use. Most of them originally used eight bit 8080 or Z80 CPUs, but many of them have been upgraded to 16 or 32 bit CPUs.

We still have two Morrow Decision I S-100 systems, and I'm holding on to them for when I can get the time to work on numeric machining and motion control. The large cards and easy-to-work-with bus make an ideal combination for learning and experimenting. I can build 'smart cards' with their own CPU to take some of the burden off the main CPU, or even use several single board computers in a master slave configuration. If I need a large memory space and maximum speed I'll switch to something like AMPRO's new Little Board/286® or Hawthorne's 68000 Tiny Giant® , but it's much easier to learn with the simpler eight bit system—and it's easy to add lots of I/O. If you need something like six bidirectional parallel ports, and ten 20ma serial ports, plus an IEEE-488 instrumentation port, it can be done.

The S-100 hardware hackers bible *Interfacing to S-100/IEEE 696 Microcomputers*, by Garetz and Libes has been out of print, but it is now available again ($24.95 + $2.25 shipping from M&T Books 501 Galveston Dr., Redwood City, CA 94063 (800) 533-4372). If you have any interest in S-100, and don't have this book, get a copy now. The fact that the book was reprinted is another example that S-100 is not dead. Now if we could only get some more articles...

Z-Support Moves

Joe Wright, who you have read about in Sage's column, has taken over the distribution of the Z software from Echelon. I still don't have all the details, but contact Joe at Alpha Systems Corp., 711 Chatsworth Place, San Jose, CA 95128 (408) 295-5594 for any Z type software.

CP/M 3 Bonanza

If you have an original distribution disk for CP/M 3, check the directory with DU or a similar disk utility—whatever you do, don't write to the disk. An anonymous caller told me that he bought an OEM disk marked "update" for $2.00 at a surplus sale. When he snooped at the directory, he found that all the version 3 ASM source code was there as deleted files! He just un-deleted them, and is now the happy possessor of the source code. Either some disgruntled programmer decided to give us a gift, or else they just didn't realize that the undeleted files were there. They must have used a track by track disk duplication instead of a file copy program to dupe the distribution disks. Check whatever you have, and let me know what you find—even if you too want to remain anonymous.

Special Bar Code Issue Coming

Bar codes, those funny lines that the supermarket scanners use, are becoming a very important computer related tool. They have already found many uses in industry, but we have not even begun to realize their potential.

Most of the needs I see are in the area of information and data programming, but there are also many non-business applications. One example is adapting a hand held laser scanner with a menu sheet for computer interfacing by the handicapped—or by executives with keyboard phobia.

There is a definite need for system designers and programmers to implement bar code interfaces. Even more, there is a need for the people who can visualize the new applications for bar codes.

We are preparing a special issue on bar codes, and will also carry additional material in future issues. We invite your participation in the form of articles, ideas, comments, software, etc. We are especially interested in information on unusual applications.

The Shrinking Computer

AMPRO has just announced an AT compatible single board computer which is about the size of a half-height 5¼" disk drive. It includes 1M bytes of DRAM, 3

(Continued on page 5)

C CODE FOR THE PC

source code, of course

	Bluestreak Plus Communications (two ports, programmer's interface, terminal emulation)	$400
	PforC or PforCe++ (COM, database, windows, file, user interface, DOS & CRT)	$345
	CQL Query System (SQL retrievals plus windows)	$325
	GraphiC 4.1 (high-resolution, DISSPLA-style scientific plots in color & hardcopy)	$325
	Barcode Generator (specify Code 39 (alphanumeric), Interleaved 2 of 5 (numeric), or UPC)	$300
NEW!	Vmem/C (virtual memory manager; least-recently used pager; dynamic expansion of swap file)	$250
	PC Curses (Aspen, Software, System V compatible, extensive documentation)	$250
	Greenleaf Data Windows (windows, menus, data entry, interactive form design)	$250
	Vitamin C (MacWindows)	$200
	TurboTEX (TRIP certified; HP, PS, dot drivers; CM fonts; LaTEX)	$170
	Essential resident C (TSRify C programs, DOS shared libraries)	$165
	Essential C Utility Library (400 useful C functions)	$160
	Essential Communications Library (C functions for RS-232-based communication systems)	$160
	Greenleaf Communications Library (interrupt mode, modem control, XON-XOFF)	$150
	Greenleaf Functions (296 useful C functions, all DOS services)	$150
	OS/88 (U**x-like operating system, many tools, cross-development from MS-DOS)	$150
	ME Version 2.0 (programmer's editor with C-like macro language by Magma Software; Version 1.31 still $75)	$140
	Turbo G Graphics Library (all popular adapters, hidden line removal)	$135
	PC Curses Package (full Berkeley 4.3, menu and data entry examples)	$120
	CBTree (B+tree ISAM driver, multiple variable-length keys)	$115
	Minix Operating System (U**x-like operating system, includes manual)	$105
	PC/IP (CMU/MIT TCP/IP implementation for PCs)	$100
	B-Tree Library & ISAM Driver (file system utilities by Softfocus)	$100
	The Profiler (program execution profile tool)	$100
	Entelekon C Function Library (screen, graphics, keyboard, string, printer, etc.)	$100
	Entelekon Power Windows (menus, overlays, messages, alarms, file handling, etc.)	$100
NEW!	TurboGeometry (library of routines for computational geometry)	$90
NEW!	QC88 C compiler (ASM output, small model, no longs, floats or bit fields, 80+ function library)	$90
	Wendin Operating System Construction Kit or PCNX, PCVMS O/S Shells	$80
	C Windows Toolkit (pop-up, pull-down, spreadsheet, CGA/EGA/Hercules)	$80
	Professional C Windows (windows and keyboard functions)	$80
	JATE Async Terminal Emulator (includes file transfer and menu subsystem)	$80
	MultiDOS Plus (DOS-based multitasking, intertask messaging, semaphores)	$80
	WKS Library (C program interface to Lotus 1-2-3 program & files)	$80
	Professional C Windows (lean & mean window and keyboard handler)	$70
NEW!	lp (flexible printer driver; most popular printers supported)	$65
	Quincy (interactive C interpreter)	$60
	EZ_ASM (assembly language macros bridging C and MASM)	$60
	PTree (parse tree management)	$60
	HELP! (pop-up help system builder)	$50
	Multi-User BBS (chat, mail, menus, sysop displays; uses Galacticomm modem card)	$50
	Make (macros, all languages, built-in rules)	$50
	Vector-to-Raster Conversion (stroke letters & Tektronix 4010 codes to bitmaps)	$50
	Coder's Prolog (inference engine for use with C programs)	$45
	C-Notes (pop-up help for C programmers ... add your own notes)	$40
	Biggerstaff's System Tools (multi-tasking window manager kit)	$40
	PC-XINU (Comer's XINU operating system for PC)	$35
	CLIPS (rule-based expert system generator, Version 4.1)	$35
	Tiny Curses (Berkeley curses package)	$35
	TELE Kernel or TELE Windows (Ken Berry's multi-tasking kernel & window package)	$30
	Clisp (Lisp interpreter with extensive internals documentation)	$30
	Translate Rules to C (YACC-like function generator for rule-based systems)	$30
	6-Pack of Editors (six public domain editors for use, study & hacking)	$30
	Crunch Pack (a dozen file compression & expansion programs)	$30
	ICON (string and list processing language, Version 7)	$25
NEW!	FLEX (fast lexical analyzer generator; new, improved LEX)	$25
	LEX (lexical analyzer generator; an oldie but a goodie)	$25
	Bison & PREP (YACC workalike parser generator & attribute grammar preprocessor)	$25
	AutoTrace (program tracer and memory trasher catcher)	$25
	C Compiler Torture Test (checks a C compiler against K & R)	$20
	Benchmark Package (C compiler, PC hardware, and Unix system)	$20
	TN3270 (remote login to IBM VM/CMS as a 3270 terminal on a 3274 controller)	$20
	A68 (68000 cross-assembler)	$20
	List-Pac (C functions for lists, stacks, and queues)	$20
	XLT Macro Processor (general purpose text translator)	$20
NEW!	C/reativity (Eliza-based notetaker)	$15
	Data	
	WordCruncher (text retrieval & document analysis program)	$275
	DNA Sequences (GenBank 52.0 including fast similarity search program)	$150
	Protein Sequences (5,415 sequences, 1,302,966 residuals, with similarity search program)	$60
	Webster's Second Dictionary (234,932 words)	$60
	U. S. Cities (names & longitude/latitude of 32,000 U.S. cities and 6,000 state boundary points)	$35
	The World Digitized (100,000 longitude/latitude of world country boundaries)	$30
	KST Fonts (13,200 characters in 139 mixed fonts: specify TEX or bitmap format)	$30
	USNO Floppy Almanac (high-precision moon, sun, planet & star positions)	$20
	NBS Hershey Fonts (1,377 stroke characters in 14 fonts)	$15
	U. S. Map (15,701 points of state boundaries)	$15

The Austin Code Works — *Voice: (512) 258-0785*

11100 Leafwood Lane — *acw!info@uunet.uu.net* — *BBS: (512) 258-8831*

Austin, Texas 78750-3409 USA — *FidoNet: 1:382/12*

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

Reader's Feedback

Wants Hardware Construction Articles

I own a XEROX 820-II which serves me very well except I would like more memory and graphics capability. I own two printers, a Diablo 620 daisy wheel and a Centronics 702 dot matrix.

I would like to see a construction project in your magazine once in a while. Projects worthy of consideration are stand alone prom burner, HD64180 computer, and a 68000 computer which would run IBM software (which I would like) all with wire lists and large schematics, both of which you could sell.

If you venture into the construction area just remember to do only one project at a time and finish it or you will go the way Computer Smyth went.

Another area you might venture into is to start an assembly language course for the Z80, HD64180 and 68000.

H.D.

Jay Sage Fan

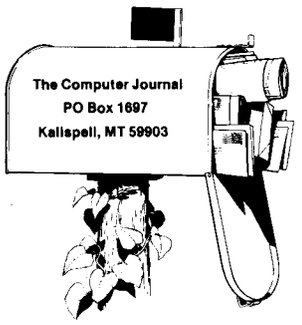
I subscribe to TCJ simply because of Jay Sage's column and for the other ZCPR support you provide. One recent subscriber was moved to do the same after seeing a couple issues of TCJ at my house, and yet another has his "check in the mail."

But I am also the owner of a S-100 system and therefore have a great deal of interest in many of the hardware articles you provide. Like at least one other letter writer, I am intrigued at the idea of replacing my 8" dives with "work-alike" 3" or 5" (as have been reported in the various articles on floppy disk formatting, etc.), but am incapable of figuring out how to do that for myself. Articles in *S-100 Journal* and *Microsystems/Journal* on building/modifying the DRC LS-100 RAM disk, for example, are just the thing for people like me.

Manage to read through nearly every article in your magazine, even when its subject matter or technical level exceeds my own. That's how I've learned as much as I have about computers over the years and I intend to keep up the effort.

Thanks for keeping it all going.

M.B.



Seagate 225 Feedback

Regarding your general inquiry in the last issue concerning overheating of Seagate 225s, please be advised that I have run with one in my Katpro '84 series CP/M machine (with TurboROM, SWP Board, etc.) nearly everyday for the past 2+ years without even a hint of a problem. I am aware of others also with 225s of about this age and they also say that they experience no problems.

We have one common denominator, however, that might be the reason why ours have worked flawlessly and yours reacts adversely to its heat; our fans. I don't notice your saying that your heat-plagued system had a muffin fan cooling the 225. We all do. As I recall, these small fans cost under $10 when originally purchased new. I've seen them used for less.

This might not be why five of us never have had a problem with our hot 225s. But the guy without the fan can't say that.

On another matter, I found the last issue quite exciting. I'm a CP/Mer who is not as advanced as I should be (either I'm a slow learner or it just takes longer than 4+ years to understand everything I should know in order to more fully appreciate all the articles in TCJ). But suddenly I felt as if you either had found my level or I had somehow made it to yours (the latter is doubtfull).

It could be that the writing was a bit more understanding of those of us from "technically-disadvantaged" backgrounds who still need some mental handholding. I notice that Jay Sage even took notice of this in his last (very interesting) article. And it also might have been what was covered. Welcome to Bridger Mitchell (one of CP/M's all-time all stars) and to Phil Hess's investigation of DOS WS4 for the SWP board and vanilla DOS world. And of course, Jay's irrepressible energy always seems to have something to excite CP/M advocates.

I may not know as much as I need to in order to fully appreciate everything in TCJ. But I sure know what I like. And I liked #32

H.V.

Editor's Note: The 225 was running with very little cooling when I had the problems. It's still running fine since I moved it to the bigger box with a good fan. We are at 3,000 feet, does the lower air density have an effect under marginal conditions?

Another S-100 User

Like your magazine very, very, much, glad you're still with us.

Glad to see 1) CP/M columns. Especially glad to see CP/M, Z80, 64180 stuff; it's getting extremely scarce these days. We've a ton of CP/M hard and soft ware. I'm still trying o get a RAM drive up and running on my S-100 system. 2) 680X0 hardware and assembly language articles. 3) Generic hardware articles. 4) Nitty gritty, down and dirty, hardware articles.

Don't care about MSDOS, 80XXX, clone, etc. world. There's too much written about them as it is.

Would like to see (and I know you can't print something someone doesn't write for you) something on the Atari 680X0 machines. The 520 and 1040 STs are such a good buy, they simply have to be about the best test bed around for 680X0 projects, taking nothing away from the Hawthorne Tech. product.

C.H.

S-100 Again

I have three systems. (1) A CP/M S-100 system with 8" drives which I built myself. I use it for frequency counter, modem, and EPROM burner. (2) A CP/M MORROW Design with 5¼" drives. I recently assembled this from surplus boards and use it for environmental logging. I built a 12 channel A/D and expanded I/O with 32 bits input and 16 bits output. (3) A IBM clone with 20 meg drive

used for AUTOCAD (schematics, etc.).

My main interest is in real time control applications and the hardware/software compromises necessary.

I really enjoy TCJ and realize that we seem to be a dying breed (those who are interested in what is behind the keyboard).

D.C.

Still More S-100

I would like to read more about ZCPR and K-OS ONE, and any S-100 projects.

I have a Morrow Decision I Z80 S-100, a Cromemco System III Z80 S-100, and an IBM-AT clone with MSDOS and EGA. I also have a Vector Grafic chassis S-100 with Z80 and 68K CPU, an Itegrand S-100 with 68K CPU, a Momentum "Hawk-32" 68K UNIX multiuser, and six Momentum "Eagle" 68K single user work stations—I want to put K-OS ONE on all these 68K systems.

J.S.

Editor's Note: There are still a lot of very active S-100 users. We would like to see more S-100 articles.

From Down Under

My systems are a Heathkit/Zenith 150 (PC clone), and a Pinnacle 68000 (much like the Sage). I'll probably buy the K-OS ONE and try to get it running like Bill Kibler has done on the Sage. I'm interested in general OS stuff for the 68000 and 8086/8088.

Your issue #31 arrived here in Australia in the middle of April— that's OK. Don't do like some other magazines and only send them by air—makes the other magazines unaffordable.

R.A.

Editor

(Continued from page 2)

counter-timers, 7 DMA channels, 16 level interrupts, optional 80287, real time clock, parallel printer port (with bi-directional data lines), two RS-232C serial ports, 2 drive floppy controller, optional onboard EGA/CGA/MDA/Hercules video controller, and SCSI interface—all this on a 5 × 8 board!

This board is intended for embedded applications, only needs about 8 watts at +5 volts, standard PC and AT bus plug in cards can be connected, and SSD software is available to EPROM MS-DOS based applications for diskless DOS operation.

All this power on a small low power board which generates little heat and which can run with out a disk drive makes me want to design a system with three or four of these in a single box—I don't want multi-users on a single CPU, I want one user on multi-CPUs. ■

MOVING?

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.

Call For Papers

TCJ is establishing a forum on the following areas, and we welcome your submissions and proposals. Candidates for membership in the peer review and advisory groups, including group coordinators, will also be considered.

• **Education in the Next Decade** — Our contacts with both the educators who are preparing the curriculums and the people in industry who need to employ workers with the necessary skills, indicate that the requirements are changing. Industry sources say that current graduates do not have the knowledge to fill available real world positions, and the educators say that they do not have the course material and specific requirements needed to implement the courses. TCJ invites papers from both Academia and Industry to discuss the problem and propose solutions.

• **Language Development** — There is a great need for language development in the areas of command parsers, user interfacing, custom languages, ROM based embedded controller systems, etc. We need papers covering both the theoretical and practical aspects from the viewpoints of both the developers and the users.

• **Database Development** — The commercial programs are very powerful, and there are good texts which explain the commands and functions. What is missing is tutorials on the concepts of the practical aspects of designing and developing a database — the nitty-gritty details on implementing a database rather than an explanation of the tools.

There is also a need for papers on using high level languages to replace or supplement DBMS programs where it is easier or more efficient to perform some of the operations outside of the DBMS.

Other suggested topics are welcome. Query regarding book or monograph manuscripts.

The Computer Journal
190 Sullivan
Columbia Falls, MT 59912
(406) 257-9119

Data File Conversion

Writing a Filter to Convert Foreign Formats

by Art Carlson

Data handling programs, such as database management systems, use many different structures for their internal data files. Some of the variations are fixed length fields versus random length fields, field and/or record delimiters versus no delimiters, the delimiters used, and how numeric data is stored. Since there is often the need to transfer data from one program to another, most database programs make some kind of provision for importing or exporting data, but the systems are often incompatible. A lot of our work involves filtering files in order to enable transfer between programs.

The first database program I acquired was the CP/M version of Condor® , which provides five formats for reading and writing ASCII files. The one we usually use is the [B] option which creates a standard MailMerge® file with variable length fields. Each character type field (which might contain commas) is enclosed in quotation marks, the fields are separated by commas, and each record is ended with a carriage return and a line feed.

I am currently starting to use dBase® (actually I'm using Nantucket's Clipper® dBase compiler) which can use the COPY or APPEND command to import or export data using the same MailMerge format.

This makes it easy to create compatible forms and then move data from one system to the other. I can receive data files as either Condor files, dBase files, or MailMerge ASCII files, and then transfer them through the MailMerge format intermediate file. Unfortunately things are not always that simple. There are many other data handling programs with different import/export schemes, and the person sending the data does not always understand what is required—some programs don't even provide for data import/export through the use of disk files. With those programs you have to retype everything!

We recently received a mailing list which was prepared using a Macintosch® database program. It had been output as an ASCII file for use in a form letter, but neither Mailmerge nor Condor could read the file. Whenever I have problems with a text or data file, the first thing that I do is to DUMP a portion of it to the printer in both HEX and ASCII. The debuggers can be used, but I use a simple DUMP program because some of the debuggers (DDT for example) can not handle a file which is larger than available memory.

The dump showed that the file had variable length fields, with the fields terminated with a tab character (^I or 09HEX) and the record terminated with a carriage return (^M or 0DHEX). Empty fields and fields which contained a comma were enclosed in quotation marks. I have converted similar files by using the WordStar® search and replace function, but that would take too long for a file this size. It was an ideal opportunity to write a simple filter program.

I consider these filters as *throwaway programs* because they are written for one time use. Anything which does the job is acceptable, and they are "quick and dirty" with little user interfacing and a minimum of comments. I will probably never use the program again, but I may modify it or use parts of it for another project. I chose to write it in C because I am comfortable with C and it is well suited for the job. It could just as well be written in BASIC or Pascal. It could also be done in assembler, but that would involve too much programming time for one time use.

Everyone has their own style. They talk about top down programming, bottom up programming, and middle out programming. It all boils down to first figuring out what has to be done, and then figuring out how to do it. Sometimes I use flowcharts, and sometimes I use pseudo code. If I hit a wall, I may even use both at the same time. It seems that flowcharts are helpful for visualizing the overall structure of an involved program and planning the method of attack, while pseudo code assists in writing the code. In this example I could see what had to be done and wrote the following pseudo code.

```
Write ''
While not End Of File
  Pass chars until ^I or ^M
  If ^I write '',''
  If ^M write '' ^M ^J ''
```

The C source code shown in Figure 1 was written for the BDS C® Compiler, but could easily ported over to another compiler. There is a problem, in that this program writes an extra quote after the last record, but it was easier and faster to edit out the extra quote than it was to to write the code to keep checking for the last record.

This program does a bare minimum of error checking or user interfacing. It was tempting to dress it up with a nice interface for this article, but I felt that it would be more truthful to show it just the way I used it. The most glaring fault is that it does not check to see if the output file already exists, and will just overwrite it. In this article we'll talk about converting data files and leave error checking and interfacing for another time.

Writing a Filter in C

It is always difficult to decide on the amount of detail to include in the description of a program. I don't want to include all the basics of C programming every time; but then again, I don't want to ignore someone who is just getting started in C. This time I'll cover a lot of the fundamentals. In future articles I'll place the fundamentals in a separate sidebar.

Lines 1 through 10 contain comments and information. In C programming, comments start with /* and end with */. BDS C allows comments to be nested, which makes it easy to comment out blocks of code for testing. The UNIX® standard does not allow comment nesting. Datalight® follows the UNIX standard by not allowing comment nesting, while Lattice C® version 3.00 does allow nesting. Both BDS and Lattice provide a compiler option to disable comment nesting, causing comments to be processed in the same way as UNIX.

```
 1: /*
 2:     PRDB.C      2/2/88
 3:     Written for BDS C V1.6   RAC
 4:
 5:     A program to convert the MAC database ASCII file to comma
 6: delimited for Condor and MailMerge
 7:     The file we received has 13 fields, delimited with 09hex,
 8: with a 0Dhex record delimiter
 9:
10: */
11:
12: #include <stdio.h>
13:
14: FILE *ifd, *ofd;
15: int c;
16:
17: main(argc,argv)
18: char **argv;
19: {
20:     if (argc!=3)
21:         {
22:         printf(''\nUsage; prdb oldfile newfile\n'');
23:         exit();
24:         }
25:     if((ifd = fopen(argv[1],''r'')) == NULL)
26:         {
27:         printf(''\nCan't open file %s\n'',argv[1]);
28:         exit();
29:         }
30:     if((ofd = fopen(argv[2],''w'')) == NULL)
31:         {
32:         printf(''\nCan't open file %s\n'',argv[2]);
33:         exit();
34:         }
35:
36:     fprintf(ofd,''%c'',0x22); /* send '' */
37:     while ((c=fgetc(ifd)) != EOF)
38:     {
39:         if(c==0x09)
40:             fprintf(ofd,''%c%c%c'',0x22,0x2c,0x22);
41:         else if(c==0x0d)
42:             fprintf(ofd,''%c%c%c%c'',0x22,0x0d,0x0a,0x22);
43:         else if(c==0x22)
44:             continue;
45:         else
46:             fprintf(ofd,''%c'',c);
47:
48:     }
49:     printf(''\nOperation Completed -- Returning to CP/M\n'');
50:     fclose(ifd);
51:     fclose(ofd);
52: }
53:
```

Line 12 tells the compiler to include the standard I/O header file which defines certain system dependent data structures, and macros such as the EOF (End Of File marker). For BDS C, this must be the first statement after the initial comments.

Lines 14 thru 34 constitute a block of code for programs which read one file and write to a second file. I store this block on disk, read it in when writing a program, then make minor modifications for the individual program. I could place it in the library and just call it as a function, but it usually needs some changes. In C, variables must be declared before they are used. Line 14 declares *ifd and *ofd as type FILE, for the input and output files. Line 15 declares the integer c which is used for the individual characters which will be tested. Even though these are characters, in BDS C they are declared as an integer because a char can not have a negative value (in BDS C) and the program would not find the EOF marker of −1 if c was declared as char. I spent hours trying to find out why a program would not stop at the end of a file, and the problem was that I had declared c as a char!

Every C program must have the function main. This is the function which is run, and which calls any other functions. Main's arguments argc and argv are the only arguments that main can have, and they are obtained from the command line.

argc contains the number of individual command line strings, which are delimited with spaces on the command line. argc will always be at least 1, because the name of the executing program counts as the first one. Lines 20 thru 24 check to see if both the input and output files are given on the command line, and returns to the system with an error message if there are not exactly three arguments. A good user interface would ask for the missing information instead of dumping back to the system.

Lines 25 thru 29 attempt to open the input file (argv[1]), and return to the system with an error message if the file can not be opened as entered in the command line. The option "r" in line 25 specifies that the file will be opened as a text file in the read mode. fopen initializes the buffer, and returns a file pointer (ifd) to be used in all subsequent references to operations on the associated file.

Lines 30 thru 34 attempt to open the output file (argv[2]) in the text write mode. If it can not open the file, possibly because of a full or write protected disk, it returns to the system with an error message. If the file already exists, it will be overwritten, destroying the contents of the original file! A safer approach would be to first attempt to open the file in the read mode. If it can be opened to read, it already exists, and the program can then ask if it is OK to destroy it. If it does not exist, then it can be opened to write to.

After line 36 writes an initial quote mark to the output file, the real action starts in line 37. The while loop gets one character at a time until the end of the file (EOF) is reached. Each character is tested in lines 39 thru 43, with the appropriate action taken. In C, hes numbers are preceeded with '0x', and the 0x09 in line 39 is the ASCII code for a control I. If the character fails all the tests, lines 45 and 46 send it to the output file unaltered.

Finally, line 49 sends a completion message to the console, and the files are closed in lines 50 and 51.

I'm sure that there is a lot that could be done to improve the program, but I only had to write 10 lines of code. The rest was a block read with WordStar. It only took a few minutes to code and compile the program—it took about 5 minutes to run (it was finished when I got back with my cup of coffee)—and I was back to working with the database. Data file manipulations are very interesting, and they can get very involved. Let me know if you'd like to see more advanced articles on the subject. Perhaps you'd like to respond with an article describing what you do, or how you do it with assembler, BASIC, Pascal or something else. ■

DosDisk™ -- An MS-DOS Disk Emulator for CP/M

DosDisk, for CP/M 2.2 and CP/M Plus Z80 computers, allows CP/M programs to use files stored on an MS-DOS (PC-DOS) floppy disk *directly* -- without intervening translation or copying. You can *log into* the pc disk, *including subdirectories*. Regular CP/M programs can read, write, rename, create, delete, and change the attributes of MS-DOS files. The disk, with any modified files, can immediately be used on a pc.

Preconfigured Versions are available for:

all Kaypros with a TurboRom
all Kaypros with a KayPLUS rom and QP/M
Xerox 820-I with a Plus 2 rom and QP/M
Ampro Little Board
SB180 and SB180FX with XBIOS
Morrow MD3
Morrow MD11
Oneac ON!
Commodore C128 with CP/M 3 and 1571 drive

The resident system extension (RSX) version uses about 4.75K of main memory (plus 2K for the command processor). For the SB180 and SB180FX, a banked system extension (BSX) version is also available; it needs about 5K of the XBIOS system memory and *uses no main memory*.

A **Kit Version** requires *advanced assembly-language experience* in Z80 programming and *technical knowledge of your computer's BIOS*. You will need to write a special **DosDisk** overlay.

The BIOS must be able to be configured to use the physical parameters of an MS-DOS disk and to use the logical disk parameter header (dph) and disk parameter block (dpb) values supplied by **DosDisk**. The driver code itself (the code that programs the disk controller, reads and writes sectors, etc.) must reside *in the BIOS*.

On DateStamper, QP/M and CP/M 3 systems **DosDisk** automatically stamps MS-DOS files with the current date and time when they are created or modified.

DosDisk supports the most popular MS-DOS format: double-sided double-density 9-sector 40 track disks. It cannot format disks or run MS-DOS programs.

Z3PLUS™ --The Z-System for CP/M Plus

The *state-of-the-art* ZCPR version 3.4 system for CP/M Plus (CP/M 3) Z-80 computers *installs automatically* and retains CP/M Plus advantages -- fast disk operations, redirection of screen, keyboard and printer; automatic execution of submit files.

Z3PLUS is fully configurable and requires no assembly. It is shipped with key Z tools and will run most Z-System CP/M 2.2 utilities without modification.

==========

DosDisk and **Z3PLUS** are available directly from the author of DateStamper and BackGrounder ii:

Plu*Perfect Systems
410 23rd St.
Santa Monica, CA 90402

Name: _____

Address: _____

_____

Computer: _____

Operating system: _____

Disk format: _____

Check Product:

[ ] **DosDisk** preconfigured version ..... $ 30.00
[ ] **DosDisk** kit version ..... $ 45.00
[ ] **DosDisk** manual only ..... $ 5.00
[ ] **DosDisk** BSX and RSX,
for SB180/SB180FX with XBIOS ..... $ 35.00
[ ] **Z3PLUS** ..... $ 69.95
(in California, 6.5% sales tax) ..... _____

shipping/handling ..... $ 3.00

total enclosed ..... $_____

Advanced CP/M
Z3PLUS & Relocation

by Bridger Mitchell

Z3PLUS is Here!

The ZCPR3 System is arguably the most important advance in CP/M® operating system capabilities that has appeared to date. Each year, its community of users with CP/M 2.2 compatible comupters has continued to press forward with vigorous new developments. Yet, for too long, users of CP/M Plus® (CP/M 3) computers have been unable to benefit from ZCPR3, apart from the limited capabilities provided by CCP105 and a named-directory RSX.

I have now completed the "port" of ZCPR version 3.4 and the Z-System® utilities to Z-80® compatible computers running CP/M Plus. The system is called *Z3PLUS*. It is a state-of-the-art system that installs automatically and can be very easily configured for different system components and buffer sizes; moreover, these settings can be changed on-the-fly, even within one multiple-command line.

Z3PLUS is implemented as a special type of RSX. This means that the major advances introduced in CP/M Plus, including directory hashing, track buffering, console and list redirection, automatic submit file execution, and RSX support, are fully available when the Z3PLUS system is running. And because the ZCPR3 buffers are located in the TPA memory bank, virtually all Z-System utilities will run without modification on Z3PLUS systems. (Limited upgrading is required for programs that calculate the free space on a disk or use BIOS file services.) In fact, with Z3PLUS, users have the best of both worlds—CP/M Plus and ZCPR3!

By the time this column appears in print Z3PLUS will be available from Plu*Perfect Systems, Sage Microsystems East, and Echelon. If you have a Morrow MD-5, Commodore 128, Osborne Executive, Amstrad, S100, or other CP/M Plus Z-80 computer, here at last is your chance to leap effortlessly to a more powerful and flexible operating environment.

ZCPR3? Absolutely Not!

The ZCPR3 system has been a curious mix—innovative concepts that greatly extend the performance of the CP/M command-processing system combined with a clumsy and antiquated method of generating the necessary system files. The original ZCPR3 developers were obviously acquainted with macro assemblers and regularly used routines from relocatable library files, the ZCPR3 system itself has required both the first-time and veteran user alike to specify tedious and arcane sets of assembly-language equates and then re-assemble an entire batch of files almost any time ZCPR3 was installed on a new, or even modified system. Many potential users gave up in frustration. Others who eventually succeeded, cringed at the thought of modifying their systems once they had them running.

Echelon eased the pain for many by supporting the development of "bootable disk" Z-Systems incorporating both ZCPR3 and a replacement BDOS, pre-assembled and installed on the system tracks for a specific computer. And Joe Wright came up with Z-COM, a method of automatically installing a ZCPR3 system of one specific size in a running CP/M 2.2 system. Yet these important advances remained bound to absolute addressing.

During this period, at Plu*Perfect Systems Derek McKay and I had designed the TurboRom for Kaypros with support for full relocatation of the BIOS, BDOS, command processor, and DateStamper. And in writing BackGrounder ii I had extended this approach to a full task-switching command processor with multiple overlays.

From these experiences I understood that a fully-relocatable ZCPR3 system could readily be achieved. In the TurboRom utilities we had provided the user with software that would create any sized TPA system (in 0.25K increments) to accomodate varying BIOS requirements. Even more than its convenience, this flexibility is important because it allows the user to add disk drivers, more or different-sized disks, and other system software as the need arises, without ever reassembling the system files. It was clear that similar techniques could be applied to the ZCPR3 command processor, Z-System segments, and a replacement BDOS.

Thus, when I took up the task of getting ZCPR3 to run on CP/M Plus computers, my design goal from the outset was a fully relocatable Z-System, one that would run the same files on any CP/M Plus system with no assembly! The specific method that I developed uses named-common address spaces. This technique is not widely known to Z-System programmers, but it was enthusiastically adopted by Jay Sage and Joe Wright for ZCPR34 and NZCOM, and is now the foundation for a truly portable set of Z-System files.

The development of a fully-relocatable Z-System is a milestone in the evolution of CP/M operating systems. These new standards mean that you can now take the same Z-System segment file (for example, a resident command package) and run it on any ZCPR34 compatible system, regardless of the TPA size or configuration of that system.

I've organized the remainder of this issue's Advanced CP/M column around the concepts of relocatable code and named-common address bases, and how they are used in both the new Z-System and for other operating-system needs. I'll discuss several closely related topics, first reviewing how an assembler uses named-common address bases to make use of different segments of code and data that have known structures but unknown

---

Bridger Mitchell is a co-founder of Plu*Perfect Systems. He's the author of the widely used DateStamper (an automatic, portable file time stamping system for CP/M 2.2); Backgrounder (for Kaypros); BackGrounder ii, a windowing task-switching system for Z80 CP/M 2.2 systems; JetFind, a high-speed string-search utility; DosDisk, an MS-DOS disk emulator that lets CP/M systems use pc disks without file copying; and most recently Z3PLUS, the ZCPR version 3.4 system for CP/M Plus computers.

Bridger can be reached at Plu*Perfect Systems, 410 23rd St., Santa Monica CA 90402, and via Z-Node #2, (213)-670-9465.

locations. This leads to the new ZRL file type for Z-Systems and to a new multi-purpose linking loader called JetLDR. Finally, we'll look at one solution to this issue's puzzle: how can you write a routine that will run anywhere in memory?

I'd expected to discuss Resident System eXtensions in this space, but the completion of the named-common standard for Z3PLUS, ZCPR34 and NZCOM took priority. The next column should get to that topic, and will make use of the new multi-purpose JetLDR for convenient installation and initialization of a CP/M 2.2 RSX.

Assemblers and Relocation

To discuss relocation, we first need to understand some basic operations of a Z-80 assembler.

An assembler generates code (instructions for the central processor) by converting symbols in the input stream into op-codes, constants and addresses. It processes a number of other useful directives, called pseudo-ops ("pseudo" because they are not actual CPU operation codes), that make the assembly programmer's life easer, but don't need to be covered here.

Translating "RET" into a "C9" byte or "defw 80" into the byte pair "00 50" is straightforward. The real action is in manipulating symbolic references and calcaulating addresses.

An assembler has two addressing modes—absolute and relative. In absolute mode all addresses are "resolvable" (can be determined) from the information in the source file. (The actual production of the addresses may require several passes throught the source, depending on the technology used in the assembler). The output of an absolute mode assembly is an absolute file, ready-to-run at a specified address, normally 100h.

In relative mode, some of the addresses are not resolvable without external information. The assembler does what it can with the source file information, generating all op codes, constants and absolute addresses, and does some processing of the relocatable addresses. The output of the assembly is a relocatable (REL) file which contains all of the code, but with some addresses not yet calculated and with "tags" for the bytes that require further processing.

The format used by the assembler for the relocatable file varies by assembler. Microsoft uses a bit-encoded format; SLR and TDL use two different byte-oriented formats. Each format encodes addressing tags and symbols differently. Unfortunately, the formats are incompatible; fortunately, we don't have to concern ourselves with these details in this column!

Linkers

The standard tool for processing a REL file is a linkage-editor (linker). It takes as input one or more REL files plus information about the values of external symbols and produces an output file. The most common use of a linker is to merge several REL files (e.g. a main file, and a library of standard subroutines called from the main code), resolve the references between them into absolute addresses, and output an absolute file ready-to-run at a specified address, usually 100h.

Some linkers can merge and resolve addresses with the available input and produce relocatable output for a subsequent linking step. For example, the SLR+® linker will generate output in PRL and SPR relocatable formats (which we'll discuss later). The TDL linker has the very useful ability to produce an *output* file in REL format file; using this feature one can merge a number of REL files into a single module that can still be relocated to any ready-to-run address. (Unfortunately, the TDL relocatable format is totally different from both SLR and Microsoft! Output to a REL file is a feature I'd very much like Steve Russell and Al Hawley to incorporate into their linkers.)

You might think that a linkage editor is necessary in order to use a program in relocatable format, but sometimes it isn't. If all of the code needed by the program is contained in the one REL file, the process of resolving addresses is simpler than that required for merging several files. This opens up a very interesting opportunity—postpone resolving the final addresses of a program until the moment that it is loaded into memory. By using a *linking loader* we can keep code in relocatable format and convert it to absolute addresses to fit the needs of the system on which it will run. JetLDR is a new utility I have written that performs exactly this function, for a wide variety of applications. But before describing it we need to understand named-commons.

Relocation Bases

An assembler accepts directives to place the source code or data into any of several different *address spaces*. In absolute mode, only one address space is used, and its relocation base address is 0000h. The assembler maintains a program counter,, and the ORG pseudo-op instructs the assembler to set the program counter to a specified address. Then, as the assembler converts code from the input into machine instructions and data, it increments the program counter by the number of bytes used.

In relocatable mode, several address spaces are available, and each has its own program counter. The pseudo-ops CSEG and DSEG instruct the assembler to assemble the code that follows into the code segment's address space (CSEG) or data segment's address space (DSEG).

These two directives are ordinarily used as their names suggest, to separate code and data in a program. You can intermix the two, for example, by writing:

```
        CSEG
        ld   de,msg

        DSEG
msg:    db   'Hello!$'

        CSEG
        ld   c,9
        call bdos
```

This works, because the assembler will continue to increment the program counters for the code and data segments whenever the corresponding directive appears in the input stream. Actually, their names notwithstanding, there's no reason you can't put code in the DSEG, or data in the CSEG.

Unless you've worked with FORTRAN you may be unaware that full-featured assemblers also have a number of *named-common address spaces*. Named-common is often used to pass data between FORTRAN subroutines by placing the data in commonly-accessable memory and labeling the memory block with a commonly-shared name.

Named-commons work very much like CSEG and DSEG. You instruct the assembler to use a named-common address space with the COMMON directive. For example,

```
COMMON /MSGS/

msg1: db   'first message$'
msg2: db   'another one$'
```

The assembler will tag the addresses for msg1 and msg2 to indicate that the associated bytes are to be found in the MSGS address space. These tags, along with the data bytes themselves, will go into the REL output file for later processing by the linker or linking loader.

The assembler does treat the program counters for named-commons a little differently. If the same named common is declared again, the program counter does not continue where it left off, but starts again at 0. The reason for this difference is to allow several REL modules to refer to the same addresses in named-common address spaces. Suppose, for example, we had a second named common in our source file:

```
COMMON /PDATA/

pcols: db   80
prows: db   66
...
```

one that contains printer parameters. Then, in another rel file, routines could refer to these parameters by:

```
COMMON /PDATA/

pcols equ $
prows equ $+1
```

Using Named Commons for System Segments

An operating system is made up of several segments of code and data structures, such as the BIOS, the BDOS, the command processor, and the terminal capabilities buffer. For the most part each code segment is self-contained except for references it makes to routines or data in other segments *by position*. A routine that calls a BIOS function refers to a jump vector at a known offset from the start of the BIOS; a ZCPR34 command processor routine that is checking the error code of the previously-executed command refers to an offset from the start of the message buffer; and so forth.

In each case, the inter-segment references are to *known offsets from unknown base addresses*. Given this type of structure, it is convenient to assemble each segment separately and use named-commons to provide the intersegment connections.

The New ZRL File Type

For the Z-System—Z3PLUS, ZCPR34 and NZCOM—we have established a standard set of named commons, shown in the Z3COMMON.LIB file in Listing 1. Each name begins and ends with an underscore character (__) to emphasize that the symbol describes a segment address base rather than an address itself. Relocatable files that are assembled using these named-common bases are termed "ZRL" (Z-system ReLocatable), to distinguish them from ordinary REL files.

Suppose,now, that the external environment's message buffer is located at 0FC00h. In the traditional ZCPR3 command processor, there would be an equate

```
z3msg equ 0FC00h
```

in an "include" or "maclib" file, and references to the message buffer in the code would be of the form

```
ld   hl,z3msg
```

When assembled, this address (0FC00h) is hard-wired into the resulting command processor file, and that command processor can be used only in a system in which all hard-wired addresses match those in the system. Even if the assembler outputs a REL file, the z3msg address is fixed, and the command processor file will only work on systems that have the message buffer at that exact address.

Using the named-common approach we omit all hard-wired references to addresses outside the module's own segment (other that absolute addresses on page 0, such as "call 5"). Instead, we declare "z3msg" to be in the named-common address space "__MSG__" and equate "z3msg" to its base address

```
COMMON /_MSG_/
z3msg  equ $
```

We place all of the command processor code in the code segment

Figure 1
The Procedure for Creating and Running Your Program ANYWHERE.

```
To create the program file:

   (1) assemble your code into REL file
               Z80ASM file/n/r,file,,
   (2) link it into a PRL module
               SLRNK+ file/k,file/j,file/e

With a debugger:

   (3) patch the length of code into 104h
        ZSID ANYWHERE.COM
        iFILE.PRL
        r100
        display the length, which is the word at 201
        DW 201,203
        patch the length into ANYWHERE at 104
        SW 104
           length

   (4) append the code+bitmap at 147h
        m300,xxx,147
   (5) save into a COM file
        g0
        save nn myprog.com

To run the file:

    (6) A>myprog<cr>              (execute at 100)

or

    (7) A>get nnnn myprog.com
        A>jump nnnn               (execute at nnnn)
```

```
   CSEG
; ...
   ld   hl,z3msg  ; sample reference to z3msg
; ...
```

The result: the address in the relocatable ZRL file for "z3msg" is not 0FC00h, but 0000 plus a tag to the named-common base __MSG__.

JetLDR—A Linking Loader for ZRL Files

The ZRL file requires one more step of processing to become executable—linking to the final run-time addresses for the system where it is to run. This process is conventionally performed by a linking editor (such as SLRNK or L80). The user supplies final addresses for each segment (CSEG, DSEG, and named commons), and the linker produces an absolute image file (COM or perhaps CIM).

This is where JetLDR comes in. It performs the linking step by obtaining the final addresses *from the current external environment* of the computer system in which it is running. Thus, a Z-System ZRL file is tailored to the exact system that is running at the moment.

JetLDR provides rapid loading and relocation for all types of ZCPR3 system packages. Packages (segments) may be loaded as separate files or as members of a library. Thus, JetLDR should fully replace LDR and LLDR. The complete command-line syntax is:

```
A> JetLDR //
   gives usage message
```

Listing 1

```
                                Z3COMMON.LIB
        Standard named-common relocation bases for the Z-System.

;
        COM     /_BIOS_/
bios    equ     $               ; BIOS location (xx00h)
;
        COM     /_ENV_/
z3env   equ     $               ; environment descriptor
;
        COM     /_SSTK_/
shstk   equ     $               ; shell stack
;
        COM     /_MSG_/
z3msg   equ     $               ; message buffer
;
        COM     /_FCB_/
extfcb  equ     $               ; external fcb
;
        COM     /_MCL_/
z3cl    equ     $               ; multiple command line buffer
;
        COM     /_XSTK_/
extstk  equ     $               ; external stack
;
;
;
; The following named commons are for use of Z3PLUS only.
; The declarations are listed here for reference;
; they should appear in the command-processor source file,
; not this ''include'' file.
;
;       COM     /_CCP_/
;ccp    equ     $          ; command processor
;
;       COM     /_SCB_/
;scb    equ     $          ; cp/m 3 system control block (xx00h)
;
;       COM     /_RSX_/
;rsx    equ     $          ; cp/m 3 RSX containing Z-system
;
;
; The following named common is for use of NZCOM only.
; The declaration is listed here for reference;
; it should appear in the bios source file,
; not this ''include'' file.
;
;       COM     /_CBIO_/
;cbios  equ     $               ; base of original system bios
;
;
        CSEG                    ; ensure code segment
```

```
A> JetLDR [du:]file1.typ, [du:]file2.typ, ...
  loads file1.typ, file2.typ, ...

A> JetLDR [du:]lbrfile[.lbr] file1.typ file2.typ ...
  loads membvrs file1.typ, file2.typ,
    ... of the library file ''lbrfile.lbl''
```

The optional "du:" may be a drive/user spec, or a named directory. If neither is given, JetLDR searches the path for the file.

The file types may be:

```
FCP - flow commands     ENV - environment
IOP - input/output      NDR - named directories
RCP - resident commands Z3T - terminal capabitlities
ZRL -- FCP, RCP, IOP, CCP, CP3, DOS, DO3,
    BIO, or CFG in relocatable format
```

If the file type is ZRL or REL (in SLR or MS-relocatable format) then JetLDR will relocate and load the FCP, RCP, IOP, CCP, DOS, BIO, or CFG package, provided that its corresponding module name (created by the assembler using the NAME directive) is:

```
FCPxxx  xxx = any ASCII characters
RCPxxx
IOPxxx
CCPxxx  ( for CP/M 2.2 )
CP3xxx  ( for CP/M 3 )
DOSxxx  ( for CP/M 2.2 )
DO3xxx  ( for CP/M 3 )
BIOxxx
CFGxxx  for configuring following module( s )
xxxxxx a custom module
```

Which named commons should be used by different modules requires some discussion. At the time it loads a new segment, JetLDR will correctly resolve all of the named-common segment addresses shown in Z3COMMON.LIB to those of the system that is in memory.

Initially, I had expected the code in an RCP, for example, to refer to a command processor address by useing offsets from the __CCP__ named common. (That address would be needed, for example, for parsing service or obtaining the names of the command processor's built-in commands.) However, the final versions of Z3PLUS and NZCOM provided the capability of changing the sizes and locations of some Z-System buffers while leaving intact the code in others. Thus, it is possible with NZCOM, for example, to enlarge the named-directory buffer and reload the command processor at a lower address, without reloading the existing RCP. In order to use this advanced feature, an RCP must use only the named commons for segments whose addresses will not change dynamically.

RCP, FCP, and IOP modules should use only __ENV__ references, and obtain the addresses of other segments from the ZCPR3 external environment, The other named-commons are provided for the command processor, BDOS, and special Z3PLUS and NZCOM modules.

To convert an existing source file (e.g., an RCP) to ZRL form, remove any "include" or "maclib" statements for either Z3BASE.LIB or other files with absolute system segment references. Then check the source file for any external references and convert them to named-common base references. For example, to access the wheel byte, replace

```
ld    a,( z3whl )       ; old way
```

with

```
push   hl          ; new way
ld     hl,( z3env+29h )    ; get ptr to wheel byte
ld     a,( hl )
pop    hl
```

Using M80 or an SLR assembler, assemble the file to relelocatable format, and to help distinguish it from an "ordinary" REL file, specify the output file type, or rename it, to type ZRL.

Loading Command Processor, BDOS and BIOS Modules

On CP/M 2.2 systems, JetLDR relocates a CCP package and writes the absolute image to a file in the root directory, because there is no general-purpose method of installing that image into the warm-boot procedure of the host computer. The user can then run SYSGEN or the corresponding utility for his system to install the image. On CP/M Plus systems, JetLDR loads the command processor directly into memory within the Z3PLUS RSX buffer.

JetLDR loads a DOS package to the current BDOS base (xx00h or xx80h) address and executes BDOS function 13 (reset disk system) before returning. JetLDR loads a BIO package to the current BIOS base address (xx00h) as pointed to by (0001h). As soon as a BIO package is moved into position JetLDR executes a *cold* boot; no further messages or loading are attempted!

I have focused on using JetLDR to load Z-System packages in ZRL format. Of course, it will also load named-directory (NDR) and terminal capability (Z3T) files , and will load the traditional absolute-code forms of RCP, FCP, and IOP packages. JetLDR does extensive checking for addressing conflicts before loading a package, and it will not load a package too large for the current buffer size. This will help to catch most mistakes that result from specifying the wrong absolute code package for the current system, an easily made slip if you have several system sizes. In the longer term, I recommend that users switch to using only ZRL files for Z-System segments containing code (i.e. everything but Z3ENV, Z3T and NDR). Only one ZRL file is needed for all system sizes, and its addresses will be correct in every case.

De-installing an IOP

As originally defined, an IOP had no way of being de-installed. Now, before loading an IOP package, JetLDR calls the existing IOP SELECT routine with register B = 0FFh. This value is an invalid device selection. It is used here to enable any new IOP to execute its deinitialization routine before being overloaded by another IOP.

The __ID__ Named Common

JetLDR, Z3PLUS and NZCOM normally require that the named-common segments include only *addresses*, not actual code. However, there is one exception: the __ID__ segment. This relocation base is special—it may include up to 256 bytes of null-terminated ASCII data. Its purpose is to embed identifying information in the ZRL file, such as version number, date, and supported features.

For example, an RCP source file might contain:

```
COMMON  /_ID_/
defb    '3/26/88 vers. 1.2c 1.0K',0dh,0ah
defb    'H, SAVE, P, POKE, SP',0
```

JetLDR will display the ID text when it loads the file. It can also be viewed, somewhat clumsily, with a debugger or disk utility. Perhaps someone will eventually write a simple librarian that scans ZRL files and displays their names and ID fields.

The Extended External Environment Type

ZCPR34, NZCOM and Z3PLUS have extended the traditional ZCPR3 external environment to support additional vital system information. An extended environment is identified by an environment type byte that has a value of 80h or greater. Currently, type 80h is defined to include the following data (future extensions will use higher type values and be compatible with these parameters):

```
z3env+ 08h  db  environment type
z3env+ 34h  dw  valid drives vector
z3env+ 3Fh  dw  CCP base address ( xx00 or xx80 )
z3env+ 41h  db  CCP buffer size ( 80h records )
z3env+ 42h  dw  BDOS base address ( xx00 or xx80 )
z3env+ 44h  db  BDOS buffer size ( 80h records )
z3env+ 45h  dw  BIOS base address ( xx00 )
```

JetLDR supports the extended external environment types (type >= 80h) and provides an automatic way for existing ZCPR3 systems to upgrade the in-memory external environment to include the extensions.

JetLDR's algorithm is this:

1. If the host environment is not type >= 80h, JetLDR assumes a standard system (BDOS size 0E00h, CCP size 800h), and computes the BDOS and CCP addresses from the value at 0001h. It installs these addresses and sets environment type 80h. It sets valid bits for all drives up to the environment's current "maxdrv."
2. Otherwise, it preserves the environment type and system addresses, overlaying them on any ENV segment that may be loaded.

Sharp-eyed readers will note that the extended environment has "stolen" bytes originally defined for CRT1 and Printers 2 and 3. As the Z-System has actually developed, these parameters are of very limited use, and almost no program actively refers to them.

An application can determine whether an environment descriptor contains valid system data by testing bit 7 of the environment type byte (z3env+8). By using JetLDR to load any system segment, you will automatically convert the running environment to type 80h, with current system parameters. Thus, JetLDR provides a painless, assembly-free method for you to upgrade an existing environment to the new standard and be able to use the new extended-environment Z-System tools.

User Configuration of JetLDR

There is has one additional module type, CFG, that adds great flexibility. JetLDR loads a CFG module into one of its own buffers. Once loaded, the CFG code can control or supplant any of JetLDR's normal relocation, validation, and loading steps. This feature has potentially wide application. One immediate use, for example, is to cause JetLDR to write the command processor to the system tracks of the A: drive disk, so that the new file will in fact be warm-booted. JetLDR doesn't provide this service itself, because it is machine-dependent, but a user can write his own CCPCFG module to do so.

A more far-ranging application is to load quite different types of modules, such as resident system extensions (in either main or banked memory). In the next column I will show how JetLDR can simplify and routinize the installation and initialization of CP/M 2.2 RSX modules.

Why the Jet?

JetLDR does load multiple files faster than traditional loaders, JetFind searches a set of files for matching expressions with alacrity, and Z3PLUS is also fleet-footed. In each case I've been able to boost performance by minimizing the time spent stepping between tracks on a physical disk. I hope to discuss some of these Jet techniques in a future column.

Anywhere in Z-80 Space

How would you write a program that must run, without external assistance, anywhere in memory it happens to be located? Such code is *position-independent*. I've needed this capability several times, yet I never stopped to construct a really satisfactory solution, until, in the midst of nailing down the final Z3PLUS code for handling type-4 environment files with Joe Wright and Jay Sage, I suddenly looked at this curious puzzle in a new way.

Before peeking at my solution you may want to try your hand at an answer. To be position-independent, your code must solve two problems: determining what its actual location is, and relocating itself to run at that address.

Most memory references in Z-80 code are absolute; the exceptions are relative jumps and stack operations. If a program can be written to use just those two types of opcodes, it will run anywhere. Unfortunately, that doesn't make for much of a program!

It seems, then, that in general a program must follow one of two paths. Either it is assembled or linked to resolve all of its adddresses to absolutes at the location at which it will actually run. This is a standard COM file. Or, it must be in relocatable format and loaded by a linking loader that resolves the addresses relative to the program's runtime address. This is how EXE files in MSDOS are loaded, and how JetLDR loads ZRL files.

The ANYWHERE Linker/Loader.

Can you devise a way to do the relocation without the assistance of an external loader or linker? ANYWHERE (Listing 2) is my solution.

Imagine that ANYWHERE has just been loaded into memory at some address, perhaps 9123h. How does it work?

ANYWHERE's "Where Am I" routine, at 0006' solves the first problem by calling a routine and then retrieving its own return address from the stack.

WAI temporarily puts a RET instruction at 0000h and then calls that address with an efficient, one-byte RST 0 instruction. The "routine" at 0000h immediately returns to the address "me", but in the process the address of "me" has been pushed (by the "call") and then popped (by the return). So, by decrementing the stack pointer by two bytes, we can retrieve the address of "me" with a "pop hl" at 0017'.

Of course, this wouldn't work if something else had used the stack between the call and completing the decrementing of the stack pointer! Therefore, we ensure no interference by disabling interrupts in this critical section. Actually, the code is more conservative; it is remotely possible that an interrupting routine would refer to location 0000h, so we disable interrupts just before modifying that byte.

Now that ANYWHERE knows the true address of "me",it proceeds to calculate its own starting address ("anywhere") and the address of the code that will be executed ("prog").

While doing so, the routine patches its very first byte to a null—the Z-80 opcode for a NOP instruction. This is an important precaution—it ensures that ANYWHERE will re-execute correctly, by skipping over the relocation routine and jumping directly to the (already-relocated) code at prog. The original byte at the start of ANYWHERE is the opcode for "ld hl,nnnn". It acts as a dummy instruction, simply loading the hl register with the data value generated by the "jr prog" instruction, and allowing execution to proceed in line to the next instruction at 0003'. When the NOP replaces the first byte, the "jr prog" at 0001' becomes effective and control branches to the code at "prog".

Listing 2

ANYWHERE.ASM - Position-Independent Code Linker/Loader

```
        0001  prl equ 1                 ; 0 if spr
0000'         anywhere:
0000' 21              db      21h       ; op code for LD hl,nnnn.  It becomes
                                        ; NOP after initial execution, so that
0001' 18 45           jr      prog      ; re-execution will skip to code

        0004' length equ $+1
0003' 01 0000         ld      bc,$-$    ; patch in length of code here

              ; The Where Am I routine:

0006' 21 0000 wai:    ld      hl,0000 ; point at rst 0 location
0009' E5              push    hl        ; set ix = 0
000A' DD E1           pop     ix        ;
000C' DD 39           add     ix,sp     ; and save sp in ix
000E' 7E              ld      a,(hl)    ; save the byte at 0000
000F' F3              di                ; ensure no interrupts
0010' 36 C9           ld      (hl),0C9h ; plug in RET
0012' C7              rst     0         ; ''call 0000''
0013' 77        me:   ld      (hl),a    ; restore 0000 byte
0014' 3B              dec     sp        ; move stack pointer
0015' 3B              dec     sp        ; ..so we can pop ''me''
0016' FB              ei                ; again safe for interrupts

0017' E1              pop     hl        ; fetch addr of ''me''
0018' 11 FFF2         ld      de,anywhere-me  ; subtract to get..
001B' 19              add     hl,de     ; hl = true location of 'anywhere'
001C' 36 00           ld      (hl),0    ; store NOP, to prevent second ...
                                        ; relocation, should code be re-executed
001E' 11 0048         ld      de,prog-anywhere
0021' 19              add     hl,de     ; hl -> prog

              ; Word-wide PRL relocator.
              ; enter: hl = base of code to be relocated
              ;        bc = length of code
0022'         reloc:
0022' E5              push    hl        ; set de' = base addr of code
0023' D9              exx
0024' D1              pop     de
        0001  if prl
```

The PRL Relocatable Format.

The ANYWHERE code at 0022' relocates the code that begins at "prog" based on a PRL bitmap. PRL-format code consists of relocatable code, assembled relative to 100h, with an appended bit map that indicates which bytes require relocation. There is one relocation bit (0 = absolute, 1 = relocatable) for each byte of code.

Actually, the term Page ReLocatable is a bit (!) of a misnomer, because the bitmap information applies to individual words in the code, and the PRL image can be relocated anywhere in memory. In contrast to the general REL format generated by an assembler in relative mode, the PRL format is very simple. All that it contains is the code and a tag bit for each byte of code. There are no named commons or symbols. (If a DSEG is used, it begins at the byte following the CSEG code).

The SPR (System Page Relocatable) format is similar to PRL. The code image is assembled relative to 0000h, rather than 100h, and DSEG bytes are separated from the CSEG code, with a separate bitmap, so that they could potentially be loaded to a different area of memory. By setting the PRL equate false when assembling ANYWHERE you could use it with an SPR format. But if the code contains DSEG data, you'd have to move the DSEG and the bit maps around; it's simplest to stick with the PRL format.

Relocation requires operating on word values, and ANYWHERE uses stack operations in an unusual and effective way. We set the stack pointer to the address of the word we need to relocate, use a "pop" to fetch a word of code requiring relocation, add the base address to it, and "push" the relocated value back into position. To use the stack pointer this way, we must save its value and disable interrupts.

Initialization of the relocation routine, at 0022', consists of:

a) Setting DE' to the base address of the code (less 100h for PRL code, which is assembled for an origin of 100h).

```
0025' 15              dec     d       ; - 100h if prl assembly
                endif
0026' D9              exx

0027' F3              di              ; no interrupts while we use sp
0028' F9              ld      sp,hl   ; sp -> start of code, lag 1 byte
0029' 3B              dec     sp      ; ..because prl marks the high byte

002A' 09              add     hl,bc   ; add code length, hl ->prl bitmap
002B' 1E 01           ld      e,00000001b ; init the rotation byte
                                      ; it will set CY every 8 bytes
002D' 78        rloop: ld     a,b     ; check byte count
002E' B1              or      c
002F' 28 13           jr      z,rdone
0031' 0B              dec     bc      ; reduce byte count
0032' CB 0B           rrc     e       ; every 8 bits the CY is set
0034' 30 02           jr      nc,rsame ; ..not set

0036' 56              ld      d,(hl)  ; get d = next byte from bitmap
0037' 23              inc     hl      ; and advance bitmap pointer
0038' CB 02     rsame: rlc    d       ; shift bitmap byte left into CY
003A' 30 05           jr      nc,noof ; no relocation needed

003C' D9              exx             ; get word to relocate from ''stack''
003D' E1              pop     hl
003E' 19              add     hl,de   ; relocate by de' = load addr (-100h if prl)
003F' E5              push    hl      ; put it back
0040' D9              exx

0041' 33        noof:  inc    sp      ; -> next byte of code
0042' 18 E9           jr      rloop   ; and loop

0044' DD F9     rdone: ld     sp,ix   ; restore the stack
0046' FB              ei              ; and permit interrupts again

                ;       Assemble with a RET instruction here.
                ;       This ensures that ANYWHERE.COM will run
                ;       harmlessly if no code is overlayed at 'prog'.

                ; Overlay the code, followed by the PRL bit map, here:
                ;
0047' C9        prog:  RET

                       end
```

b) Setting the stack pointer to point to the first byte of the code, minus 1.

c) Setting HL to point to the bitmap.

d) Initializing the E register as a bit counter that will generate a carry on every 8 rotations.

The relocation loop itself, at 002D', is straightforward. We count down the number of code bytes, rotate the bit counter, and fetch a new bitmap byte when the carry flag is set every eighth time. Next we rotate the bitmap byte itself; if the bit is set we fetch the code word and relocate it. Finally we increment the stack pointer to the next byte of code and loop.

The final instructions restore the stack pointer and enable interrupts. Control then proceeds to execute the relocated code, at "prog".

Using ANYWHERE

To use ANYWHERE you need to assemble your code, link it into a PRL file, and then patch that file and its length into ANYWHERE. Instructions are shown in Figure 1.

What's it good for? The elegant and efficient ANYWHERE code serves to illustrate word-relocation using the PRL relocatable format, stack pointer operations, interrupt management, and an answer to the where-am-I question. Beyond that, I can invisage several interesting applications:

An operating system could be user-extensible by providing a buffer of adequate size and interface specifications to accomodate a user-written driver for a new disk device, real-time clock, or other hardware. The user needn't know the location of the buffer to write the driver, and indeed, the operating system may not know until it loads the code.

Programs that require user-coded configuration or hardware driver routines could provide a configuration buffer "anywhere". The hardware-specific code can be assembled separately, once, and patched into any buffer location; the same driver module could thus be used in several programs that require it, regardless of their buffer addresses.

No doubt there are other interesting uses for position-independent code. Send me your best ideas for publication in this space!

Limitations and Extensions

The overhead is 47h bytes for ANYWHERE, plus up to one-eighth of the length of the code for its bitmap (if the code has an unitialized data area at its end, with labels assigned by equates rather than "defs" directives, the bitmap can overlap this area).

This ANYWHERE could be enhanced to save entering registers and pass them to the code at "prog". A second change, which might be needed for some interface specifications, would be to have the code pull itself on top of the ANYWHERE code when it is relocated, overlaying that code and leaving just the relocated image in the buffer beginning at the anywhere address. ■

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, Lisa, Macintosh, DOS 3.3, ProDos; Apple Computer Company. CP/M, DDT, ASM, STAT, PIP; Digital Research. DateStamper, BackGrounder ii, DosDisk; Plu*Perfect Systems; Clipper, Nantucket; Nantucket, Inc. dBase, dBase II, dBase III, dBase III Plus; Ashton-Tate, Inc. MBASIC, MS-DOS; Microsoft. WordStar; MicroPro International Corp. IBM-PC, XT, and AT, PC-DOS; IBM Corporation. Z80, Z280; Zilog Corporation. Turbo Pascal, Turbo C; Borland International. HD64180; Hitachi America, Ltd. SB180 Micromint, Inc.

Where these, and other, terms are used in The Computer Journal, they are acknowledged to be the property of the respective companies even if not specifically acknowledged in each occurrence.

Data Base

A Data Base Primer

by Art Carlson

More programming and consulting hours are spent on business programming than in any other single field. The majority of these activities are concerned with the handling of data. Not all of the work involves data bases as we normally think of them, but a good understanding of data bases is needed because they form the basis for most business programming.

The first large scale computer applications were for processing mailing lists, orders, and accounting transactions—and most of us still think of these examples when someone mentions data bases. It sounds like a very dull, boring subject. This is unfortunate, because data programming covers a much wider range of activities—much more than just printing labels and invoices, and keeping track of account balances.

The early uses merely replaced clerks and bean counters. Business managers told the programmers to make the computers duplicate what people had been doing. The managers had no concept of what additional information and guidance the computers could provide, and they wouldn't have know how to apply the information if they had it. At that time, the U.S. was the business center of the world, and their main concern was to ship the product and put the money in the bank. Today, things are different, and every business (regardless of size) has to extract information from their data and to use it for management and planning. Information processing is also being used for non-business uses such as laboratory numeric data acquisition and analysis.

The Need for Information Processing

There is a tremendous amount of information available—and the amount is increasing rapidly. It's more than we can handle. It's overwhelming, and whether the information is in the form of text or numbers, it's all data.

The emphasis has been on processing data in order to count items or to total accounts, and there is still much to be done in that area (especially for small businesses). The real need is to process the data in order to extract information, and to develop management decisions based on the information.

There are too many variables, and things are changing too rapidly, for individuals to assimilate the information and make the required decisions. We have microprocessor control systems in our cars, and in our automated factories. What we need now are management control systems—and for them we have to learn how to extract information from data files.

Our Goals

There is a lot of published material on how to use the major programming tools, but there is very little published on what the program should do. What is missing is information on the practical aspects of designing and developing a database—the nitty-gritty details on implementing a database rather than an explanation of the tools. Our goal is to cover the basic design of the data structure, what it should do, and how to select the best way to do it.

Our audience is the programmer who is looking for the right solution for a specific application, not the end user who just wants to plug-n-play (although he may very well be the programmer's customer). We will talk about data bases, user interfaces, the uses for data, and the programming tools available. We don't intend to merely publish code for one product, but rather to cover many different products demonstrating their weak points and their strong points.

Our goal is to cover the basic design of the data structure, what it should do, and how to select the best way to do it.

What is a Database?

In simple terms, a database is a collection of related data organized for convenient access. It could be a list of baseball players with their batting averages. It could be the scores for your bowling league. It could be General Motor's inventory of unsold vehicles, on both the company and dealer lots.

In order to justify a database, it has to serve a useful purpose. There has to be some reason why you want to use it to arrange, select, compile, or report the data. If you had data on a dozen ball players, a handwritten list would be sufficient for you to eyeball and pick out the three best hitters. If there were a thousand players, a DBMS would make the selection easier. If you had to locate a vehicle with the right prameters of model, engine, transmission, color, etc. from GM's inventory to fill an order, it would be impossible without a DBMS.

There are several ways of arranging the data, and different systems call the same things by different names. In this series we'll use the following more or less common generic terms. The fundamental elements are: (1) The file, which is a collection of records treated as a unit. (2) A record, which is a collection of data fields treated as a unit, (3) A field, which is a data element. A short example should help clarify this.

In order to keep track of the stock market, you might want to work with the following data:

```
Company Name
Date
Closing Value
Change (in dollars)
```

Each of the above four lines is a field. The combined data in the four fields for one company is a record, and each entry for a different date or a different company is a another record. The collection of all the records is a file. It is convenient to visualize the file

(Continued on page 42)

SCSI for the S-100 Bus

Another Example of SCSI's Versatility

by Dr. John C. Ford, Indiana University of Pennsylvania

The small computer system interface (SCSI) is a standardized protocol for linking computers to other computers, to mass storage devices, or to other devices which observe the protocol. It is perhaps most commonly used for attaching a hard disk to a single computer, although the full standard supports many other tasks. In this article, I'll present a description of one manufacturer's solution to interfacing SCSI devices to an S100 system, including an outline of the software necessary to support SCSI, and an example of an MSDOS program to access the SCSI system.

First, more specifically, what is SCSI? Several articles (1-8) have appeared recently which outline the protocol in some detail. Additionally, the full specification is available (9). Briefly, it consists of the definition of signals in a 50 pin bus which connects the host processors (*initiators*) and the device controllers (*targets*), an electrical specification for those signals, and a description of the system states which define the communications occuring over the bus.

Typical SCSI systems are shown in Figure 1. While this article deals primarily with a simple, single-initiator system such as shown in 1a or 1b, one of the more attractive features of the SCSI protocol is the ability to have multiple initiators sharing the bus, accessing the same I/O devices (Figure 1c).

Communications over the bus occur as follows: a host processor receives a request (e.g., from the operating system) for information which it must acquire from the SCSI storage device. The host then configures itself into the initiator mode, and begins arbitration. This arbitration involves requesting access to the SCSI bus; if the host is the highest priority initiator, other initiators on the bus will relinquish the bus to our example host. After winning arbitration (which is optional in single-initiator systems), the host requests the attention of the selected target. Following response by the desired target, the entire SCSI transaction becomes target-controlled. The initiator monitors the state of the SCSI bus, and

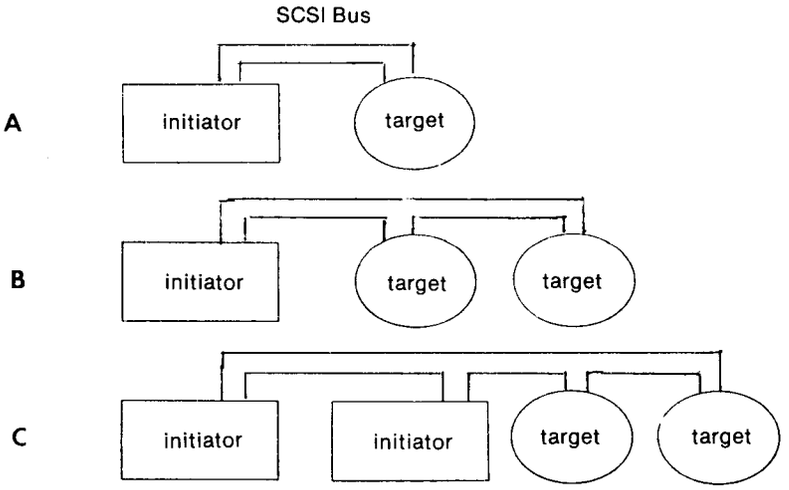


Figure 1: Diagram of possible SCSI configurations. A) Single initiator, single target; B) Single initiator, multiple target; C) Multiple initiator, multiple target.

Listing 1) Phase-driven SCSI low-level format routine, showing the method of accessing the SCSI bus.

```
;       Listing 1 - SCSI test driver
;
;       adapted from Rick Lehrbaum's example SCSI driver in
;       The Computer Journal, issue 26, page 12 ff, 1986
;
;       conversion to 8086 assembly by J. C. Ford, 4/88
;

; macro to slow down S100 and 8086
;       this is to prevent the 8086 from accessing the I/O ports
;       too quickly.  it forces the instruction queue to empty

pause           macro
                jmp  $+2
                endm

; general equates

lf              equ 0Ah
cr              equ 0Dh

; the next address is Lomas specific.  I used the default

ncrbase         equ 40H          ; base address of ncr 5380

; 5380 input-only and input/output registers
```

the target signifies what information is to be put on the bus, who (the target or the initiator) is to put the information on the bus, and, finally, when the entire transaction is complete. After beginning the transaction, the initiator simply responds to the state dictated by the target.

Attaching an S100 system to a SCSI bus may seem inappropriate. After all, we could buy an S100-to-ST412 hard disk controller and forget about the SCSI controller. However, as mentioned above, an S100-to-SCSI board (a host adapter) allows not only our S100 to access the hard disk, but also provides the same access to any other SCSI initiator. It is conceivable to attach multiple S100's to a single hard disk, or an S100 and several MacIntoshes, etc. The only limitation is the address space of the SCSI bus. (There can be upto eight SCSI devices, and a typical SCSI hard disk controller can control two drives.) A second advantage is that once the SCSI adapter software is installed, all SCSI devices can be accessed; adding a tape backup unit becomes a matter of installing a SCSI-to-QUIC controller board and a QUIC tape drive. The software to copy the files from one device to another is essentially the same as the software to copy the files on the disk itself. Yet another advantage, which has yet to fully materialize, is that, as other systems use SCSI and create a demand for SCSI devices, the price for a device and its SCSI controller will become lower than the cost of a specialized S100 interface card and the device. It's also worth mentioning that a wide variety of SCSI controllers are available, including RAM-disks, printer servers, and real-time I/O controllers which attach the SCSI to the STD bus (1). Finally, SCSI is a high-performance bus; the early SCSI cards could support 500 Kbytes/sec and current versions can exceed 1 Mbyte/sec transactions. Information bandwidths of these magnitudes mean that we are not compromising the performance of our systems.

The SCSI electrical definition is essentially a 50 pin parallel bus. Eight of the lines are defined as data lines, nine as control lines, and the bulk as ground lines. While it is possible to use conventional parallel ports to implement a SCSI interface, or to design a specialized SCSI adapter using TTL (10), there is now available a single LSI SCSI interface chip, the NCR 5380, which fully supports the protocol and is as easy to attach to the S100 bus as a conventional LSI parallel port. This article focuses on the Lomas Data Products host adaptor (LDP-HA), which uses this chip. The LDP-HA also supports four RS-232 serial ports and a clock-calendar, although these features are not discussed here. The cost of the

```
ncrcsd      equ ncrbase+0    ; current scsi data register
ncricr      equ ncrbase+1    ; initiator command register
ncrmr       equ ncrbase+2    ; mode register
ncrtcr      equ ncrbase+3    ; target command register
ncrcsbs     equ ncrbase+4    ; current scsi bus status
ncrbsr      equ ncrbase+5    ; bus & status register
ncridr      equ ncrbase+6    ; input data register
ncrrpi      equ ncrbase+7    ; reset parity/interrupt

; the next addresses are also Lomas specific

ncrdack     equ ncrbase+18h  ; dack pseudo-dma
scsi_ctrl   equ ncrbase+0Bh  ; control port for an 8255
scsi_portc  equ ncrbase+0Ah  ; the SCSI ID for the LDP-HA is set by DIP
                             ; switches at this port of the 8255

;  5380 output-only registers

ncrodr      equ ncrbase+0    ; output data register
ncrser      equ ncrbase+4    ; select enable register
ncrsds      equ ncrbase+5    ; start dma send
ncrsdtr     equ ncrbase+6    ; start dma target receive
ncrsdir     equ ncrbase+7    ; start dma initiator receive

;  flag masks for current SCSI bus status register

ncrrst      equ 10000000b
ncrbsy      equ 01000000b
ncrreq      equ 00100000b
ncrmsg      equ 00010000b
ncrcd       equ 00001000b
ncrio       equ 00000100b
ncrsel      equ 00000010b
ncrdbp      equ 00000001b

; flag mask for bus and status register

ncrphm      equ 00001000b  ; phase mismatch

; flag masks for SCSI status

BUSY_STATUS   equ 08h
CHECK_STATUS  equ 02h

target_ID     equ 01h        ; SCSI device 0 - NOTE this is system specific
                             ; the address used here will depend on the target
                             ; address of the controller board in your system

code          segment
assume        cs:code, ds:code, es:code, ss:stack

start:
main:
scsi_trial:
              mov  ax, code
              mov  ds, ax
              mov  es, ax
              call hdinit       ; this also does SCSI init
              call scsireset
              call zero_unit
lbusy:        call test_ready
              mov  al, status
              test al, BUSY_STATUS
              jnz  lbusy                  ; loop if busy
              test al, CHECK_STATUS
              jz   continue
              call req_sense              ; there was an error
continue:
mbusy:        call mode_select
              mov  al, status
              test al, BUSY_STATUS
              jnz  mbusy
              test al, CHECK_STATUS
              jz   cont2
              call req_sense
```

```
cont2:
fbusy:      call format_unit
            mov  al, status
            test al, BUSY_STATUS
            jnz  fbusy
            test al, CHECK_STATUS
            jz   cont3
            call req_sense
cont3:
            mov  ah, 4Ch
            int  21h              ; exit


hdinit:     mov  al, 92h          ; set up 8255 mode
            out  scsi_ctrl, al
            pause
            mov  al, 55h
            out  scsi_portc, al

;           fall through into a 5380 reset

ncrinit:    xor  ax, ax           ; just reset 5380
            out  ncricr, al
            pause
            out  ncrmr,  al
            pause
            out  ncrtcr, al
            pause
            out  ncrser, al
            ret

scsireset:  mov  ax, 0000000010000000b
            out  ncricr, al
            mov  al, 100          ; generate long delay
rst1:       dec  ax
            jnz  rst1
            xor  ax, ax
            out  ncricr, al
delay:      mov  cx, 0            ; set up counter
rst2:       mov  ax, cx
            dec  cx
            jnz  rst2
            in   al, ncrrpi       ; reset interrupt indicator
            ret

disbyte:    push ax
            push bx
            push cx
            push dx
            push ax               ; save byte
            mov  cl, 4
            shr  al, cl           ; get high nibble
            call disnibble
            pop  ax
            and  al, 0fh          ; get low nibble
            call disnibble
            mov  al, 20h          ; output a space between bytes
            mov  ah, 2            ; DOS output byte
            int  21h
            pop  dx
            pop  cx
            pop  bx
            pop  ax
            ret

disnibble:  add  al, 30h          ; change to ASCII
            cmp  al, 39h
            jle  dn2              ; jump if less than or equal
            add  al, 7            ; change hex to ASCII
dn2:        mov  ah, 2            ; DOS output byte
            int  21h
            ret


; if you're going to add arbitration code, it would be done before
; calling select
```

board ($375) is quite reasonable in comparison to the same (and other) manufacturer's charge for S100 hard disk controllers.

[The SCSI controller used in my system (the Adaptec ACB 4000) was $100. While this controller is appropriate for single-initiator systems, it will be necessary to switch when I opt to add a second initiator to the system. However, it will not be necessary to change the software, excepting perhaps the device-specific information in the formatting routines.]

Lomas supplies a technical manual with the LDP-HA, although I have found numerous errors in it. Error is perhaps overly harsh, but I find it hard to describe otherwise differences between the jumper numbering on the board and in the manual, omission of board jumper numbers, and an incorrectly modeled example driver. Lomas does include a schematic, although the numbering of jumpers on the schematic is not in full agreement with the board, and one set of jumpers is unnumbered on the schematic. Finally, the selection of the wait-states and of the SCSI ID number would have been easier if the board had clearly indicated the proper orientation of the jumpers. (In other words, I couldn't tell whether I was selecting seven wait states, or none!) In all fairness, Lomas does not encourage direct sales, so that presumably you will be able to direct questions to your dealer. Further, the individuals at Lomas were extremely helpful when I called them.

Excepting omissions in the screen mask indicating jumper and cable orientation, such as those mentioned above and the orientation of pin 1 on the SCSI bus, the board appears well designed. There were no wire-wrap jumpers or cut traces, and the traces are nicely organized, indicating that the design is solid. The board includes card ejectors, a nice touch. It comes with jumpers installed for a full LDP system, i.e., the address is 40H, compatible with Lomas's other I/O cards, and it ran without reconfiguration.

My system is a Lomas Data Products S100-PC, running MSDOS 2.11. The processor card is the LDP Lighting 1, with the 8086 and a ROM monitor. For an additional $35, Lomas supplied me with replacement ROMs which contain routines to access the SCSI adaptor, a binary image which Lomas says will boot DOS from the hard disk, a modified version of IO.SYS compatible with the hard disk, the assembly source code for a loadable MSDOS driver to read and write sectors from the disk, and a utility program to create a data file needed in the assembly of the driver. (It is necessary for the user to supply MASM and LINK, or equivalents.) Lomas does NOT supply

information on how to directly access the SCSI, although that information can be obtained from a listing of the monitor ROM source (available for $15). This latter information would be useful, since a hard disk needs a low-level format before the DOS format (which simply sets up the directory structure). Unfortunately, Lomas cannot provide a low-level format routine, since this command varies among controllers, and would in any event vary according to the capacity of the hard disk used.

Lomas does provide a generalized example SCSI driver in the manual, which could be entered, modified for MSDOS (it is written for CP/M-86), and used, although there is no real description of how to utilize it. However, SCSI is a state-driven protocol: the driver should not count the bytes output, nor assume which state is occuring, but rather let the target control the information flow (2, 3). The SCSI standard specifies that certain (six-byte) commands are mandatory, but the manufacturer is permitted to incorporate into their design optional and/or vendor-unique commands, which may be ten-byte commands. While the end-user may not care, the systems integrator may find that use of these commands results in higher performance. The LDP driver accomodates both types of commands, but does so by expecting the calling routine to supply it with a count of the number of bytes to transfer.

Further, the Lomas ROM monitor does not seem to be compatible with the ACB 4000. Attempts to use the provided Lomas DOS driver (which relies on the ROM routines) resulted in no response; moreover, the error light on the ACB 4000 lit. This suggested a software incompatibility, and the need to create my own SCSI routines.

No problem. Part of Rick Lehrbaum's excellant SCSI series included an example SCSI driver for the Z-80 (5). I converted this to 8086 assembly language and attempted to perform a low-level format, to no avail. After much teeth-gnashing, I realized that Mr. Lehrbaum's code has a flaw. In the original code, the routine to output a byte (wscsi) enabled the data bus upon entry, watched for the REQ, then enabled the acknowledge (ACK). However, in the 5380, enabling ACK involves a write to the same register which controls the data bus enable, and Mr. Lehrbaum's code disabled the data bus when it enabled the ACK. This resulted in the LDP-HA removing the data from the bus at the same time it told the target to accept it!

The code presented in Listing 1 overcomes these problems. Since this code directly accesses the 5380, it eliminates the need to use the ROM routines, and thus makes possible the creation of a SCSI

```
;************** ENTRY POINT FOR SCSI DEVICE ACCESS ***********

select:     xor  ax, ax            ; clear register in order to set all
            out  ncrtcr, al        ; Assert bits in TCR. Prepare phase
            mov  al, target        ; get target ID from memory
            out  ncrodr, al
            mov  al, 00000101b     ; set Assert Data Bus and SEL bits
            out  ncricr, al
            pause
waitblp:    in   al, ncrcsbs       ; get current scsi bus status
            test al, ncrbsy        ; look at busy line
            jz   waitblp           ; wait for busy
            xor  ax, ax
            out  ncricr, al        ; clear SEL and release data bus

; drop through into phase

;************** MASTER BUS PHASE PROCESSING ROUTINE ************

phase:      xor  ax, ax
            out  ncrmr, al          ; reset ncr ctrl registers
            mov  ax, offset message ; ready message_pointer
            mov  message_pointer, ax
            xor  ax, ax
            out  ncricr, al
            mov  ax, offset status  ; now ready status_pointer
            mov  status_pointer, ax
ph1:        in   al, ncrcsbs        ; check for BSY active
            test al, ncrbsy
            jnz  ph2
            ret                     ; return if BSY drops out
ph2:
            test al, ncrreq
            jz  ph1                 ; not valid if REQ not valid
            and  ax, 0000000000011100b      ; get MSG, C/D, I/O
            shr  ax, 1                      ; move it to set up for tcr
            shr  ax, 1
            out  ncrtcr, al
            mov  bx, offset phasetable
            add  bx, ax
            add  bx, ax             ; point to correct phase
            jmp  cs:[bx]            ; go do it

phasetable: dw   dataout       ; jump table for SCSI modes
            dw   datain
            dw   cmdout
            dw   statin
            dw   undefined
            dw   undefined
            dw   msgout
            dw   msgin

dataout:    mov  bx, offset datptr
            jmp  wscsi

datain:     mov  bx, offset datptr
            jmp  rscsi

cmdout:     mov  bx, offset cmdptr
            jmp  wscsi

statin:     mov  bx, offset status_pointer
            jmp  rscsi

msgout:     mov  bx, offset message_pointer
            jmp  wscsi

msgin:      mov  bx, offset message_pointer
            jmp  rscsi

undefined:
            ret
```

```
wscsi:
                push bx             ; push the address
                pop  di             ; di points to address location
                                    ; of pointer
                mov  bx, [di]       ; bx points to data
wscsi1:         in   al, ncrbsr     ; check for phase mismatch
                test al, ncrphm
                jz   gotophase
                in   al, ncrcsbs    ; check for busy
                test al, ncrbsy
                jz   gotophase
                test al, ncrreq
                jz   wscsi1         ; loop unti req or phase change
                mov  al, [bx]       ; get the data to send
                out  ncrodr, al     ; send it
                inc  bx             ; increment pointer
                mov  [di], bx       ; store it
                mov  al, 01h
                out  ncricr, al     ; assert data bus
                pause
                mov  al, 00010001b  ; set ack and data bus
                out  ncricr, al
                pause
waitreq:        in   al, ncrcsbs    ; wait for req to go away
                test al, ncrreq
                jnz  waitreq
                xor  ax, ax
                out  ncricr, al     ; and when it does, clear ack

;               drop through to phase

gotophase:      jmp  phase

rscsi:
                push bx
                pop  di
                mov  bx, [di]
                in   al, ncrbsr     ; check for phase mismatch
                test al, ncrphm
                jz   gotophase
                in   al, ncrcsbs    ; check for busy
                test al, ncrbsy
                jz   gotophase
                test al, ncrreq     ; check for req
                jz   rscsi          ; loop until phase change, etc.
                in   al, ncrcsd     ; get data
                mov  [bx], al       ; save it
                inc  bx             ; increment pointer
                mov  al, 00010000b  ; set ack
                out  ncricr, al
                pause
noreq:          in   al, ncrcsbs    ; wait for req to go away
                test al, ncrreq
                jnz  noreq
                xor  ax, ax
                out  ncricr, al
                jmp  phase           ; loop back

test_ready:     mov  cmdptr, offset tr_cmd
                mov  datptr, offset datbuf
                call select
                ret

zero_unit:      mov  cmdptr, offset zu_cmd
                mov  datptr, offset datbuf
                call select
                ret

req_sense:      mov  cmdptr, offset rs_cmd
                mov  datptr, offset datbuf
                call select

;               now add code to print out the four bytes of returned data
;               this returned data contains specific information about
;               the nature of the error
```

driver without the ROM source listing. Be aware, however, that Listing 1 is NOT an MSDOS SCSI driver, although it could be changed into one.

Listing 1† consists of two major parts. The first consists of the generalized SCSI routines. These are the SCSI initialization, SCSI reset, select, phase, and associated subroutines. The initialization code simply initializes the LDP-HA's parallel port to read the SCSI ID and sets the 5380 into initiator mode, while the reset routine issues a reset command to the SCSI bus. While this is a good idea when initializing the bus, it may also cause the various SCSI devices to go through long initialization routines.

The actual SCSI transactions are performed by calling the `select` subroutine. On entry, the variable `cmdptr` must point to a buffer containing the SCSI command, `datptr` must point to a buffer which will receive data from the SCSI target or which contains data to be sent to the target, and the variable `target` must contain the desired SCSI device ID. Pointers to `status` and `message` will be set up in the routines. Select will awaken the appropriate target and pass control to `phase`. Phase monitors the state of the SCSI bus signals, and passes control (via jumps through `phasetable`) to routines which set up the appropriate pointers and either read (`rscsi`) or write (`wscsi`) a byte from or to the SCSI bus. Unlike the Lomas driver, this code is completely state-driven. After performing each bus read or write, the routine passes control back to phase.

When the target signifies the completion of the transaction, control is returned to the point in the program which called select. Status contains data representing whether or not the transaction was completed—this is not the same as successful completion. On the ACB 4000, a status of 0 indicates completion, a status of 2 signifies the availability of error information (check status), and a status of 8 means that the selected device is busy and could not accept the command. A busy status requires repeating the entire SCSI transaction, while the check status should be followed by a SCSI request sense command, which will result in more complete error information being transferred.

The routines rezero, test-ready, sense, mode-select, and format demonstrate the use of select to handle SCSI transactions. The SCSI commands are handled by

† This was my first attempt at 8086 assembly language. If there are instructions which represent gross inefficiencies, I apologize and look forward to your suggestions. However, the program does work.

initializing cmdptr and datptr to point to appropriate buffers containing the information. Mode-select is particularly instructive, because this command sends information about the capacity of the disk to the ACB 4000. The ACB 4000 stores this information on the disk, and upon subsequent power-ups, rereads it. Thus the ACB 4000 does not have to be retold what size disk it is dealing with.

Main in this program performs a low-level format, expecting the ACB 4000 to be attached to an ST 225 hard disk as unit 0. It simply calls the routines described above in the appropriate sequence and performs some testing of status. This not only provides you with a low-level SCSI format routine, should you be in my position when I wrote this, but also gives a general example of direct access to the SCSI bus.

However, this driver is not complete. Just as Mr. Lehrbaum's model, this driver does not support the entire SCSI protocol, including arbitration, disconnect-reconnect, etc. In order to utilize this code as a generalized MSDOS driver, it is necessary to add conversion of the DOS block-device driver information into the appropriate SCSI commands and to establish the appropriate pointers.

In spite of the vaguenesses and inconsistencies of the manual, I have found the LDP-HA to have integrated easily into my system. I am already attempting to increase the sophistication of the driver to support arbitration and to use the pseudo-DMA mode of the 5380, and am looking forward to being able to expand my system into a multiprocessor data acquisition system, using the SCSI bus as the interprocessor link.

Acknowledgement

The funds to purchase the LDP-HA were made available to me by The College of Natural Science and Mathematics, Indiana University of Pennsylvania, and the release time necessary to perform this work was funded by the Provost's Scholarly Activities Fund.

```
                mov  dx, offset errormsg
                mov  ah, 09h
                int  21h                    ; output error message
                mov  bx, offset datbuf
                mov  cx, 04h                ; set up counter
loopout:
                mov  al, [bx]               ; get byte
                inc  bx                     ; increment pointer
                call disbyte                ; display byte in al as hex
                loop loopout
                ret

format_unit:    mov  cmdptr, offset fu_cmd
                mov  datptr, offset datbuf
                call select
                ret

mode_select:    mov  cmdptr, offset ms_cmd
                mov  datptr, offset mode
                call select
                ret

target          db  target_ID           ; a generalized routine would put
                                        ; appropriate target ID here before
                                        ; calling select
message         db  ?                   ; actual variable locations
status          db  ?

; I used the following pointers to allow me to use the same form for
; the statin, msgin, and msgout routines as for the datain, etc.  Rick
; Lehrbaum's example didn't, but that was because his routine didn't
; return to PHASE after each byte, but rather output (input) a series of
; bytes until the controller was happy

message_pointer  dw  2( ? )             ; pointers to variable locations
status_pointer   dw  2( ? )

cmdptr          dw  2(?)                ; pointers to storage locations
datptr          dw  2(?)

datbuf          db   512 dup (0)  ; general purpose buffer

; notes on the following -

;       first of all, these commands are all set up for the Adaptec 4000
;       SCSI controller - while most of the commands are SCSI standard, the
;       ordering of data for the mode select isn't, so watch yourself.

;       the logical unit zero is specified by the three HIGH bits of
;       of the byte.  If you want to attach two disks, you need to
;       include code to select between the different l.u. numbers

tr_cmd          db   0              ; test unit ready command
                db   0              ; logical unit zero
                db   0, 0, 0, 0     ; reserved

zu_cmd          db   1              ; rezero unit command
                db   0              ; logical unit zero
                db   0, 0, 0, 0     ; reserved

rs_cmd          db   3              ; request sense command
                db   0              ; logical unit zero
                db   0, 0, 4, 0     ; request four bytes of info
```

```
fu_cmd      db   4            ; format unit command
            db   0            ; logical unit zero
            db   0, 0, 2, 0   ; interleave of 2

ms_cmd      db   15h          ; mode select command
            db   0            ; logical unit zero
            db   0, 0, 22, 0  ; going to send 22 bytes

;           the following info is set up specifically for the
;           Adaptec 4000 SCSI controller driving an ST225 as l.u. 0

mode        db   0,0,0,8          ; mode select parameter list
            db   0,0,0,0,0,0,2,0  ; extent descriptor list (512 byte blks)
            db   1
            db   2, 105           ; number of tracks (617 total)
            db   4                ; number of heads
            db   1, 44            ; reduced write current cylinder (300)
            db   1, 44            ; write precompensation cylinder
            db   1                ; landing zone beyond outermost track
            db   1                ; step rate, 28 usec

errormsg    label
            db    cr, lf,
            db  ' SCSI error reported - request sense data follows '
            db    cr, lf, '$'

code        ends

stack       segment para stack 'stack'
            db  200 dup (00h)     ; 200 bytes of stack space
stack       ends

            end
```


References

1 R. Lehrbaum, The SCSI Interface: Introductory Column To A Series, The Computer Journal, vol 22, 25 (1986).

2 R. Lehrbaum, The SCSI Interface: Introduction To SCSI, The Computer Journal, vol 23, 7 (1986).

3 R. Lehrbaum, The SCSI Interface: The SCSI Command Protocol, The Computer Journal, vol 24, 9 (1986).

4 R. Lehrbaum, The SCSI Interface: Building A SCSI Adapter, The Computer Journal, vol 25, 23 (1986).

5 R. Lehrbaum, The SCSI Interface: Software for the SCSI Adapter, The Computer Journal, vol 26, 12 (1986).

6 R. Lehrbaum, Using SCSI for Real Time Control: Separating the Memory and I/O Buses, The Computer Journal, vol 28, 25 (1987).

7 R. Lehrbaum, Using SCSI for Generalized I/O: SCSI Can Be Used for More Than Just Hard Disks, The Computer Journal, vol 31, 6 (1987).

8 H. Tytus, Interfacing Using The SCSI Bus, Micro/Systems Journal, vol 2, 46 (1986).

9 ANSC X3T9.2 SCSI Specification, Computer and Business Equipment Manufacturer's Association.

10 ACB-4000 Series User's Manual, Adaptec, Inc., 1985.

11 LDP-SCSI Owner's Manual, Rev 0, Lomas Data Products, Inc., 1987. ■

Use a Mouse on Any Hardware

Implementing the Mouse on a Z80 system

by Richard Rodman

The Logimouse® R7 and C7 mice (Logitech, Inc., 6505 Kaiser Dr., Fremont, CA 94555, (415) 795-8500) are widely available and interface via a standard RS-232 serial port. It seemed that this mouse could be easily used with my Z-80 system. After some experimentation, this proved to be true.

The Logimouse R7, which I used, has an external power supply connecting to a DB-25 female connector. Data comes out on pin 3; pin 7 is grounded. I'm not sure if it's necessary to drive DTR on pin 20, but I did. The Logimouse C7 does not require an external power supply.

The mouse is held in the hand with the cord proceeding away, in the opposite direction from the arm. The palm rests on the flat area on the top of the mouse, and the fingers operate 3 buttons on the far end of the mouse. This is important—the directions "up," "down," "left" and "right" below depend on this orientation.

The mouse sends data in 5-byte packets at 1200 baud. The first byte of the packet has bit 7 set, and bits 0, 1, and 2 set or reset according to the status of the 3 mouse buttons. Bit 0 will be 0 if the right button is down, or 1 if it is up. Bit 1 contains the status of the middle button, and bit 2 that of the left button. Bits 3 through 6 are all zero.

The second and fourth bytes are movement values in the horizontal or X direction (left to right). A value which is negative indicates motion to the left; a value which is positive indicates movement to the right.

The third and fifth bytes are movement values in the vertical or Y direction (up and down). A value which is negative indicates motion downward (toward the user); a value which is positive indicates motion upward (away from the user).

The program given in the Listing is written in Software Toolworks C for CP/M. It tracks the movement of the mouse with the cursor of a video terminal, and displays the status of the three buttons in the lower right corner of the screen. The program has in-line assembly code for a Z80-CTC and a Z80-SIO working together. To modify for other serial port

```
/* --- MOUSE.C --- Read serial Logimouse on arbitrary hardware
    Implementation given is for Software Toolworks C.
    Hardware port logic is for Z80 SIO f CTC.

    By Richard Rodman.  Any use whatsoever of this code is heartily
    encouraged.
Usage:
    If a command line parameter is used, it will simply display
    the bytes received in hex.  Otherwise, the cursor will track
    movement of the mouse, and the button status of each button will
    be displayed.

    Press esc to stop.

Mouse data packet structure:
    First byte: 10000LMR    L = 0 if left button down, else 1
                            M = 0 if middle button, else 1
                            R = 0 if right button, else 1
    Second byte:    delta x, negative = left, positive = right
    Third byte: delta y, negative = down, positive = up
    Fourth byte:    Another delta x value
    Fifth byte: Another delta y value

    The entire packet is sent if anything changes.

History:
    870706 rr orig version */

#include ''tprintf.c''        /* for debugging only */

int cursx, cursy;       /* cursor location */

main( argc, argv )
int argc;
char *argv[];
{
    int i, byte[ 5 ];
    char butstr[ 4 ];

    minit();        /* init serial port for mouse */

    clrscn();       /* clear the terminal screen */

    butstr[ 3 ] = '\0'; /* terminate string for display */

    cursx = 40;
    cursy = 12;     /* center the cursor */
    goxy( cursx, cursy );   /* and display it */

    while( 1 ) {        /* do forever */

/* Check local console for press of ESC key. */
```

```
        if( bdos( 6, 0x00FF ) == '\033' ) break;

/* Check the mouse for a character */

        if( mstat() ) {

/* If command line parameter was present, just display it. */

            if( argc > 1 ) printf( "%02x ", minput() );
                else {

/* Read the 5-byte packet from the mouse */

                for( i = 0; i < 5; ++i ) {
                    while( ! mstat() ) /* wait */ ;
                    byte[ i ] = minput();
                }

/* Process buttons in byte 0 */

                butstr[ 0 ] = butstr[ 1 ] = bjtstr[ 2 ] = ' ';
                if( ! ( byte[ 0 ] f 0x04 )) butstr[ 0 ] = 'L';
                if( ! ( bste[ 0 ] f 0x02 )) butstr[ 1 ] = 'M';
                if( ! ( byte[ 0 ] f 0x01 )) butstr[ 2 ] = 'R';
                goxy( 75, 23 );
                printf( butstr );

/* The cursor movements are signed characters.  Process these.  Use a
    slew of 256 as full-screen.  The Y movement needs to be negated. */

                cursx += 80 * ( extend( bste[ 1 ] )
                    + extend( byte[ 3 ] )) / 256;
                cursy -= 24 * ( extend( bste[ 2 ] )
                    + extend( byte[ 4 ] )) / 256;

/* Make sure the cursor stays on the screen */

                if( cursx < 0 ) cursx = 0;
                if( cursx > 79 ) cursx = 79;
                if( cursy < 0 ) cursy = 0;
                if( cursy > 23 ) cursy = 23;

                goxy( cursx, cursy );
            }
        }
    }
}

/* extend sign on integer */

int extend( c )
int c;
{
    if( c > 128 ) c -= 256;
    return c;
}

/* clear the terminal screen */

clrscn()
{
    printf( "\033E" );
}

/* go to x, y */

goxy( x, y )
{
```

hardware, modify the routines `minit( )`, `mstat( )` and `minput( )`.

Since I couldn't determine the reason for the two movement values in each direction, I simply added them. This gives values in each direction of −256 to +254. This value, once calculated, needs to be scaled to the resolution of your display, so that moving the mouse produces proportional movement on the screen. Because I was using a normal 80 by 24 video terminal, I scaled the horizontal values by multiplying by 80 and dividing by 256, the vertical values by multiplying by 24 and dividing by 256. The resulting movement values are added to the current X and Y cursor position.

If you use a graphics display, multiply by your actual horizontal and vertical resolutions instead. Remember to insure that the X and Y values don't exceed the dimensions of the display.

You may desire to divide by a value less or greater than 256. Smaller values make the mouse respond with greater movement; larger values with less movement. The "best" value would be a value which allows accurate positioning of the cursor anywhere, without requiring the user to constantly "row" the mouse (repeatedly rolling the mouse, then picking it up and moving it back without rolling).

Since you have control of this parameter, you can make your targets large and your divisor small, and eliminate the rowing for all but the most crowded of desks.

Another point to remember: The mouse is a *relative* movement device. It does not keep track of its absolute position. Therefore, your program can only follow it when it actively examines the serial port. I suggest checking for characters periodically, and processing the mouse movement whenever a packet is waiting. This can be done by converting the `main( )` in the listing into a function, and removing the `while( 1 )` loop. This function then should be called periodically to update the mouse position while other program activity is going on.

```
    printf( ''\033Y%c%c'', y + 32, x + 32 );
}

#asm
;Z-80 SIO and CTC routines - IMS 740 slave board

CTC EQU 28H
SIO EQU 2CH
UART    EQU 1       ;uart number on SIO
CHAN    EQU 1       ;channel number on CTC

DATA    EQU SIO+UART
STAT    EQU SIO+UART+2

;   Baud Rate Divisor

B1200   EQU 16      ;baud rate divisor
#endasm

minit()
{
#asm
;   Initialize SIO

    MVI A,18H           ;Channel Reset
    OUT STAT
    MVI A,4
    OUT STAT
    MVI A,11000100B     ;8 bits xmit, *64 Async
    OUT STAT
    MVI A,5
    OUT STAT
    MVI A,11101010B     ;set DTR on, RTS on
    OUT STAT
    MVI A,3
    OUT STAT
    MVI A,11000001B     ;8 bits recv
    OUT STAT
    MVI A,1
    OUT STAT
    MVI A,0         ;no interrupts
    OUT STAT

;   Set Baud

    MVI A,01000111B     ;Counter mode, free-run, value follows
    OUT CTC+CHAN
    MVI A,B1200
    OUT CTC+CHAN
#endasm
}

/* check serial port status */

mstat()
{
#asm
    IN  STAT
    ANI 1
    MOV L,A
    MVI H,0
#endasm
}

/* read serial port data */

minput()
{
#asm
    IN  DATA
    MOV L,A
    MVI H,0
#endasm
}

#include ''stdlib.c'' /* include code for bzos() and other functions */

/* end of mouse.c */
)ry   6eMUX5-4 BAK _y ZeMUX5-1 DWG hqy . pLeMUX5-2 BAK [y 8 6eMUX5-3 BAK n[y ?
```

■

Systematic Elimination of MS-DOS Files

Part 2—Subdirectories and Extended DOS Services

by Edwin Thall

Dr. Edwin Thall, Professor of Chemistry at The Wayne General and Technical College of The University of Akron, teaches chemistry and computer programming.

MS-DOS files may be created and manipulated by two different approaches. In Part I (Issue #32), the file control block (FCB) method was described and DATACIDE, a utility capable of removing files from the root directory, presented. In Part II, the more versatile extended DOS services, introduced with version 2, are explored. These functions, besides providing a more convenient method to access files, fully support the hierarchical file structure. XFILE, a utility similar to DATACIDE but able to remove files from any directory, will be presented.

Extended DOS Services

Extended DOS services specify files by either a code (file handle) or an ASCIIZ string. A listing of selected functions, designated "H" for handle and "A" for ASCIIZ string, is provided in Table 1.

Function	Operation	
39H	Create subdirectory	(A)
3AH	Delete subdirectory	(A)
3BH	Create file	(A)
3DH	Open file	(A)
3EH	Close file	(A)
3FH	Read file or device	(H)
40H	Write to file or device	(H)
41H	Delete file	(H)
4EH	Search first match	(A)
4FH	search next match	(A)

Table 1. Extended DOS Functions (A=ASCIIZ string, H=handle)

The ASCIIZ format consists of a series of conventional ASCII characters terminated by a byte of zero (00H). Here is an example of an ASCIIZ string:

'A: \ SUB \ NAME.EXT',0

The drive (A:), path (\ SUB), and file name (\ NAME.EXT) are stored in memory as:

```
41 3A 5C 53 55 42 5C 4E 41 4D 45 2E 45 58 54 00
```

The null byte (00H) cannot be entered directly by way of the keyboard since keystrokes of zero and Alt<0> return 30H and nothing, respectively. The backslash (\) serves as path separators.

Whenever you create or open a file (functions 3CH/3DH), you pass the name of the file as an ASCIIZ string. DOS maintains the file's control information in its own area, and returns a number in the AX register. This number, known as the file handle, must be referred to for future access of the file.

The extended DOS services rely on file handles, instead of an FCB, to keep track of files and input/output devices. The number of files that DOS can open concurrently may be declared during the boot with CONFIG.SYS (FILES = N). If the number of files are not specified, the value N = 8 is used by default. During the boot, DOS assigns file handles 0000-0004 to the following I/O devices:

0000	Standard input device (keyboard)
0001	Standard output device (screen)
0002	Standard error output device
0003	Standard auxiliary device (COM1)
0004	Standard printer device (printer)

File handles for standard devices are preassigned and do not have to be opened or created. Any message can be sent directly to the screen by means of file handle 0001. To do so, load the DEBUG utility and execute the series of instructions (omit comments):

```
A>DEBUG
-A100
DS:0100  MOV AH,40      ;WRITE TO DEVICE
DS:0102  MOV BX,0001    ;OUTPUT TO SCREEN
DS:0105  MOV CX,11      ;17 BYTES
DS:0108  MOV DX,010F    ;POINT TO MESSAGE
DS:010B  INT 21         ;CALL DOS
DS:010D  INT 20         ;RETURN
DS:010F  DB  'HANDLES ARE HANDY'
<ENTER>  2X
-G
HANDLES ARE HANDY
```

Creating Files

Extended DOS function 39H creates a subdirectory by pointing the DX register to an ASCIIZ string. Let's create \ SUBDIR in the root directory of a floppy disk. To a newly formatted 360K floppy, which includes the operating system and the DEBUG utility, enter the following program:

```
-A100
DS:0100  MOV AH,39           ;CREATE SUBDIRECTORY
DS:0102  MOV DX,0109         ;POINT TO ASCIIZ STRING
DS:0105  INT 21
DS:0107  INT 20
DS:0109  DB  'A:\SUBDIR',0   ;ASCIIZ STRING
<ENTER>  2X
-G107
```

To determine the status of this operation, the program is executed up to offset 107H. The register dump signals a successful operation when the carry flag is clear (NC). Return to DOS and display the root directory.

```
-Q
A>DIR
```

The file appears in the listing as SUBDIR <DIR>, the designation for a subdirectory file. To see the entries stored in this newly formed subdirectory, enter:

```
A>DIR \SUBDIR
```

You should be looking at two listings, a single-period and a double-period.

Normal files are created in a manner similar to subdirectories, except function 3CH is called and DOS passes the 16-bit file handle in the AX register. First, create \SUBDIR\HANDLE.TXT, and then write to the file:

```
-A100
DS:0100  MOV  AH,3C        ;CREATE FILE
DS:0102  MOV  CX,0000      ;NORMAL FILE
DS:0105  MOV  DX,0118      ;POINT TO ASCIIZ STRING
DS:0108  INT  21H
DS:010A  MOV  BX,AX        ;STORE HANDLE IN BX
DS:010C  MOV  AH,40        ;WRITE TO FILE
DS:010E  MOV  CX,16        ;WRITE 22 BYTES
DS:0111  MOV  DX,012D      ;POINT TO BUFFER
DS:0114  INT  21H
DS:0116  INT  20H
DS:0118  DB   'A:\SUBDIR\HANDLE.TXT',0
DS:012D  DB   'HANDLES ARE HANDY'
<ENTER>  2X
-G116
```

The register dump indicates that the write operation was successful (NC), 22 bytes were written to the file (AX = 0016), and the file handle is 0005 (BX = 0005). The handle assigned to the first open file is always 0005. When nothing is declared for FILES in CONFIG.SYS, three additional files may be opened and assigned handles 0006, 0007, and 0008.

Extended DOS function 3EH closes the file and releases the handle for reuse. If the file was modified, these changes are updated in the directory during the close. Set the instructional pointer register to 100H and close \SUBDIR\HANDLE.TXT, the file identified by handle 0005.

```
-RIP
IP  0116
:100

-A100
DS:0100  MOV  AH,3E        ;CLOSE FILE
DS:0102  MOV  BX,0005      ;FILE HANDLE
DS:0105  INT 21
DS:0107  INT 20
<ENTER>  2X
-G107
```

The carry flag (NC) should indicate that the file was successfully closed. Return to DOS and issue the TYPE command to read the contents of this newly created file.

```
-Q
A>TYPE \SUBDIR\HANDLE.TXT
HANDLES ARE HANDY
```

What's Inside a Subdirectory

Every disk has one root directory from which all searches begin. The size and location of the root directory is established during the format operation. While the 360K format assigns logical sectors 5-11 to store the root directory sectors, subdirectories are maintained as ordinary files and may be located anywhere on disk. The only limit on the size or number of subdirectories is available disk space.

Both root and subdirectory require the same 32-byte field (see Table 5 in Part I) to maintain control information for each entry. These entries may be data files or pointers to other subdirectories. Return to DEBUG, and invoke the L command to load the disk's 7 root directory sectors, beginning with logical sector 5, into offset 100H.

```
-L100 0 5 7
-D100,1A0
```

You should be viewing the first 160 bytes of the disk's root directory (Figure 1). The first two entries (IBMBIOS.COM and IBMDOS.COM) have an attribute byte of 27H (offsets 10BH & 12BH), the designation for system/hidden files. The next two entries (COMMAND.COM and DEBUG.COM) have attributes of 20H (offsets 14BH & 16BH) and, therefore, are normal files. The last entry, SUBDIR, is our subdirectory and was assigned an attribute of 10H (offset 18BH). In theory, a subdirectory can be read like any other file. However, the designers of DOS elected to store zero as the subdirectory's file size (offsets 19C-19FH). As a result, DOS assumes the file to be of zero length and refuses to read it.

If you are the curious type, you probably want to see what's stored inside \SUBDIR. I can think of two approaches to directly access the contents of \SUBDIR. The conventional method determines the subdirectory's entry point into the FAT, chains through its clusters until the last one, and then performs a disk read. The unconventional method offers an element of danger. The subdirectory is converted into and read like a normal file. I have opted to demonstrate the unconventional method. Use the E command to modify attribute and file size in the entry field of \SUBDIR:

```
-E18B
DS:018B  10.20
-E19D
DS:019D  00.04
```

The size of the subdirectory is now 0400H (1,024 bytes) or two sectors, the smallest file permitted by the 360K format. The two sectors can hold a maximum of 32 entries, more than enough for \SUBDIR at this time. As needed, the size of the subdirectory file will grow. To finalize these changes, write the directory sectors to the disk. Warning! You must invoke the W command with the identical parameters previously entered with the L command:

```
-W100 0 5 7
-Q
A>DIR
```

There it is! The file SUBDIR appears in the root directory as an ordinary file with a size of 1,024 bytes. If you try to look at this file with the DOS TYPE command, gibberish appears on the screen because \SUBDIR is not an ASCII file. Return to DEBUG with the loaded file and dump offsets 100-17FH:

```
A>DEBUG SUBDIR
-D100
```

The first 128 bytes of \SUBDIR are shown in Figure 2. The first two entries of any subdirectory begin with the two special 32-byte entries (single-period and double-period). These two special entries are established when the subdirectory is created, and they cannot be deleted. The single-period entry contains the cluster field entry in the FAT for \SUBDIR, while the double-period entry stores the cluster field in the FAT of the subdirectory's parent directory. Both entries are necessary since subdirectories may be nested to any number of levels.

The data file stored in \SUBDIR (HANDLE.TXT) possesses the typical 32-byte entry field. \SUBDIR does not point to other subdirectories, but if it did, the nested subdirectory would be

```
DS:0100  49 42 4D 42 49 4F 20 20-43 4F 4D 27 00 00 00 00   IBMBIO  COM'....
DS:0110  00 00 00 00 00 00 00 60-54 07 02 00 80 12 00 00   .......T........
DS:0120  49 42 4D 44 4F 53 20 20-43 4F 4D 27 00 00 00 00   IBMDOS  COM'....
DS:0130  00 00 00 00 00 00 00 60-54 07 07 00 80 42 00 00   .......T....B..
DS:0140  43 4F 4D 4D 41 4E 44 20-43 4F 4D 20 00 00 00 00   COMMAND COM ....
DS:0150  00 00 00 00 00 00 00 60-54 07 18 00 80 45 00 00   .......T....E..
DS:0160  44 45 42 55 47 20 20 20-43 4F 4D 20 00 00 00 00   DEBUG   COM ....
DS:0170  00 00 00 00 00 00 00 60-54 07 2A 00 80 2E 00 00   .......T.*.....
DS:0180  53 55 42 44 49 52 20 20-20 20 20 10 00 00 00 00   SUBDIR      .....
DS:0190  00 00 00 00 00 00 F4 18-64 10 36 00 00 00 00 00   ......t.d.6.....
```

Figure 1. First five entries of root directory

```
DS:0100  2E 20 20 20 20 20 20 20-20 20 20 10 00 00 00 00   .           .....
DS:0110  00 00 00 00 00 00 71 D3-64 10 36 00 00 00 00 00   ......qSd.6.....
DS:0120  2E 2E 20 20 20 20 20 20-20 20 20 10 00 00 00 00   ..          .....
DS:0130  00 00 00 00 00 00 71 D3-64 10 00 00 00 00 00 00   ......qSd.......
DS:0140  48 41 4E 44 4C 45 20 20-54 58 54 20 00 00 00 00   HANDLE  TXT ....
DS:0150  00 00 00 00 00 00 2C 16-64 10 37 00 16 00 00 00   ......,.d.7.....
DS:0160  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
DS:0170  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
```

Figure 2. First 128 bytes of \SUBDIR

```
;DISPLAYS FILE NAMES OF SPECIFIED DIR
CSEG    SEGMENT
        ASSUME  CS:CSEG
        ORG     100H
START:  MOV     AH,9            ;DISPLAY MESSAGE
        MOV     DX,OFFSET MESS
        INT     21H
        MOV     AH,0AH          ;INPUT ASCIIZ STRING
        MOV     DX,OFFSET STRING
        INT     21H
;CRLF
        MOV     AH,9
        MOV     DX,OFFSET CRLF
        INT     21H
;ESTABLISH NULL STRING
        MOV     BX,OFFSET STRING+1  ;POINT TO STRING SIZE
        MOV     AL,[BX]         ;GET SIZE
        MOV     AH,0            ;INITIALIZE AX
        ADD     BX,AX           ;POINT TO STRING END
        INC     BX              ;POINT TO CR
        MOV     AL,0
        MOV     [BX],AL         ;STORE NULL
;SET DTA
        MOV     AH,1AH
        MOV     DX,OFFSET DTA
        INT     21H
        MOV     AH,4EH      ;SEARCH FIRST MATCH
        MOV     CX,20H      ;NORMAL FILE
NEXT:   MOV     DX,OFFSET STRING+2   ;ASCIIZ STRING
        INT     21H
        JC      NOMATCH
        MOV     AH,9        ;DISPLAY FILE NAME
        MOV     DX,OFFSET DTA+30
        INT     21H
;BLANK DTA
        MOV     BX,OFFSET DTA+30
        MOV     CX,13
        MOV     AL,' '
BLANK:  MOV     [BX],AL
        INC     BX
        LOOP    BLANK
;SEARCH NEXT MATCH
        MOV     AH,4FH
        JMP     NEXT
NOMATCH: INT    20H
MESS    DB      0AH,0DH,'ENTER ASCIIZ STRING:',0AH,0DH,24H
STRING  DB      65,65 DUP(0)
CRLF    DB      0DH,0AH,0AH,24H
DTA     DB      43 DUP(' '),0AH,0DH,24H
CSEG    ENDS
        END     START
```

Figure 3. Assembler code for SEARCH.COM

assigned an attribute and file size of 10H and zero, respectively. Now that you have had the privilege to see what's inside a subdirectory, restore \SUBDIR to its original status:

```
-L100 0 5 7
-E18B
DS:018B  20.10
-E19D
DS:019D  04.00
-W100 0 5 7
```

I recommend that you do not attempt the unconventional approach to view subdirectory files maintained within a fixed-disk. If you make an error in writing to the directory sectors, data stored on the disk may be rendered useless. As you will see in the next section, extended DOS functions (4EH/4FH) provide a safer and easier method to access information contained inside subdirectory files.

Searching Directories

Extended DOS function 4EH searches the specified directory for the first matching file, while function 4FH continues the file search that was begun by function 4EH. A short program called SEARCH, a simplified version of the DIR command, demonstrates how to use these functions to display the file names stored in any directory.

The assembly code for SEARCH.COM is listed in Figure 3. The program displays a message and waits for you to enter the file specification. After the string is entered, the program changes it to an ASCIIZ format by replacing the carriage return (0DH) with the null byte (00H). Remember to include at least one global file name character (* or ?). Otherwise, only the specified file name can be matched. Some file specification examples are shown in Figure 4.

The search functions assume that you have previously used 1AH to declare a disk transfer area (DTA). Whenever a match is found, the 43 byte DTA is filled with information regarding the file's name, date, time, size, and attribute (see Table 2). The last 13 locations of the DTA (bytes 30- 42) store the file's name/extension in the form of an ASCIIZ string. It is this portion of the DTA that is displayed after every match. An unsuccessful search (CY) terminates the program.

Introducing XFILE

Extended DOS function 41H allows you to erase files from any directory. To delete \SUBDIR\HANDLE.TXT from the disk, type the commands shown in Figure 5.

The operation fails only if an element of the path name does not exist, or the designated file has the read-only attribute.

The assembly code for XFILE.EXE, the utility capable of deleting files from any directory, is listed in Figure 6. The program is organized to:

1. Position the cursor in the upper left corner.
2. Save the original caps state and set the caps-lock key.
3. Wait for the file specification to be entered and then convert it to an ASCIIZ string (FSTRING).
4. Move drive/path from FSTRING to DSTRING.
5. Set up the DTA, search for first match, and then move the DTA's ASCIIZ string to DSTRING.
6. Display file name on screen and offer the option to delete file and search next match <Y>, search next match <N>, or quit program <Q>.
7. Restore original caps state before returning to DOS.

You must enter the drive/path/file as stipulated in SEARCH. Remember to include backslashes because XFILE moves all memory locations to the left of the last backslash in FSTRING to DSTRING.

Let's compare XFILE to DATACIDE. Whereas DATACIDE relies on FCB functions and can access files only from the root directory, XFILE utilizes extended DOS services and supports tree file structures. DATACIDE loads the disk's entire directory into memory and searches for file names on its own. A hard-disk contains 32 directory sectors and DATACIDE needs to set aside 16,384 bytes to hold this data. On the other hand, XFILE depends on DOS to search directories for specified file names. As a result, not only is XFILE more powerful, it is considerably smaller than DATACIDE.

Bytes	Represents
0-20	Reserved by DOS
21	Attribute of matched file
22-23	Time
24-25	Date
26-29	size
30-42	File name/extension (ASCIIZ)

Table 2. Information returned in the DTA by functions 4EH/4FH

```
A:\X*.*          displays files beginning with X from drive A

B:\SUB1\*.COM      displays .COM files from B:\SUB1

C:\SUB1\SUB2\*.*   displays all files from C:\SUB1\SUB2
```

Figure 4.

```
-A100
DS:0100  MOV  AH,41     ;DELETE FILE
DS:0102  MOV  DX,0109   ;POINT TO FILE SPECIFICATION
DS:0105  INT  21
DS:0107  INT  20
DS:0109  DB   'A:\SUBDIR\HANDLE.TXT',0
<ENTER>  2X
-G
```

Figure 5.

Figure 6.

```
;*************************************************************************
SSEG    SEGMENT  STACK
        DB       20 DUP  ('STACK   ')
SSEG    ENDS
;*************************************************************************
DSEG    SEGMENT
FSTRING DB   64,65 DUP (0)       ;ASCIIZ FILE STRING
DSTRING DB   64,65 DUP (0)       ;ASCIIZ DELETE STRING
DTA     DB   43 DUP (0)          ;DATA TRANSFER AREA
FENTRY  DW   ?                   ;FILE ENTRY IN DSTRING
CPS     DB   ?                   ;ORIGINAL CAPS-LOCK STATE
MESS1   DB   0AH,0AH,0DH,'ENTER FILE SPECIFICATION:',0AH,0DH
        DB   '(EXAMPLES:  A:\*.* , B:\PATH\*.COM , C:\PATH1\PATH2\DATA.*)'
        DB   0DH,0AH,0AH,24H
MESS2   DB   0DH,0AH,'DELETE FILE?   <Y>YES   <N>NO   <Q>QUIT     $'
MESS3   DB   0DH,0AH,'FILE REMOVED FROM DIRECTORY  $'
MESS4   DB   0DH,0AH,'FILE COULD NOT BE DELETED $'
MESS5   DB   0DH,0AH,0AH,'NO FILES LOCATED',0DH,0AH,0AH,24H
CRLF    DB   0DH,0AH,0AH,24H
DSEG    ENDS
;*************************************************************************
CSEG    SEGMENT
MAIN    PROC   FAR
        ASSUME CS:CSEG,DS:DSEG,ES:DSEG,SS:SSEG

START:
;SET RET AND DS/ES REGISTERS
        PUSH DS
        SUB  AX,AX
        PUSH AX
        MOV  AX,DSEG
        MOV  DS,AX
        MOV  ES,AX

        CALL CURSOR      ;POSITION CURSOR
        CALL CAPS        ;SET CAPS-LOCK KEY
        CALL STRINGS     ;INPUT & SET UP ASCIIZ STRINGS
        CALL SEARCH      ;SEARCH DIR & OFFER DELETE OPTION
        CALL CAPS2       ;RESTORE ORIGINAL CAPS STATE
        RET
MAIN    ENDP
;-------------------------------------------------------------------------
;POSITION CURSOR IN UPPER LEFT CORNER
CURSOR  PROC NEAR
        MOV  CX,25
CLEAR:  MOV  DL,0AH      ;CLEAR SCREEN
        MOV  AH,2
        INT  21H
        LOOP CLEAR
        MOV  AH,2        ;POSITION CURSOR
        MOV  BH,0
        MOV  DX,0
        INT  10H
        RET
CURSOR  ENDP
;-------------------------------------------------
;SET CAPS LOCK
CAPS    PROC NEAR
        PUSH DS
        MOV  AX,0
        MOV  DS,AX
        MOV  BX,0417H    ;POINT TO CAPS-LOCK STATE
        MOV  AL,[BX]
        POP  DS
        MOV  CPS,AL      ;SAVE ORIGINAL CAPS STATE
        PUSH DS
        MOV  DX,0
        MOV  DS,DX
        OR   AL,40H      ;SET CAPS-LOCK BIT
        MOV  [BX],AL
        POP  DS
        RET
CAPS    ENDP
;-------------------------------------------------
;RESTORES ORIGINAL CAPS-LOCK STATE
CAPS2   PROC NEAR
        MOV  AL,CPS      ;GET ORIGINAL CAPS-LOCK STATE
        PUSH DS
        MOV  DX,0
        MOV  DS,DX
        MOV  BX,0417H
        MOV  [BX],AL
        POP  DS
        RET
CAPS2   ENDP
;-------------------------------------------------
```

```
;SET UP ASCIIZ STRINGS
STRINGS PROC NEAR
;DISPLAY MESSAGE
        MOV  AH,9
        MOV  DX,OFFSET MESS1
        INT  21H
;INPUT FILE STRING
        MOV  AH,0AH
        MOV  DX,OFFSET FSTRING   ;INPUT FILE STRING
        INT  21H
;STORE NULL AT END OF FSTRING
        MOV  BX,OFFSET FSTRING+1  ;POINT TO FSTRING SIZE
        MOV  AL,[BX]     ;GET SIZE
        MOV  AH,0
        ADD  BX,AX       ;POINT TO END OF FSTRING
        INC  BX          ;POINT TO 0DH
        MOV  AL,0
        MOV  [BX],AL     ;STORE NULL
;SET DSTRING FOR DELETION
        MOV  SI,OFFSET FSTRING+2  ;POINT TO PATH OF FSTRING
        MOV  DI,OFFSET DSTRING    ;POINT TO STRING FOR DELETION
SLASH:  DEC  BX          ;POINT TO END OF FSTRING
        MOV  AL,[BX]     ;GET FSTRING CHAR.
        CMP  AL,''\''      ;LOOK FOR LAST \
        JZ   FOUND       ;JUMP IF \ FOUND
        JMP  SLASH       ;KEEP LOOKING FOR \
FOUND:  INC  BX
        MOV  CX,BX
        SUB  CX,SI       ;SET COUNT
        CLD              ;CLEAR DIRECTIONAL FLAF
        REP  MOVSB       ;MOVE PATH FROM FSTRING TO DSTRING
        MOV  FENTRY,DI   ;SAVE FILE ENTRY IN DSTRING
        RET
STRINGS ENDP
;-----------------------------------------------------------
;SEARCH DIR FOR MATCH & OFFER OPTION TO DELETE FILE
SEARCH  PROC NEAR
        MOV  AH,1AH      ;SET UP DTA
        MOV  DX,OFFSET DTA
        INT  21H
        MOV  AH,4EH      ;SEARCH FIRST MATCH
        MOV  CX,20H      ;NORMAL FILE
        MOV  DX,OFFSET FSTRING+2  ;POINT TO ASCIIZ STRING
        INT  21H
        JC   NOMATCH     ;JUMP IF NO MATCH FOUND
        JMP  MATCH
NEXT:   MOV  AH,4FH      ;SEARCH NEXT MATCH
        MOV  CX,20H
        MOV  DX,OFFSET FSTRING+2
        INT  21H
        JC   QUIT
;BLANK OUT PREVIOUS ENTRY IN DSTRING
MATCH:  MOV  BX,FENTRY   ;GET FILE ENTRY IN DSTRING
        MOV  CX,13
        MOV  AL,'' ''
BLANK:  MOV  [BX],AL
        INC  BX
        LOOP BLANK
;MOVE FILE NAME TO DSTRING
        MOV  DI,FENTRY   ;GET FILE ENTRY
        MOV  SI,OFFSET DTA+30    ;POINT TO FILE NAME IN DTA
        MOV  CX,13          ;13 CHARAC. ASCIIZ STRING
        CLD                 ;CLEAR DIRECTIONAL FLAG
        REP  MOVSB          ;MOVE ASCIIZ FROM DTA TO DSTRING
        MOV  AH,9           ;SKIP LINE
        MOV  DX,OFFSET CRLF
        INT  21H
;DISPLAY FILE NAME
        MOV  BX,OFFSET DTA+30  ;POINT TO ASCIIZ IN DTA
SCREEN: MOV  DL,[BX]
        CMP  DL,0        ;END OF ASCIIZ STRING?
        JE   OPTION
        MOV  AH,2        ;DISPLAY FILE NAME CHARAC.
        INT  21H
        INC  BX
        JMP  SCREEN
;OPTION TO DELETE
OPTION: MOV  AH,9        ;DISPLAY OPTION MESSAGE
        MOV  DX,OFFSET MESS2
        INT  21H
        MOV  AH,1
        INT  21H
        CMP  AL,''Y''
        JZ   DEL         ;IF ''Y'', DELETE FILE
        CMP  AL,''N''
        JZ   NEXT        ;IF ''N'', NEXT FILE
        CMP  AL,''Q''
        JZ   QUIT        ;IF ''Q'', QUIT PROGRAM
        JMP  OPTION      ;REPEAT OPTION
DEL:    MOV  AH,41H      ;DELETE FILE FUNCTION
        MOV  DX,OFFSET DSTRING
        INT  21H
        JC   FAILED
        MOV  DX,OFFSET MESS3
        MOV  AH,9
        INT  21H
        JMP  NEXT        ;DISPLAY NEXT FILE NAME
FAILED: MOV  AH,9        ;OPERATION FAILED
        MOV  DX,OFFSET MESS4
        INT  21H
        JMP  NEXT        ;DISPLAY NEXT FILE NAME
NOMATCH:MOV  AH,9        ;NO FILES LOCATED
        MOV  DX,OFFSET MESS5
        INT  21H
QUIT:   RET
SEARCH  ENDP
;-----------------------------------------------------------
CSEG    ENDS
;***********************************************************
        END   START
```

Figure 6. Assembler code for XFILE.EXE ■

If You Don't Contribute Anything....

....Then Don't Expect Anything

TCJ is User Supported

The ZCPR3 Corner

by Jay Sage

For my column this time I plan to cover two subjects, both of which I have dealt with somewhat at length in the past. Nevertheless, there just seems to be a lot more to say on these subjects. The first is ARUNZ; the second is shells in general, and the way WordStar® 4 behaves (or rather misbehaves) in particular.

I was quite surprised and pleased by the enthusiastic response to my detailed treatment of ARUNZ in issue 31. Apparently, there were many, many people who were unaware of what ARUNZ was and who are now quite eager to put it to use. There are two specific reasons for taking up the subject of ARUNZ here again so soon.

First of all, I think that readers will benefit from a discussion of some additional concrete examples. Since my own uses are the ones I know best, I plan to take the ALIAS.CMD file from my own system as an example and discuss a number of interesting scripts. My first cut at doing that for this column came out much too long, so I will cover half of the file this time. The other half will be covered in the next column.

The second reason is that I have just gone through a major upgrade to ARUNZ. It is now at version 0.9J. Several aspects of its operation as described in my previous column have been changed, and quite a few new parameters have been added.

The changes in ARUNZ were stimulated by two factors. One is the two new dynamic Z Systems that will have been released by the time you read this: NZCOM for Z80 computers running CP/M 2.2 and Z3PLUS for Z80 computers running CP/M-Plus. These two products represent a tremendous advance in the concept of an operating system, and everyone interested in experimenting with or using Z System—even if he already has a manually installed ZCPR3 running now—should get the one that is appropriate to his computer.

With these new Z System implementations, if your level of computer skill is high enough to run a wordprocessor or menu program, then you can have a Z System designed to your specifications in a matter of minutes. You can change the design of your Z System at any time, even between applications. As described later, ARUNZ now has some parameters to return addresses of system components so that aliases can work properly even when those system components move around, as they may do under these dynamic systems.

My New Computer System

The second impetus came from my finally building for myself a state-of-the-art computer! For most of my work in the past I have used a BigBoard I with four 8" floppy disk drives and an SB180 with four 5" disk drives. Neither machine had a hard disk.

The SB180, my main system for the past year and a half, had been sitting on the floor in the study. The printed circuit board was mounted in a makeshift chassis with two power supplies, just as I got it from someone who bought it at the Software Arts liquidation auction and after they had stripped out the disk drives (at $25 I could hardly complain!). I added my own drives, which sat in the open air (for cooling among other reasons) in two separate drive cabinets elsewhere on the floor. All in all not very pretty and not as functional as it could have been.

The sad part of it is that during all this time I had everything needed to turn the SB180 into an enjoyable and productive system. A high-speed 35 Mb hard disk was collecting dust on a shelf; an attractive surplus Televideo PC-clone chassis adorned the work bench in the basement; the XBIOS software disks sat ignored in one of my many diskette boxes.

Finally one weekend I decided that it would be more efficient in the long run to take some time off from my programming and writing work to reconstruct the system. Indeed, it has been! The SB180 is now attractively mounted in the Televideo chassis with one 96-tpi floppy and one 48-tpi floppy. The hard disk is configured as four 8 Mb partitions and runs very nicely with the fast Adaptec 4000 controller.

With the hardware upgraded, I then did the same to the software. Installing XBIOS on the SB180 took so little time that I really had to kick myself for not doing it sooner. Richard Jacobson was quite right in his description of it in issue 31. Thank you, Malcom Kemp, for a really nice product.

Once I was fixing things up, I decided I should really do it up right, so I also purchased the ETS180IO+ board from Ken Taschner of Electronic Technical Services—this despite the fact that a Micromint COMM180 board was also a part of my longstanding inventory of unused equipment. I cannot compare the ETS board to the COMM180, never having used the latter, but I certainly am highly pleased with it. XBIOS includes complete support for the ETS board, so configuring the system to make use of the extra ETS180IO+ features, like the additional parallel and serial ports and the battery-backed clock, was very easy.

I have been so pleased with the new system that I even went out and bought a real computer table for it to sit on. For the past years, the terminal's CRT unit had been sitting on one of those flimsy folding dining-room utility tables, with a yellow-pages phone book under it to jack it up to the right height. The keyboard sat on a second folding table, and the whole thing was always in imminent danger of toppling over. What a pleasure it is to sit at the new system.

While I'm waxing enthusiastic, let me mention one other thing I did to reduce the disarray in the study. I bought four Wilson-Jones media drawers to house my vast collection of floppies. These diskette cabinets resemble professional letter filing cabinets. A drawer, which can hold more than 100 floppies, pulls out on a full suspension track so that one can easily reach all the way to the back. Since there is no top to flip open, units can be stacked on top of each other to save a great deal of table space. Clips are provided to secure the units to their neighbors both horizontally and vertically.

The only drawback to these disk drawers has been their cost. Inmac and the other major commercial supply houses want more than $60 each! But Lyben, which sends its catalogs out to many computer hobbyists, offers them for only $35. Extra dividers, which I recommend, are just under $6 per package of five. Lyben can be reached at 313-589-3440 (Michigan). [Note added at last

moment—I am sorry to say that I just received the new Lyben catalog, and the price has now gone up to $45. Although this is still a bargain when compared to other vendors' prices, I'm glad I put in my order when I did.]

ARUNZ VERSION 0.9J

Now that I have had my chance to show my excitement over the new state of my computer and computer room, let's get on with the discussion of ARUNZ. First we will discuss the changes introduced since version 0.9G, both the old features that have changed and the new features that have been introduced.

Changes in Old Features

Because, as noted in my last column, ZCPR34 can pass commands containing explicit file types or wildcard characters ('?' and '*'), the characters used to define special matching conditions in the alias names in ALIAS.CMD had to be changed. The period, which had been used to indicate the beginning of optional characters in the alias name, has been replaced by the comma. The question mark had been used to indicate a wild-character match in the alias name. Since it can now be an actual character to be matched, the underscore has replaced it.

Since the command verb can now include an explicit file type (not necessarily COM) and/or a directory prefix, several changes have been made to the parameters that parse the command verb. In general, all of the command line tokens are now treated in the same way; all four token parsing parameters ('D', 'U', ':', and '.') now work with digits from 0 to 9 and not just 1 to 9. Thus the command line

```
C3:TEST>arunz b12:test.z80 commandtail
```

or, with ARUNZ running as the extended command processor (ECP), the command

```
C3:TEST>b12:test.z80 commandtail
```

will have the following parameter values for token 0, the command verb:

```
$D0     B
$U0     12
$:0     TEST
$.0     Z80
```

This is s significant change. please take careful note of it. The parameters $D0 and $U0 no longer necessarily return the logged in drive and user. For the standard configuration of ZCPR33 (and 34) a verb of the form B12:TEST cannot be passed to the extended command processor; the presence of an explicit directory specification results in the immediate invocation of the error handler (skipping the ECP) if the file cannot be found in the specified directory. However, if a file type is included, the 'bad' command will be passed to the ECP.

New Features

There are now three new parameters that do return information about the directory that was current (logged in) when ARUNZ was invoked. These parameters are shown below with their meaning and the values they would have with the example command above:

parameter	meaning	value
$HD	Home Drive (D)	C
$HU	Home User (U)	3
$	Home Both (i.e., DU)	C3

There is also a new parameter to denote the entire command line, including both the command verb and the command tail. Many people in the past confused "command line" with "command tail" and attempted to use the parameter $* for the former. The new parameter is '$!'. It is roughly equivalent to '$0 $*', but there is one important difference. The latter parameter expression always includes a space after the command verb, even if there was no tail ('$*' was null). This space caused problems with some commands. For example, when the SLR assembler SLR180 is invoked with nothing after it, it enters interactive mode and allows the user to enter a series of assembly requests. Unfortunately, the code is not smart enough to distinguish a completely nonexistent command tail from one with only spaces. When the command "SLR180_" is entered, where '_' represents a blank space, the assembler looks for a source file with a null name. Not finding it, it returns with an error message.

I used to deal with this problem by writing a complex alias of the form:

```
SLR180   if nu $*;asm:slr180;else;asm:slr180 $*;fi
```

With the new parameter, all this complication can be avoided. The script is simply:

```
SLR180     asm:$!
```

If you are wondering why one would want an alias like this, just wait a while. It will be explained later.

There is also a whole set of new parameters that generate the addresses of almost all of the Z System modules. This capability will become important with the dynamic Z Systems now being introduced (NZCOM and Z3PLUS). With those systems, the addresses of the RCP, FCP, CCP, and so on can all change during system operation. The new parameters permit one to make reference to the addresses of those modules even when they move around. My ALIAS.CMD file described below will have some examples of how these parameters are used.

These parameters begin with $A ('A' for address) and are followed by an additional letter as follows:

B	BIOS	L	MCL (command Line)
C	CCP	M	MSG (message buffer)
D	DOS	N	NDR
E	ENV	P	PATH
F	FCP	R	RCP
I	IOP	S	STK (shell stack)
		X	XFCB (external FCB)

Amazingly enough, these names are all mnemonic except for the conflict over 'M' between the multiple command line buffer (MCL) and message buffer (MSG). I resolved this by using 'L' (think of LINE) for the MCL.

Finally, there is a new symbol that can be used to make a special kind of alias name specification in ALIAS.CMD. If a name element begins with a '>', then only the file type of the command verb is used in the comparison. Without this feature one had to use very complex forms to recognize a file type. For example, suppose you want to be able to enter the name of a library file as LBRNAME.LBR as a command and have VLU invoked on it. The following script used to be required:

```
?.LBR=??.LBR=???.LBR=????.LBR=?????.LBR=??????.LBR=
   ???????.LBR=????????.LBR   vlu $0
```

Every possible number of characters in the library name had to be dealt with explicitly. With the new symbol and the other ARUNZ09J features, one can define this script more simply as follows:

```
>LBR    vlu $:0
```

Example ALIAS.CMD File

Now that we have described the new resources available in ARUNZ09J, we will begin our look at part of the ALIAS.CMD file that I am using right now on the SB180. It will be the second half of the file, because that part contains some items of immediate relevance.

First some words of philosophy. There are many ways in which Z System can be used effectively, and I am always amazed and impressed at the different styles developed by different users. What I will now describe is my approach. As they say, yours may differ! In any case, I hope these comments will stimulate some good ideas, and, as always, I eagerly await your comments and suggestions.

I am a strong believer in short search paths. When I make a mistake in typing a command, I do not want to have to twiddle my thumbs while the command processor thrashes through a lot of directories searching for the nonexistent command. I want the error handler to take care of it as quickly as possible. As a result, the search path on my SB180 includes only one directory, A0, the RAM disk. (With XBIOS, the RAM disk can be mapped to the A drive.)

When I enter a command, it is searched for only in A0. If it is not found there, then ARUNZ (renamed to CMDRUN.COM) is loaded from A0, and it looks for a script in ALIAS.CMD, also in A0. If ARUNZ cannot resolve the command, then the error handler, EASE in my case, is invoked (you guessed it, also on A0). Thus no directory other than the RAM disk is accessed except by an explicit directory reference generated either by an alias script or by a manually entered command. Everything appears to operate instantaneously.

Aliases to Provide Explicit Directory Prefixes

Obviously, I cannot keep all the COM files that I use in directory A0. In fact, with the tiny RAM disk on the SB180 (and allowing about 100K for a BGii swap file), there is barely enough room for CMDRUN.COM (ARUNZ), ALIAS.CMD, EASE.COM, EASE.VAR, IF.COM, ZF.COM (ZFILER), ZFILER.CMD, SAVSTAMP.COM, ZEX.COM, ZEX.RSX, and a few directory programs. Fortunately, this is all that really needs to be there.

So what do I do about all the other COM files that I want to use? There are two possibilities. I could invoke them manually with explicit directory references, as in "B0:CRC FILESPEC", but this would clearly be a nuisance (and contrary to the spirit of Z System!). The other alternative is to provide alias definitions in ALIAS.CMD for all the commands in other directories that I want to use.

A second half of my ALIAS.CMD file is shown in Listing 1. The group of aliases at the very end comprises several sets of definitions that do just what I have described for several of the directories on the hard disk. As I use programs in other directories, I add them to the ALIAS.CMD file.

These aliases are included at the end, by the way, so that other definitions can preempt them as desired. If you look carefully, you will see some aliases defined here that are also defined earlier in the ALIAS.CMD file. The earliest definition always takes precedence, because ARUNZ scans ALIAS.CMD from the beginning and stops as soon as it encounters a matching name specification.

Directory B0, named SYS, contains most of my system utilities. Directory B1, named ASM, contains my assembly language utilities. A few commonly used files are in other directories. The aliases defined in these sections do nothing more than add an explicit directory prefix to the command entered. For example, the script definition

```
AFIND    b0:$!
```

would take my command line "AFIND TAIL..." and turn it into "B0:AFIND TAIL...". Note how compact the definitions can be. You do not need a separate line for each command. Similar scripts could be constructed, by the way, for COM files kept in COMMAND.LBR and extracted and executed by LX. I do not use LX, so I have no examples to show.

There are several fairly easy ways to automate the construction of these entries in the ALIAS.CMD file. If you use PMATE or VEDIT as your text editor, you can write macros that will perform the entire process. That is how I generated the aliases you see. With the PMATE macro, I can easily repeat the process from time to time to make sure that all my COM files are represented by aliases. So far I have run my PMATE macro on user areas 0, 1, 2, 3, and 4 of hard disk partition B.

Lacking these tools, you can run "SD *.COM /FX" to get a file DISK.DIR containing a horizontally sorted listing of all the COM files in a directory (without going to a lot of trouble, I do not get a sorted listing from PMATE). Then use your favorite editor, whatever it is, to add carriage returns so that each file is on its own line and to delete all of the text after the file name (i.e., the dot, file type, and file size). If there are any commands for which you want to have special aliases (we'll see some examples shortly), you may delete their names from the list (or you can leave them—they do no harm). Then close up the list, inserting equal signs and, when the line is wide enough, add the command script. Finally, merge this with the rest of your ALIAS.CMD file.

Aliases for Special Command Redefinitions

Just before the simple redefinition aliases there are six commands that have been separated out for special treatment. Consider the first of them:

```
ZP,ATCH      b0:zpatch $*
```

I find that my fingers have some difficulty typing the full ZPATCH correctly, and this alias permits me to enter simply ZP. Note that in this case we cannot use "b0:$!" for the script because the alias name allows for forms other than an exact ZPATCH. If the script used the $! parameter and the command was entered as ZP, then the expanded script would become "B0:ZP ...", which would not work.

The alias for crunching is similar in some respects but more elaborate. The letter combination CH must give me trouble, because I often type CRUNCH wrong, too, unless I work very carefully. This alias not only lets me use the short form CR; it also allows the command to work with named directories.

```
CR,UNCH      b0:crunch $d1$u1:$:1.$.1 $d2$u2:
```

By expanding the first and second parameters explicitly, named directory references can be converted to the DU: form that CRUNCH can deal with.

The alias for DATSWEEP goes a little further than the other two insofar as alternative forms are concerned.

```
DATSW,EEP=DS=SWEEP      b0:datsweep $*
```

It allows abbreviated forms as short as DATSW, but it additionally allows alternative nicknames for the command, such as DS or the more familiar SWEEP, which it replaces on my system.

The next example in this section shows how a program that does not know about Z System file specifications at all can be made to work with them anyway.

```
LDIR         $d1$u1:;b0:ldir $:1;$hb;
```

For LDIR I just started to use LDIR-B, which displays date stamp information about files in the library. Unfortunately, it does not know about named directories; in fact, it does not even know

anything about user numbers. If he is true to form, Bruce Morgen, the Intrepid Patcher, will soon have a ZLDIR-B or an LDRZ-B that will accept full Z System file specs, and I will be able to retire this alias.

At present, however, LDIR-B accepts only the standard CP/M syntax for files. As a result, it is not enough simply to pick apart the token, as it would be if LDIR would accept the form DU:NAME.TYP. Instead, the directory specified for the library is logged into, then the LDIR command is run on the library name, and finally the original directory is relogged. This will work very nicely unless the user number specified is higher than 15 (and your Z33/Z34 is not configured for logging into high user numbers).

The last two examples in this series illustrate still another way to make aliases lighten the typing burden. With XBIOS, alternative versions of the operating system are described in model files. These typically have a file type of MDL, but that type is not required or the default. Consequently, the SYSBLD system-defining utility and the XBOOT system-loading utility must be given an explicit file type. Since I always use MDL for the type, I created these aliases to add the file type for me so that I can enter the commands simply as "SYSBLD TEST" or "XBOOT BIGSYS".

```
SYSBLD          b0:;b0:$0 $1.mdl;$hb:
BOOT=XBOOT      b0:;b0:xbtot $1.mdl
```

The XBOOT alias lets me save a little typing by omitting the leading 'X' if I wish. The SYSBLD alias returns to the original directory when it is finished. Since XBOOT coldboots a new operating system, any trailing commands are lost anyway. The XBOOT command will soon support a warmboot mode, in which, like NZCOM and Z3PLUS, the new system is created without affecting the multiple command line, shell stack, or other loaded system modules that have not changed their address or size. I might then add an alias REBOOT or WBOOT (warmboot) that will load a new system and return to the original directory.

Memory Display Aliases

In my system development work I often have occasion to examine various parts of memory. I might want to look at the beginning of the BIOS to check the hooks into an RSX (resident system extension), or I might want to see the contents of the ZCPR3 message buffer to see how some flags are being used.

I used to have a set of aliases like these with explicit addresses in the script ("P FE00" to look at the ENV, for example). This relieved my mind of the task of remembering the addresses where these modules were located in memory. With the new dynamic systems, even a good memory will not suffice, since the modules can move around, and one can not easily be sure just where they are at any given time.

By using the new parameters that I described earlier, the scripts always have the correct addresses. [Actually, they can still be fooled if these parameters are used in multiple-command-line scripts that include the loading of a new dynamic system. As I warned in my earlier article on ARUNZ, all parameters are expanded at the time the alias is invoked. If the system is changed after that, the parameter values may no longer be correct when that part of the script actually runs.]

Shells and WordStar Release 4

As I noted in an earlier column, WordStar Release 4 was a very exciting event for the CP/M world in general and the Z-System world in particular. It was the first major commercial program to recognize Z System and to make use of its features. Unfortunately, the Z System code in WS4 was not adequately tested, and many errors, some quite serious, slipped through. Some of the most significant errors concern WS4's operation as a ZCPR3 shell.

Let's begin with a little background on the concept of a shell in ZCPR. Normally, during Z System operation the user is prompted for command line input. This input may consist of a string of commands separated by semicolons. When the entire sequence of commands has been completed and the command line buffer is again empty, the user would be prompted again for input.

This prompting is performed by the ZCPR command processor, which, because it is limited in size to 2K, is correspondingly limited in its power. Richard Conn, creator of ZCPR, had the brilliant idea of including a facility in ZCPR3 for, in effect, replacing—or, perhaps better said, augmenting—the command processor as a source of commands for the system. This is the shell facility.

Under ZCPR3, when the command processor finds that there are no more commands in the command line buffer for it to perform, before it prompts the user for input, it first checks a memory buffer called the shell stack. If it finds a command line there, it executes that command immediately, without prompting the user for input. The program run in that way is called a shell, because it is like a shell around the command processor kernel. The shell is what the user sees instead of the command processor, and the shell will normally get commands from the user and pass them to the command processor. In effect, the outward appearance of the operating system can be changed completely when a shell is selected.

A perfect example of a shell is the EASE history shell. To the user it looks rather like the command processor. But there are two very important differences. First of all, the command line editing facilities are greatly augmented. One can move the cursor left or right by characters, words, or commands; one can insert new characters or enter new characters on top of existing characters; characters or words can be deleted. One has, in a way, a wordprocessor at one's disposal in creating the command line.

The second feature is the ability to record and recall commands in a history file. Many users find that they execute the same or similar commands repeatedly. The history feature of EASE makes this very convenient. These two command generation features require far too much code to include in the command processor itself, so it is very convenient to have the shell capability.

Programs designed to run as shells have to include special code to distinguish when they have been invoked by the user and when they have been invoked by the command processor. ZCPR3 makes this information available to such programs. When invoked by the user, they simply write the appropriate command line into the shell stack so that the next time the command processor is ready for new input, the shell will be called on. After that, the user sees only the shell. Shells normally have a command that the user can enter to turn the shell off.

ZCPR3 goes beyond having just a single shell; it has a stack of shells. A typical configuration allows four shell commands in the stack. When the user invokes a command designed to run as a shell, it pushes its name onto the stack. When the user cancels that shell, any shell that had been running previously comes back into force. Only when the last shell command has been cancelled (popped from the shell stack) does the user see the command processor again.

Let's look at some of the shells that are available under Z System. We have already mentioned the EASE history shell. There is also the HSH history shell, which offers similar capabilities. It was written in C and cannot be updated to take advantage of innovations like type-3 and type-4 commands. I would say that EASE is the history shell of choice today. This is especially true because EASE can do double service as an error handler as well, with the identical command line editing interface.

Then there are the menu shells, programs that allow the user to initiate desired command sequences with just a few keystrokes.

Listing 1

```
 ; Memory display aliases

PBIOS=BIOS              p $ab
PCCP=CCP=CPR            p $ac
PDOS=DOS                p $ad
PENV=ENV                p $ae
PFCP=FCP                p $af
PIOP=IOP                p $ai
PMCL=MCL                p $al
PMSG=MSG                p $am
PNDR=NDR                p $an
PPATH                   p $ap
PRCP=RCP                p $ar
PSHL=PSHELL=SHL=SHELL   p $as
PXFCB=XFCB=PFCB=FCB     p $ax

 ; Special equivalents

ZP,ATCH                 b0:zpatch $*
CR,UNCH                 b0:crunch $d1$u1:$:1.$.1 $d2$u2:
DATSW,EEP=DS=SWEEP      b0:datsweep $*
LDIR                    $d1$u1:;b0:ldir $:1;$hb:
SYSBLD                  b0:;b0:$0 $1.mdl;$hb:
XBOOT=BOOT              b0:;b0:xboot $1.mdl

 ; Complete set of direct equivalents

CMDRUN=LPUT=EDIT0=ERA=IF=REN=SD=SDD=XD=ZEX=ZF=VLU=W=ZPATCH=COPY=ECHO    b0:$!
FF=GOTO=JF=JETLDR87=NULU=PWD=SAVE=SP=UNCR=VTYPE=XDIR=AFIND=SALIAS=AREA b0:$!
BD=CRUNCH=DFA=DIFF=DISKRST=DOSERR=DU=EDITNDR=ERRSET=HSH=ALIAS           b0:$!
LLF=LGET=LOADNDR=LPUT14=LT23=LUSH=LX=MOVE=MU3=PATH=PAUSE=PIP=PROT       b0:$!
PROTCCP=PUBLIC=PUTDS=Q=SAP=SAVENDR=SAVSTAMP=SFA=SHCTRL=SHOW=SQ=STAT     b0:$!
SUB=SYSGEN=UF=UNERASE=XSUB=DATE=DATSWEEP=MKDIR=SHSET=TPA=Z3INS=Z3LOC    b0:$!
ZRIP=ASSGN=BSX=FVCD=HDINIT=HDUTIL=MDINIT=MPTEST=SETDFLT=STARTHD4=SWX    b0:$!
SYSBLD=XSYSGEN=TIME=XBOOT0=XVERS=MCOPY=LZED=PUTBG=STARTHD=STARTHD1      b0:$!
STRTFULL=LDR=JETLDR=LHC=SSTAT=XBOOT=MAP=STARTBIG=LT=LDIR=QL=SETD=CRC    b0:$!
4ERA=4MU3=4REN=4SAVE=T4MAKE=DOSVER=DRO=LOGGED=SRO=SRW=LBREXT            b0:$!

SLR180+=SLRNK=SLRNK+=Z80ASM=DDT=DSDZ=FORM7=MAKESYM=MLOAD=SLRMAC=XIZ     b1:$!
ZAS=ZLINK=ZXLATE=SLR180=180FIG32=180FIG+=LNKFIG+=SLRIB=ZLIB=ZREF        b1:$!
ZZCNFG=LNKFIG=180FIG                                                    b1:$!

BGSERIAL=LOADBG=STARTBG=BGPRINT=BGPRNCFG=DSCONFIG=Q=REMOVE=SECURE       b2:$!
SETBG=SPOOLER=DATSWEEP=SDD=SETTERM                                      b2:$!

INSTALL=MEX                                                             b3:$!

WSCHANGE=WS=WINSTALL=MOVEPRN                                            b3:$!
```

Listing 1. The second half of the ALIAS.CMD file from my SB180 with XBIOS, slightly shortened and rearranged.

Listing 2

```
; Program:      WSSHLFIX
; Author:       Jay Sage
; Date:         March 26, 1988

; This code is a configuration overlay to correct a problem in the shell
; handling code in WordStar Release 4.
;
; Problem:  WS takes a mistaken shortcut when installing its name on the shell
; stack.  If the stack is currently empty, it does not bother to push the
; entries up.  However, when it exits, it does pop the stack, at which point
; any garbxge that had been in the stack becomes the active shell.  This patch
; makes sure that the second stack entry is null in that case.

;---- Addresses

initsub equ     03bbh
exitsub equ     03b3h
morpat  equ     045bh

;---- Patch code

        org     initsub         ; Initialization subroutine patch

init:   jp      initpatch

;----

        org     morpat          ; General patch area

initpatch:                      ; Initialization patch
        ld      hl,(109h)       ; Get ENV address
        ld      de,1eh          ; Offset to shell stack address
        add     hl,de           ; Pointer th shell stack address in HL
        ld      e,(hl)          ; Address to DE
        inc     hl
        ld      d,(hl)
        ld      a,(de)          ; See if first entry is null
        or      a
        ret     nz              ; If not, we have no problem
        inc     hl              ; Advance to ENV pointer to
        inc     hl              ; ..size of stack entry
        ld      l,(hl)          ; Get size into HL
        ld      h,0
        add     hl,de           ; Address of 2nd entry in HL
        ld      (hl),0          ; Make sure that entry is null
        ret

        end
```

Listing 2. Source code for a patch to fix the bug in the coding of shell stack pushing and popping in WordStar Release 4.

Listing 3

```
; Program:      WSSHLOFF
; Author:       Jay Sage
; Date:         March 26, 1988

; This code is a configuration overlay to correct a problem in the shell
; handling code in WordStar Release 4.
;
; Problem:  Because WordStar runs as a ZCPR3 shell, it is impossible to use
; WS in a multiple command line with commands intended to execute after WS is
; finished.  One can disabme this by patching the ZCPR3 environment to show
; zero entries in the shell stack while WS is running.  This effectively
; disables WS4's shell capability.  Unfortunately, it means that the extended
; features of the 'R' command under ZCPR3 are also lost.

;---- Addresses

initsub equ     03bbh
exitsub equ     03bvh
morpat  equ     045bh

;---- Patch code

        org     initsub         ; Initialization subroutine

init:   jp      initpatch

;----

        org     exitsub         ; Un-initialization subroutine

exit:   jp      exitpatch

;----

        org     morpat          ; General patch area

initpatch:                      ; Initialization patch
        call    getshls         ; Get pointer to shell stack number
        ld      a,(hl)          ; Get number
        ld      (shstks),a      ; Save it for later restoration
        ld      (hl),0          ; Set it to zero to disable shells
        ret

exitpatch:                      ; Termination patch
        call    getshls         ; Get pointer to shell stack number
shstks  equ     $+1             ; Pointer for code modification
        ld      (hl),0          ; Value supplied by INITPATCH code
        ret

getshls:
        ld      hl,(109h)       ; Get ENV address
        ld      de,20h          ; Offset to numbvr of shell entries
        add     hl,de           ; HL points to numbvr of shell entries
        ret

        end
```

Listing 3. Source code for a patch that disables the shell feature of ZCPR3 while WordStar 4 is running and reenables it on exit.

Table 1

WSSHLFIX patch bytes:

initialization subrcutine:	C3 5B 04
un-initialization subroutine:	00 00 C9 (should be this way already)
general patch area:	2A 09 01 11 1E 00 19 5E 23 56 1A B7 C0 23 23 6E 26 00 19 36 00 C9

WSSHLOFF patch bytes

initialization subroutine:	C3 5B 04
un-initialization subroutine:	C3 65 04
general patch area:	CD 6B 04 7E 32 69 04 36 00 C9 CD 6B 04 36 00 C9 2A 09 01 11 20 00 19 C9

Table 1. List of HEX bytes for installing either of the patches into WordStar Release 4 to deal with the problems in the shell code.

They come in several flavors. MENU stresses the on-screen menu of command choices associated with single keystrokes. VFILER and ZFILER stress the on-screen display of the files on which commands will operate; the commands associated with keys are not normally visible. Z/VFILER offer many internal file maintenance commands (copy, erase, rename, move, archive). VMENU and FMANAGER are in-between. Both the files in the directory and the menu of possible commands are shown on the screen.

What Kind of Programs Should be Shells?

Not all programs should be shells. From a strict conceptual viewpoint, only programs that are intended to take over the command input function from the command processor on a semipermanent basis should be shells. The history shells and the MENU and VMENU type shells clearly qualify. One generally enters those environments for the long haul, not just for a quick command or two.

ZFILER and VFILER are marginal from this viewpoint. One generally enters them to perform some short-term file maintenance operations, after which one exits to resume normal operations. It is rare, I believe, to reside inside ZFILER or VFILER for extended periods of time, though I am sure there are some users who do so.

Many people (I believe mistakenly) try to set up as shells any program from which they would like to run other tasks and automatically return. This is the situation with WordStar. No one will claim that the main function of WordStar is to generate command lines! Clearly it is intended to be a file editor. Why, then, was it made into a ZCPR3 shell in the first place? I'm really not sure.

WordStar's 'R' command really does not offer very much. In neither the ZCPR nor the CP/M configuration does any information about the operating environment seem to be retained. For example, one might expect on return to WordStar that the control-R function would be able to recall the most recently specified file name. But this does not seem to be the case, although it could easily have been done. In the ZCPR version, the name could be assigned to one of the four system file names in the environment descriptor; in the CP/M version it could be kept in the RSX code at the top of the TPA that enables WordStar to be reinvoked after a command is executed.

The WordStar 'R' command does not save any time, either. Essentially no part of WordStar remains in memory. The user could just as well use the 'X' command to leave WordStar, run whatever other programs he wished, and then reinvoke WS. Nevertheless, I can understand why users would enjoy the convenience of a command like the 'R' command that automatically brings one back to WordStar. Shells, however, are not the way to do this, at least not shells in the ZCPR3 sense.

ZCPR2-Style Shells

In ZCPR2 Richard Conn had already implemented an earlier version of the shell concept which, interestingly enough, would be the appropriate way for WordStar and perhaps even ZFILER/VFILER to operate. He did not have a shell stack, but he did have programs like MENU that, when they generated commands, always appended their own invocation to the end of the command line. Thus if the menu command script associated with the 'W' key was "WS fn2", where fn2 represents system file name #2, then the actual command placed into the command line buffer would be "WS fn2;MENU". In this way, after the user's command ran, the MENU program would come back.

Let's compare how the two shell schemes would have worked with WordStar. Suppose we want to edit the file MYTEXT.DOC and then copy it to our archive disk with the command "PPIP ARCHIVE:=MYTEXT.DOC". We might have created the following alias script for such operations:

```
WSWORK      ws $1;ppip archive:=$1
```

Then we just enter the command "WSWORK MYTEXT.DOC" when we want to work on the file and have it backed up automatically when we are done.

Here is what WS4 does as a ZCPR3-type shell. The command line starts out as:

```
WSWORK MYTEXT.DOC
```

When the alias WSWORK is expanded the command line becomes:

```
WS MYTEXT.DOC;PPIP ARCHIVE:=MYTEXT.DOC
```

When WordStar runs, it pushes its name onto the shell stack so that it will be invoked the next time the command line is empty. Noting that the command line is not empty, it returns control to the command processor. Then the PPIP command is executed, backing up our unmodified file (horrors!!!) Finally the command line is empty and WS, as the current shell, starts running. Since it was invoked as a shell, it prompts the user to press any key before it clears the screen to start editing. By this time it has forgotten all about the file we designated and it presents us with the main menu. All in all, a rather foolish and useless way to go about things.

You might think that the problem would be solved if WS did not check for pending commands but went ahead immediately with its work. Indeed, this would work fine until the 'R' command was used. Then either the pending PPIP command would be lost (replaced by the command generated by the 'R' operation) or executed (if the 'R' command appended it to the command it generated). In either case we have disaster!

Now suppose WS4 had used the ZCPR2-style shell concept. After the alias had been expanded, the "WS MYTEXT.DOC" command would run, and we would edit our file. While in WS4, suppose we want to find where on our disks we have files with names starting with OLDTEXT. We use the 'R' command to enter the command line "FF OLDTEXT". The 'R' command would append ";WS" to the end the command we entered and insert it into the command line buffer before the current pointer, leaving the following string in the buffer:

```
FF OLDTEXT;WS;PPIP ARCHIVE:=MYTEXT.DOC
```

After the FF command was finished, WordStar would be executed again. Just what we wanted.

In fact, under ZCPR3 WS could be much cleverer than this. First of all, it could determine from the external file control block the name (and under Z33 the directory) used to invoke WordStar in the first place. There would be no need, as there is now, to configure WS to know its own name and to make sure that the directory with WS is on the command search path. The 'R' command could have appended "B4:WSNEW" if WSNEW had been its name and it had been loaded from directory B4.

There is one problem, however. We would really like WS to wait before clearing the screen and obliterating the results of the FF command. With the ZCPR3-type shell, WS can determine from a flag in the ZCPR3 message buffer whether it was invoked as a shell. For the ZCPR2-style shell we would have to include an option on the command line. WS could, for example, recognize the command form "WS /S" as a signal that WS was running as a shell. It would then wait for a key to be pressed before resuming, just as under a ZCPR3-style shell. Of course, you would not be able to specify an edit file with the name "/S" from the command line in this case, but that is not much of a sacrifice or restriction.

We could continue to work this way as long as we liked. Only

when we finally exited WS with the 'X' command would the PPIP command run. This, of course, is just the right way to operate!

ZCPR2 vs ZCPR3 Shell Tradeoffs

Once I started thinking about the old ZCPR2-type shells, I began to wonder why one would ever want a ZCPR3-type shell. At first I thought that Z2-style shells could not be nested, but that does not seem to be the case. Suppose we run MENU and select the 'V' option to run VFILER. The command line at that point would be

```
VFILER;MENU /S
```

where we have assumed that a "/S" option is used to indicate invocation as a shell. While in VFILER we might run a macro to crunch the file we are pointing to. The macro could spawn the command line "CRUNCH FN.FT". The command line buffer would then contain

```
CRUNCH FN.FT;VFILER /S;MENU /S
```

After the crunch is complete, VFILER would be reentered. On exit from VFILER with the 'X' command, MENU would start to run. Thus nesting is not only possible with Z2-type shelling, it is not limited by a fixed number of elements in the shell stack as in ZCPR3 (the standard limit is 4). Only the size of the command line buffer would set a limit.

What disadvantages are there to the Z2-style shell? Well, I'm afraid that I cannot come up with much in the way of substantial reasons. The shell stack provides a very convenient place to keep status information for a program. I do that in ZFILER so that it can remember option settings made with the 'O' command. On the other hand, this information could be kept as additional flags on the command line, as with the "/S" option flag. There is no reason why the information could not be stored even in binary format, except that the null byte (00 hex) would have to be avoided.

If the 128 bytes currently set aside for the shell stack were added to the multiple command line buffer, the use of memory would be more efficient than it is now with Z3-style shells. Z3 shells use shell stack memory in fixed blocks; with Z2 shells the space would be used only as needed. I rarely have more than one shell running, which means that most of the time 96 bytes of shell stack space are totally wasted. Of course, with the present setup of ZCPR3, the multiple command line buffer cannot be longer than 255 bytes, because the size value is stored in the environment descriptor as a byte rather than as a word. The command line pointer, however, is a full word, and so extension to longer command lines would be quite possible (I'll keep that in mind for Z35!).

Following this line of reasoning, I am coming to the conclusion that only programs like history shells and true menu shells should be implemented as ZCPR3-style shells. Other programs, like ZFILER and WordStar should use the ZCPR2 style. If I am missing some important point here, I hope that readers will write in to enlighten me.

Forming a Synthesis

So long as the command line buffer is fixed at its present length and so long as 128 bytes are set aside as a shell stack, one should make the best of the situation. Rob Wood has come up with a fascinating concept that does just that.

Rob was working on Steve Cohen's W (wildcard) shell. He recognized that on many occasions one wants to perform a wildcarded operation followed by some additional commands (just as with the WordStar example followed by PPIP). As a ZCPR3-type shell, W could not do this. It always executed what it was supposed to do after the wild operation before the wild operation!

Rob came up with a brilliant way to combine the ZCPR2 and ZCPR3 shell concepts. When his version of W is invoked manually by the user, it pushes its name, as a good ZCPR3 shell does, onto the shell stack. But it does not then return to the command processor to execute commands pending in the command line. It starts running immediately, doing the thing it was asked to do and using the shell stack entry to maintain needed data.

In the course of operation, however, it does one unusual thing. After each command that it generates and passes to the command line buffer, it appends its own name, as a good ZCPR2 shell does. This command serves as a separator between the shell-generated commands and those that were on the original command line after the W command. After the shell-generated commands have run, W starts to run. It checks the top of the shell stack, and if it finds its own name there, it says "Aha, I'm a shell", and proceeds to use the information in the shell stack to generate the next set of commands. This process continues until W has no more work to do. Then it pops its name off the shell stack and returns to the command processor. The commands originally included after the W command are still there and now execute exactly as intended. Beautiful!

WordStar Shell Bugs

It is bad enough that WordStar's conceptual implementation as a shell is flawed. On top of that, the shell code was not even written correctly. The person who wrote the code (not MicroPro's fault, I would like to add) tried to take a short cut and flubbed it. When a shell installs itself, it should always—I repeat, always—push itself onto the stack. WordStar tries to take the following shortcut. If it sees that the shell stack is currently empty, it just writes its name into the first entry, leaving the other entries as they were.

When WordStar terminates, however, it pops the stack. At this point whatever junk was in the second shell stack entry becomes the currently running shell. The coding shortcut (which I would think took extra code rather than less code, but that is beside the point) assumed that if the current shell stack entry was null, all the others would be, too. But this need not be the case at all. And in many cases it has not in fact been the case, and very strange behavior has been observed with WordStar. Some users have reported that WordStar works on their computers only if invoked from a shell! That is because WordStar properly pushes itself onto the stack in that case.

There are basically two strategies one can take for dealing with the shell problems in WordStar. One is to fix the above problem and live with the other anomalies (just don't ever put commands after WS in a multiple command line). The other is to disable the shell feature entirely.

To fix the bug described above, Rick Charnes wrote a program called SHELLINI to initialize the shell stack before using WordStar. On bulletin boards in the past both Rick and I presented aliases that one can use to disable the shell stack while WS is running and to reenable it after WS has finished. I will now describe patches that can be made directly to WordStar itself. First I will explain what the patches do; later I will discuss how to install them.

Listing 2 shows a patch I call WSSHLFIX that will fix the bug just described. The code assumes that you do not already have any initialization or termination patches installed. If you do, you will have to add the routines here to the ones you are already using.

The patch works as follows. When WS starts running, the initialization routine is called. It extracts the shell stack address from the ENV descriptor and goes there to see if a shell command is on the stack. If there is, no further action is required, since WS already works correctly in this case. If, on the other hand, the first shell entry is null, then the routine calculates the address of

(Continued on page 42)

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 6:
- Build High Resolution S-100 Graphics Board: Part 1
- System Integration, Part 1: Selecting System Components
- Optronics, Part 3: Fiber Optics
- Controlling DC Motors
- Multi-User: Local Area Networks
- DC Motor Applications

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 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

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

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

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

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.

Issue Number 30:
• Double Density Floppy Controller: An algorithm for an improved CP/M BIOS.
• ZCPR3 IOP for the Ampro L.B.: Implementing ZCPR3 IOP support featuring NuKey, a keyboard re-definition IOP.
• 32000 Hacker's Language: How a working programmer is designing his own language.
• MDISK: Adding a 1 Meg RAM disk to Ampro L.B., part two.
• Non-Preemptive Multitasking: How multitasking works, and why you might choose non-preemptive instead of preemenptive multitasking.
• Software Timers for the 68000: Writing and using software timers for process control.
• Lilliput Z-Node: A remote access system for TCJ subscribers.
• The ZCPR3 Corner
•The CP/M Corner
• The Computer Corner

Issue Number 31:
• Using SCSI for Generalized I/O: SCSI can be used for more than just hard drives.
• Communicating with Floppy Disks: Disk parameters and their variations.
• XBIOS: A replacement BIOS for the SB180.
• K-OS ONE and the SAGE: Demystifing Operating Systems.
• Remote: Designing a remote system program.
• The ZCPR3 Corner: ARUNZ documentation.
• The Computer Corner

Issue Number 32:
• Language Development: Automatic generation of parsers for interactive systems.
• Designing Operating Systems: A ROM based O.S. for the Z81.
• Advanced CP/M: Boosting Performance.
• Systematic Elimination of MS-DOS Files: Part 1, Deleting root directories & an in-depth look at the FCB.
• WordStar 4.0 on Generic MS-DOS Systems: Patching for ASCII terminal based systems.
• K-OS ONE and the SAGE: Part 2, System layout and hardware configuration.
• The ZCPR3 Corner: NZCOM and ZCPR34.

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

(Continued from page 16)

in the form of a spreadsheet with rows and columns, where the fields are the columns, the rows are the records, and the whole sheet is the file. Data for the file STKBASE might look like this:

Company	Date	Close$	Change$
IBM	5/13/88	110.375	-0.125
IBM	5/20/88	109.50	-0.875
ALLTEL	5/13/88	32.125	+0.50
ALLTEL	5/20/88	32.50	+0.375
COMPAQ	5/13/88	52.0	-0.875
COMPAQ	5/20/88	49.675	-2.375

The first row, showing that IBM closed at 110.375 on the 13th with a loss of 0.125 is a record.

Once the data is entered for the stocks of interest, you can retreive the information in many different ways. You could get the information for a given company between certain dates. You could get the information for all the companies for a given date. You could search for a company with the most positive (or the least negative) percentage change during the period. By working in conjunction with a second file which contains the date, the Dow Jones Average, and the change, you could search for the companies with the best performance during a market decline. I am not a stock analyst, so don't base your investment strategy on this simplified example!

Don't Reinvent the Wheel!

Being able to write your own program is very important when none of the commercial offerings suite your client's needs. But, at the same time, don't let your ego prevent you from using an existing product if it fills the need. I have found one program which I believe will replace a list management program that I was going to write. More details after I evaluate it.

Next Time

This has been a very brief introduction to the subject of databases and information processing. It may have gone over the heads of some of you, and been a rehash of well known facts for others. But, I had to start somewhere.

I intend on continuing with more technical information, including details on database types, tools, utilities, and reference material. I'll try to keep it general, rather than emphasizing one specific product, but there will be specific code examples when we compare different methods. There will also be companion articles which are more application specific.

Your input is needed! Your articles, letters, notes, comments, suggestions, and questions are welcome. ■

Data Programming Resources

This section will be greatly expanded, and more extensive reviews provided. This is a 'first look' at useful products. The products mentioned here are the ones I am currently using or have examined. I do not intend to publish unsupported press releases, but recommendations from readers will be included.

Reference Material

Programming in Clipper by Stephen J. Straley, Addison-Wesley (Disk Available). A very thorough book about a very useful tool. A must if you intend to use Clipper.

dBASE III PLUS Power User's Guide by Edward Jones, Osborne McGraw-Hill (Disk Available). A very useful book, with important information on how to do things with dBASE. Not just a repeat of the commands from the manual. Definitely recommended, a complete review will follow.

dBASE III Tips & Traps, by Andersen, Cooper, and Demsey, Osborne McGraw-Hill. After you are familiar with dBASE, this book will help you avoid the pitfalls or find out what went wrong. It supplements, but does not replace the user's manual.

dBASE III PLUS The Complete Reference, by Joseph-David Carrabis, Osborne McGraw-Hill (Disk Available). This large (745 page) book is where you go to find the complete details on commands and functions.

Simpson's dBASE Tips and Tricks, by Alan Simpson, Sybex (Disk Available). This book is organized by activities, with useful code examples. A very helpful book, definitely recommended.

Tools

Clipper dBASE compiler, by Nantucket. Compile dBASE code for faster operation, and to protect your source code—generate standalone programs without a royalty—include C or ASM sections—extensions and expanded capabilities. Recommended.

db_VISTA III, db_QUERY, db_REVISE, by Raima. A C programmer's delight! DBMS routines to incorporate into your C programs. Based on the network database model and B-tree indexing method. Expect to hear more about this.

Registered Trademarks

The following trademarks are acknowledged, and we apologize for any we have overlooked.

dBASE; Ashton-Tate, Inc.
Clipper; Nantucket Corp.
db_VISTA III, db_QUERY, db_REVISE; Raima.

ZCPR3 Corner

(Continued from page 39)

the beginning of the second shell entry and places a zero byte there. When this stack entry is popped later, it will be inactive.

Listing 3 shows a patch I call WSSHLOFF that will completely disable the shell feature of ZCPR3 while WS is running. It works as follows. When WS starts running, the initialization routine is called. It gets the number of shell stacks defined for the user's system in the ENV descriptor and saves it away in the termination code for later restoration. Then it sets the value to 0. WordStar later checks this value to see if the shell feature is enabled in ZCPR3. Since WordStar thinks that there is no shell facility, it operates the 'R' command as it would under CP/M. Later, on exit from WS, the termination routine restores the shell-stack-number so that normal shell operation will continue upon exit from WS.

The easiest way to install these patches is to assemble them to HEX files and use the following MLOAD command (MLOAD is a very useful program available from remote access systems such as Z Nodes):

```
MLOAD WS=WS.COM,WSSHLxxx
```

Substitute the name you use for your version of WordStar and the name of the patch you want to install. That's it; you're all done.

If you do not have MLOAD, you can install the patches using the patching feature in WSCHANGE. From the main menu select item C (Computer), and from that menu select item F (Computer Patches). From that menu, work through items C (initialization subroutine), D (un-initialization subroutine), and E (general patch area), installing the appropriate bytes listed in Table 1.

Summary

We have covered a lot of material this time. The issue of shells is a very tricky one, and I hope to hear from readers with their comments. I would also enjoy learning about interesting ARUNZ aliases that you have created. ■

(Continued from page 44)

talk to a keyboard, disk drives, and monitor. Rick VanNorman has put together the system using this design, plus some changes of his own, and is selling them as a complete system. They use a 3½ inch drive, a PC compatible keyboard and monitor. I saw one running at the FORTH INTEREST GROUP convention in San Jose last year. I got Rick to send me a copy of his ROM and boot disk with instruction for a small fee.

Unfortunately, it set for several months before I got enough money to buy the new RAMS ($15-$20 each—needed four) and other chips. The problem was time to make the changes. I finally made time, and spent several days building the system. The system didn't work completely and I wasn't sure at first if it was the AUGATS or my hurried time schedule causing the problem.

The Fixes

I tried a number of things and found about the timing of the system. In Charles Moores book he says that 125 NS RAMs will work at 5 MHZ. The problem was he also said that was assuming a 60/40 (60% low to 40% high) clock cycle. The new design starts with 10 MHZ clock and divides to 5 MHZ producing a 50/50 clock (equal low to high time). I made some changes and was able to get a 60/40 clock cycle which made the system almost work. You see I decided to save 5 dollars a chip and buy 125 NS RAMs, not the 100 NS that would normally be installed by Rick.

Like I said it almost worked, but no matter how I shifted the clock operation I was getting nowhere. I called Charles and talked to him about the problem. He indicated a number of troubles he had seen lately. One important problem was the quality of NOVIX chips has been going down and the need for faster return stack RAM has been occurring. The return stack gets accessed sometimes on the high clock cycle and for a 60/40 clock cycle you would need 80 NS RAMs. He also indicated that he had been using TI 74HC139 for address decoding as they were faster. It seems the TIs were actually meeting the minimum speed specification while other brands had just made the higher end of specifications (longer propagation time).

Well, the points about quality control going down hill didn't help me, other than making me feel I made a mistake in buying 125 NS RAMs. Charles gave me Rick VanNormans new phone number (Rick had to move unexpectedly) and so I called him. When I talked to Rick he pointed out several possibles and gave me some tips about testing the system. I didn't know how much of the system was in ROM and how much was on disk. The basic FORTH kernel is in ROM so that if it is reading the keyboard you can perform a number of tests. He had me try and fill memory that becomes the character generator. That didn't produce the desired results and we decided the best move was to take it to him if I couldn't find any 100NS RAMs locally.

I didn't find any RAM, so last weekend I went to his place and we changed out the RAMs. The unit then worked almost. Filling the memory worked somewhat but not properly. Rick then remembered that some of the NOVIX chips have the locating pin connected to the data bus. It really is a mistake to be connected but normally doesn't cause any problems. We cut my locating pin off and zippo a near working system. It responded properly now, except it still wouldn't boot the disk system. We tried a write operation and read the disk with his system and discovered an address problem. We eventually found that (a solder bridge under a socket and of my own doing) and now the system works.

What Next?

Now that I am running the system, it is easy to see how powerful the device is. I can look at all the FORTH code, which isn't a lot of code,and see how to do all the operations. You need to keep in mind that the NOVIX is updating the screen on every trace, doing the vertical and horizontal traces, reading and writing the disk drive, and reading the keyboard, as well as outputting to a printer when needed. There is only the NOVIX, RAM, ROM, clock circuit, and 3 glue chips. There is not a video controller or disk controller—none of those are needed.

Now there are a few things I don't like, one being that the screen goes away during disk accessing. The disk format is not compatible with anything. The item that bothers me the most is the cost of the devices. The NOVIX now costs about $200. The RAMs (100NS) are running typically $20. I have about $800 dollars (probably more) in the system. Rick is selling a complete system for around a $1000 (tested and running). In comparison, I can buy the incredibly slower PC clones, which can't do half what the NOVIX does, for less and have some good features like compatibility and commercial programs.

Conclusion

Where does this leave me, well I have a 6 MIPS computer that will be housed in a Plexiglas box (10 by 10 inches square) so people can see how few components are used. The pleasure of being on the leading edge of technology. A chance to become really good at FORTH and the NOVIX. So you ask what are my next projects? Well, I do plan on changing the ways the system uses inputs and outputs (a small keyboard and maybe a LCD screen), adding a serial line for modem work, and then I will get onto the next major project. That next big job is a better tutor program that will run on the PC. Yup, after all this work I am going where the money is, programs for the masses (clones).

I think the NOVIX is going to be fun playing with, but sometime the bills have to be paid, which is why I haven't been working on my major projects. Hopefully before the next issue is due I will get those projects started again. Till then keep hacking. ■

Charles Moore
COMPUTER COWBOYS
410 Star Hill Road
Woodside, CA 94062
(415) 851-4362

Rick VanNorman
Personal Forth Environment
35972 Brandywine Street
Newark, CA
(415) 795-0532

For books on the NOVIX:
OFFETE ENTERPRISES, INC.
1306 South B Street
San Mateo, CA 94402
(415) 574-8250

For manufacturing information:
NOVIX
19925 Stevens Creek Blvd.
Cupertino, CA 95014
(408) 996-9363

For a NOVIX on a PC compatible board:
Software Composers
210 California Avenue, Suite 1
Palo Alto, CA 94306
(415) 322-8763

FORTH, Inc.
111 N. Sepulveda Blvd.
Manhattan Beach, CA 90266
(800) 55-FORTH (outside California)
(213) 372-8493

THE COMPUTER CORNER

by Bill Kibler

I hope that most of our readers are not as busy as I have been. So far it has been a very busy year, and it seems like it has just started. I have had to make several unexpected trips, and have not had any time to work on my major projects.

A project which I have worked on this last month is completing my NOVIX system. As most of you remember I am a FORTH user and have even bought the NOVIX computer as sold by Charles Moore (the inventor of FORTH). The NOVIX runs FORTH directly which makes it one of the faster computers available. I have made some discoveries about its speed and operation which I think will prove interesting for all of us.

The NOVIX

The NOVIX is not your ordinary CPU. The unit is a gate array laid out in such a way that it will perform operations that are directly related to FORTH instructions. There are about 4,000 devices (transistors) in the NOVIX, where as the newer 68030 or 80386 are pushing 7 to 800,000 devices. With fewer devices the NOVIX is not slow or short on power.

The Novix gets it speed from simplistic design. For those who don't know FORTH, it is a stack oriented Reverse Polish Notation (RPN) language. It works on the principle of two stacks, one the data stack, and the other the return stack. Data is placed on the stack and arithmetic operations are then performed, leaving the results back on the stack. In the NOVIX the stacks are off the device allowing them to be any amount of 16 bit RAM you choose.

One point I need to make is that the NOVIX is a 16 bit system. It requires two separate 16 bit wide stack RAM arrays, as well as 64K of 16 bit memory. A number of parallel data lines are provided so that paging is possible for more memory usage. The secret of the device is how it uses the clock.

Normal CPUs will trigger off of the rising edge of the clock. That means that on every rising edge, internal devices are stepped and perform the next operation. An example of this would be moving data to a register. The first edge would load the instruction into the CPU. Then it would be decoded into the necessary sections of the CPU so that the next clock shift would output the address of the data. Then the next clock would load the data into the appropriate register. All totaled, 6 or 8 clock changes are needed to complete the instruction.

The NOVIX however is a gate array and as such does not work on the standard CPU principles. It uses the clock highs and lows separately. By that I mean the low portion of the clock pulse is for getting the data from memory, while the high portion is for doing the instruction. Almost all instructions can be performed in one clock cycle. It goes something like this. The clock goes low and all address lines have the address output for the static RAMS. During the low, the RAMS are fetching the data to the data bus. The clock goes high and uses the data on the data bus as the next instruction.

In the case of fetching data to the stack, the address of the data is placed on the address bus after the instruction decode time (high cycle). The low portion of the clock this time fetches the actual data and it is placed on the stack during the high portion of the clock. The operation continues like this throughout the program, using one complete clock cycle for most operations. The long fetches take 2 cycles total, which is four times less than our standard CPU.

The speed doesn't stop there, as the NOVIX is also a parallel processor. The insides of the NOVIX have several different paths that data can travel and be processed by. This works out that a quarter of the instructions can be combined into one single instruction. This is done in the compiler part of the operation and results in a 4MHZ device processing data at 5MIPS. Now lots of people are going to say that is impossible, but remember the clock performs two operations each cycle (fetch from memory on low, shift data internally on high).

Speed for What?

With all this speed what can the NOVIX do then, how about everything. What my time has been spent on is doing everything with my setup. The Charles Moore FK3 board is a simple two sided 3 inch by 5 inch board that has chips placed on both sides. This is possible by using AUGATS insert socket pins. These inserts come on a roll and are inserted into the holes of the board. The chips then go into the insert which nicks the side of the hole and makes contact.

I have mixed emotions about this idea and have replaced some of the inserts with sockets. This has caused some minor problems when upgrading as you can no longer get to the traces that need to be cut. The whole idea behind the design was to allow for easy and numerous changes to the design. To achieve this you need access to all traces and feed thrus. I personally plan on changing to solder in type sockets, something in the order of socket buses. These buses are just round sockets in strips and can be cut to any length and soldered in. They are connected together by plastic, which gives them the strength needed for removal and insertion of the chips.

The reason I don't like the inserts is not knowing if they made good contact or not. I have had a couple push through the board, as well as having trouble with them when removing chips. I am currently having problems with my system and am not really sure if the pins are all contacting properly. If the traces were just from point to point there wouldn't be any problem, but for data buses it is impossible to tell if one chip of several is not making contact. The reason I am getting in to this problem is trying to upgrade to a full computer system.

The board as designed and shipped will work as a serial auxiliary processor. That means that all contact with the system is over a serial line with another computer or terminal. The books that come with it do provide some sample FORTH programs to load and save blocks of data on the F83 system (pubLic domain FORTH). The ability to test and see the speed of the NOVIX is somewhat lost over the serial line.

Charles Moore has provided traces and design information to enable the system to

(Continued on page 43)